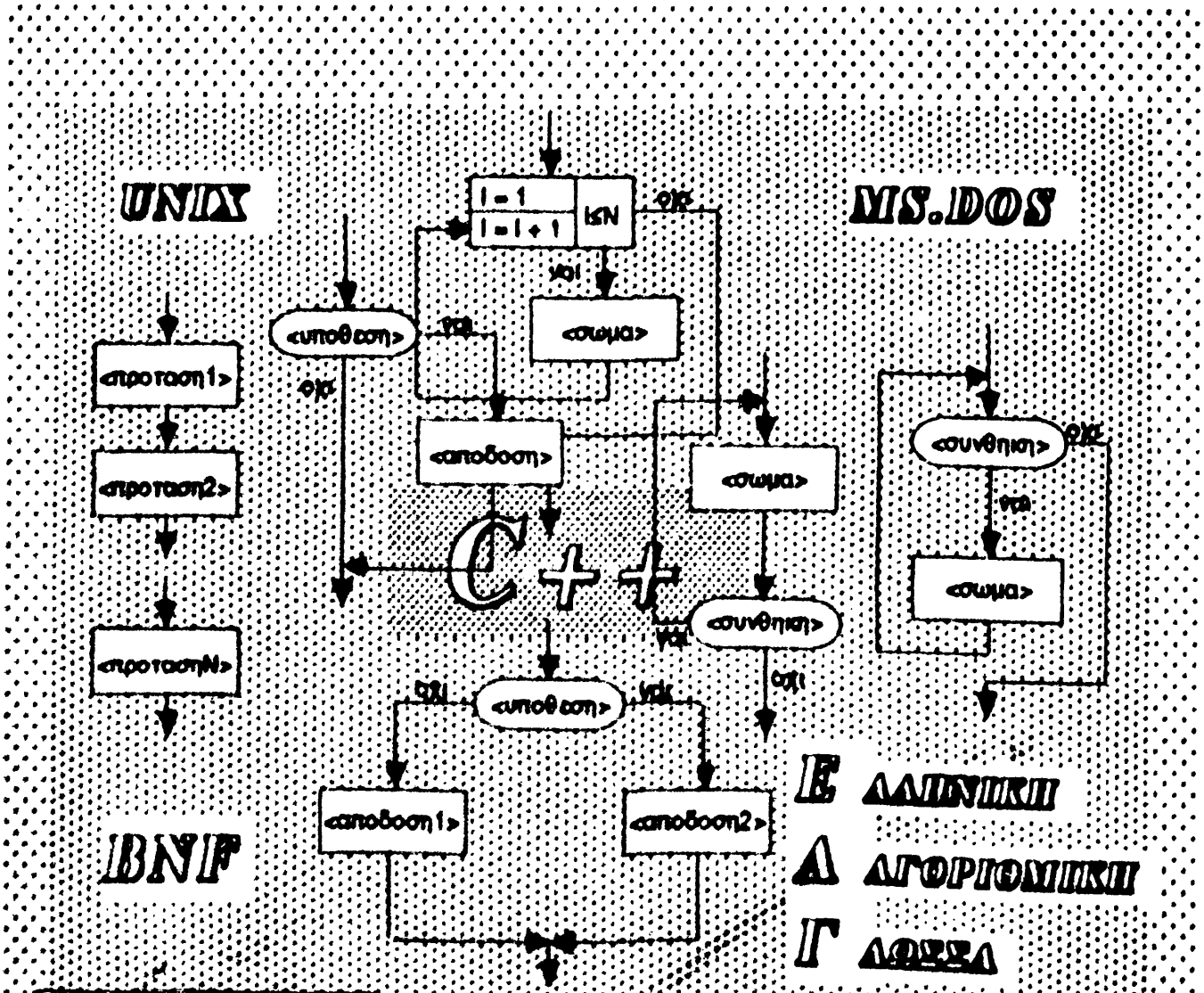
Σωκράτης Δ. Μπαλτζής
Δρ Επιστήμης Η/Υ - MSc Αριθμητικής Ανάλυσης
Μαθηματικός

ΕΙΣΑΓΩΓΗ

ΣΤΟΝ ΣΥΣΤΗΜΑΤΙΚΟ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟ ΚΑΙ

ΣΤΗ ΓΛΩΣΣΑ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ C++



ΕΚΔΟΣΕΙΣ
ΠΑΝΕΠΙΣΤΗΜΙΟΥ
ΙΩΑΝΝΙΝΩΝ

Σ.Δ.Μ.

Ιωάννινα 1995

Σωκράτης Δ. Μπαλτζής
Δρ Επιστήμης Η/Υ - MSc Αριθμητικής Ανάλυσης
Μαθηματικός

ΕΙΣΑΓΩΓΗ
ΣΤΟΝ ΣΥΣΤΗΜΑΤΙΚΟ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟ ΚΑΙ
ΣΤΗ ΓΛΩΣΣΑ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ C++

Ιούνιος 1995

ΕΙΣΑΓΩΓΗ ΣΤΟΝ ΣΥΣΤΗΜΑΤΙΚΟ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟ ΚΑΙ ΣΤΗ ΓΛΩΣΣΑ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ C++

Στη σύζυγό μου Δέσποινα,

για την ανεξάντλητη υπομονή και την αμέριστη συμπαράσταση στο δύσκολο έργο μου.

Το τέλειο δεν φθάνεται αλλά και το εφικτό δεν εξαντλείται
το διαχρονικό όμως κερδίζεται,
τελικά
η πράξη είναι ανάγκη, επιβάλλεται και κρίνεται.

Σ.Δ.Μ

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΕΦΑΛΑΙΟ 4: ΓΡΑΦΟΝΤΑΣ ΠΡΟΓΡΑΜΜΑ ΣΤΗΝ C++

ΠΡΟΛΟΓΟΣ

Μαθηματικές Επιστήμες και Επιστήμη Ηλεκτρονικών Υπολογιστών

Αναμφίβολα η νέα επιστήμη, η Επιστήμη των Ηλεκτρονικών Υπολογιστών (Ε.Η/Υ), εκπορεύεται από τη μαθηματική λογική και δεοντολογία την οποία και καταξιώνει στην πράξη. Η καταξίωση αυτή συνίσταται σε αυτούς καθεαυτούς τους στόχους της Ε.Η/Υ που είναι ο αυτόματος ή μηχανικός υπολογισμός της λύσης προβλημάτων. Το τελευταίο, κατά συνέπεια, υπαγορεύει νέα νοοτροπία στην έκφραση, στη διατύπωση δηλαδή των διαδικασιών που αφορούν την επίλυσή τους. Αυτή η νέα νοοτροπία επιβάλλει προδιαγραφές, έτσι ώστε κατά τρόπο ιεραρχημένο και αναλυτικό να περιγράφονται τα βήματα του εκάστοτε αυτόματου υπολογισμού που οδηγούν σε αυτό που ορίζουμε ως αλγόριθμο ή αλγοριθμική διατύπωση ή αλγοριθμοποίηση της διαδικασίας επίλυσης προβλημάτων.

Με την αλγοριθμοποίηση και τον αυτόματο τρόπο επίλυσης προβλημάτων η Ε.Η/Υ όχι μόνο ξεπληρώνει το χρέος της στις Μαθηματικές Επιστήμες, αλλά πηγαίνει πολύ πιό μακρυά, καθώς διεισδύει σε επιστήμες και εφαρμογές, διαδίδει και επιβάλλει τη μαθηματική λογική και δεοντολογία, ως υπέρτατη γεννήτρια δύναμη λογικής και δεοντολογίας, κατά συνέπεια διαμόρφωσης αφενός κριτηρίων προσέγγισης και χειρισμού γνώσης και εμπειρίας, αφετέρου εν γένει συμπεριφοράς.

Έτσι, με τα δύο νέα θεμελιώδη στοιχεία, την αλγοριθμοποίηση και τον αυτόματο τρόπο επίλυσης ή μηχανικής απόδειξης προβλημάτων, αλλά και τη σύγχρονη και αδιαμφισβήτητη αποστολή, άμεση ή έμμεση, των Μαθηματικών στη σύγχρονη πραγματικότητα, ανακατατάσσεται ο χάρος των Μαθηματικών Επιστημών και διαμορφώνονται ήδη τα Μαθηματικά της νέας χιλιετίας του 2,000.

Οριοθέτηση της Ύλης και Στόχοι

Με γνώμονα τα παραπάνω οριοθετείται η ύλη και καθορίζεται ο τρόπος παρουσίασης του

παρόντος συγγράμματος, που αποτελεί τον κύριο άξονα της διδασκαλίας του αντίστοιχου μαθήματος, πρώτου επιπέδου, "Εισαγωγή στον Συστηματικό Προγραμματισμό και στη Γλώσσα Προγραμματισμού C++". Το σύγγραμμα απευθύνεται κυρίως σε αρχάριους σε θέματα Ε.Η/Υ πρωτοετείς φοιτητές του Τμήματος Μαθηματικών, με ύλη που διαμορφώνεται:

1. Με τη γνώση της πραγματικότητας και της προοπτικής που δημιουργούν οι δύο επιστήμες, αυτή των Μαθηματικών και αυτή των Η/Υ, συνεργαζόμενες μαζί αλλά και η κάθε μια ξεχωριστά,

2. Με την επίγνωση που απορρέει από τη μακροχρόνια μελέτη και έρευνα, την πειραματική διδασκαλία σε όλα τα επίπεδα και διαστάσεις της εκπαίδευσης στους παραπάνω τομείς και

3. Με τη διάγνωση των ιδιαιτεροτήτων και των αναγκών των αρχάριων σε θέματα Ε.Η/Υ προαναφερθέντων φοιτητών.

Στόχος μας, λοιπόν, είναι η όσο το δυνατόν σφαιρικότερη και συνοπτικότερη παρουσίαση των εργαλείων του προγραμματισμού καθώς και η πληρέστερη εξοικείωση των φοιτητών με κριτήρια διαχρονικής γνώσης του χώρου του προγραμματισμού, όπως επίσης, ευέλικτης και αποτελεσματικής προσέγγισης στη θεωρία και στην πράξη αυτού καθεαυτού του προγραμματισμού και των δυνατοτήτων των εμπειριών του, που συνίστανται:

1. Στην ανάλυση των δεδομένων και στη σχεδίαση της διαδικασιακής στρατηγικής επίλυσης των πάσης φύσεως προβλημάτων Υπολογιστικής - Πληροφορικής,

2. Στη σύνθεση και στην περιγραφή της διαδικασίας επίλυσής τους με γλώσσες περιγραφής αλγορίθμων, όπως τα Λογικά Διαγράμματα και την Ελληνική Αλγοριθμική Γλώσσα, τη γνωστή ΕΑΓ, τέλος,

3. Στην κωδικοποίησή τους στη γλώσσα προγραμματισμού C++.

Παρουσίαση της Ύλης και Τρόπος Προσέγγισής της

Η ύλη του παρόντος ταξινομείται σε δύο ενότητες, κάθε ενότητα περιέχει δύο κεφάλαια, ενώ δύο παραρτήματα δίδονται στο τέλος συμπληρωματικά. Ακολουθεί μια παρουσίαση των ενοτήτων, των κεφαλαίων και των παραρτημάτων και προτείνεται ένας τρόπος προσέγγισής τους για διευκόλυνση των αναγνωστών-μελετητών.

Η ΑΙ ενότητα με τίτλο "**Συστηματικός Προγραμματισμός**" περιλαμβάνει τα βασικά εργαλεία του προγραμματισμού, δηλαδή τις γλώσσες, που αφορούν την έκφραση και την επικοινωνία προς όλες τις κατευθύνσεις του, π.χ. ψευδοκώδικες, κώδικες, κ.λ.π. Επίσης, περιλαμβάνει τις βασικές έννοιες και αρχές του συστηματικού προγραμματισμού καθώς και τη μεθοδολογία ανάπτυξης προγράμματος. Η ενότητα αυτή περιέχει παραδείγματα "βασικών αλγορίθμων", που διατυπώνονται σε λογικά διαγράμματα και σε ΕΑΓ καθώς και τα αντίστοιχα προγράμματά τους σε C++. Αυτοί οι αλγόριθμοι-προγράμματα περιγράφουν "βασικές πράξεις" της Υπολογιστικής - Πληροφορικής, οι οποίοι στα παραδείγματα αυτά, για λόγους διδακτικής απλότητας, εφαρμόζονται σε απλά αριθμητικά δεδομένα. Τέλος, την ενότητα συμπληρώνουν ερωτήσεις για εξάσκηση και εμπέδωση του θεωρητικού υπόβαθρού της και ασκήσεις για εξοικείωση, εμβάθυνση και διεύρυνση των μηχανισμών των "βασικών πράξεων" που προαναφέραμε.

Προτρέπουμε τους αναγνώστες-μελετητές να δώσουν ιδιαίτερο βάρος και προσοχή στους "απλούς" μηχανισμούς των "βασικών πράξεων", διότι με αυτούς σχηματίζονται άλλες πράξεις πιό σύνθετες, που αποτελούν τα θεμέλια σχεδόν όλων των εφαρμογών και των προεκτάσεών τους. Αντιπροσωπευτικά δείγματα υπό μορφή παραδειγμάτων-προγραμμάτων αλλά και άλυτων ασκήσεων, δίνονται στο τέταρτο κεφάλαιο.

Έτσι, αναλυτικά στο πρώτο κεφάλαιο, "**Τυπικές Γλώσσες και Προγραμματισμός**", μελετώνται εν τάχει οι τυπικές γλώσσες και τα χαρακτηριστικά τους. Τυπικές λέγονται οι γλώσσες που επινοούνται από τους ανθρώπους για συγκεκριμένο σκοπό κάθε φορά. Από τις τυπικές γλώσσες αναπτύσσονται διεξοδικότερα και με παραδείγματα εκείνες που έχουν άμεση σχέση με τον προγραμματισμό, γλώσσες δηλαδή που έχουν ακόμα πιό εξειδικευμένα και ιδιαίτερα χαρακτηριστικά και είναι:

1. Οι μεταγλώσσες, οι οποίες είναι γλώσσες που επινοήθηκαν και χρησιμοποιούνται για την περιγραφή και την παρουσίαση γλωσσών, π.χ. γλωσσών περιγραφής αλγορίθμων και προγραμματισμού,

2. Οι γλώσσες περιγραφής αλγορίθμων (ψευδοκώδικες), οι οποίες είναι γλώσσες που επινοήθηκαν και χρησιμοποιούνται για την περιγραφή και την παρουσίαση των αλγορίθμων με συστηματικό και τυπικά θεμελιωμένο τρόπο, και

3. Οι γλώσσες προγραμματισμού (κώδικες), οι οποίες είναι γλώσσες που επινοήθηκαν και χρησιμοποιούνται για την κωδικοποίηση (μετάφραση) των αλγορίθμων, ώστε αυτοί, δηλαδή οι αλγόριθμοι, να γίνονται αποδεκτοί - κατανοητοί από τους Η/Υ. Επίσης, σχετικά με τις γλώσσες προγραμματισμού δίνονται επιπλέον οι δυνατότητες για μια γενική ανασκόπηση της εξέλιξής τους, για μια συνοπτική παρουσίαση των θεμελειωδών εννοιών, αρχών και συστατικών των γλωσσών προγραμματισμού ανωτέρου επιπέδου, για μια ταξινόμηση ως προς τη χρήση τους και τέλος αρχές και κριτήρια αξιολόγησής τους.

Ο αναγνώστης - μελετητής διαβάζοντας πρώτα το κεφάλαιο αυτό, ή μελετώντας το, έστω και, μετά το δεύτερο κεφάλαιο, πέραν της εξοικείωσής του με τα εργαλεία έκφρασης του διαδικασιακού προγραμματισμού σε όλο τους το εύρος και την ποικιλία, αποκτά καταρχήν την ελάχιστη, ίσως, εμπειρία αλλά συγχρόνως την απαραίτητη γνώση και αναπτύσσει επαρκή κριτήρια για την επιλογή και αξιοποίηση των κατάλληλων για τη δουλειά του γλωσσών.

Το δεύτερο κεφάλαιο, "**Περί Συστηματικού Προγραμματισμού**", είναι το σημαντικότερο κεφάλαιο από άποψη θεμελίωσης του προγραμματισμού. Εφιστάται ιδιαιτέρως η προσοχή, η υπομονή, η επιμονή και η επιμέλεια των αναγνωστών-μελετητών στο κεφάλαιο αυτό. Οι αρχάριοι κυρίως, θα πρέπει να επανέλθουν πολλές φορές για να εξοικειωθούν, δηλαδή να καταλάβουν και να μάθουν:

1. να χειρίζονται με ευχέρεια,

(α) τα συστατικά - κανονικές βαθμίδες - με τα οποία συνθέτονται οι διαδικασίες επίλυσης προβλημάτων, οσοδήποτε πολύπλοκες και αν είναι, με άλλα λόγια πειθαρχία στις αρχές του συστηματικού προγραμματισμού,

(β) τις τρείς βασικές ενότητες - διάβασε (είσοδος δεδομένων), υπολόγισε, τύπωσε (έξοδος αποτελεσμάτων) - από τις οποίες αποτελείται κάθε αλγόριθμος,

(γ) τις γλώσσες περιγραφής αλγορίθμων, δηλαδή τα Λογικά Διαγράμματα (διαγραμματική γλώσσα) και την Ελληνική Αλγοριθμική Γλώσσα (ΕΑΓ),

2. τη μεθοδολογία ανάπτυξης προγράμματος, που περιλαμβάνει τρία στάδια:

(α) την ανάλυση του προβλήματος και τη σχεδίαση της διαδικασίας επίλυσής του,

(β) τη σύνθεση και τη διατύπωση του αντίστοιχου αλγορίθμου σε γλώσσα περιγραφής αλγορίθμων και τέλος

(γ) τη γραφή του προγράμματος, δηλαδή την κωδικοποίηση του αλγορίθμου σε γλώσσα προγραμματισμού, στην περίπτωσή μας στην C++.

Τα παραδείγματα του κεφαλαίου ολοκληρώνουν τα περί της μεθοδολογίας ανάπτυξης προγράμματος. Τα παραδείγματα αυτά είναι "βασικοί αλγορίθμοι-προγράμματα", οι οποίοι περιγράφουν "βασικές πράξεις" της Υπολογιστικής - Πληροφορικής και διατυπώνονται σε λογικά διαγράμματα, σε ΕΑΓ και σε C++. Για να γίνουν άμεσα κατανοητοί οι απλοί αλλά πολλοί βασικοί μηχανισμοί τους, τα παραδείγματα χρησιμοποιούν αριθμητικά δεδομένα. Οι μηχανισμοί αυτοί αντιστοιχούν σε βασικές πράξεις, δηλαδή πράξεις που δεν αναλύονται περεταίρω, αλλά με αυτές ορίζονται άλλες πιό σύνθετες, οι οποίες με τη σειρά τους αποτελούν τα θεμέλια σχεδόν όλων των εφαρμογών και των προεκτάσεών τους, όπως έχουμε ήδη προαναφέρει.

Προτρέπουμε τους αναγνώστες-μελετητές, για να αποκτήσουν την απαιτούμενη για το σημείο αυτό εμπειρία γραφής αλγορίθμων, να πάρουν μολύβι και χαρτί και να ασχοληθούν πολύ με τα παραδείγματα και τις άλυτες ασκήσεις. Όσοι έχουν μελετήσει αποτελεσματικά το κεφάλαιο θα πρέπει να λύσουν με ευκολία όλες τις άλυτες ασκήσεις, για οποιαδήποτε απορία ανατρέξτε και πάλι στα αντίστοιχα παραδείγματα του κεφαλαίου. Επίσης, για οποιαδήποτε διευκρίνιση ή απορία που αφορά την Ελληνική Αλγοριθμική Γλώσσα (ΕΑΓ), συμβουλευτείτε το ΑΙ Παράρτημα που αναφέρεται στον τρόπο που αυτή ορίζεται και χρησιμοποιείται. Στα παραρτήματα θα αναφερθούμε αναλυτικά παρακάτω.

Η ΒΙ ενότητα με τίτλο **"Εισαγωγή στη Γλώσσα Προγραμματισμού C++"** περιλαμβάνει καταρχήν μια συνοπτική ιστορική αναδρομή και περιγραφή της C++, καθώς και όσα πρέπει και χρειάζεται να γνωρίζετε στην παρούσα εισαγωγική φάση, για να γίνει ο (διαδικασιακός) προγραμματισμός σε C++ εύχρηστο, αποτελεσματικό, χρήσιμο και με προοπτική εργαλείο στη δουλειά σας. Δηλαδή, ο αρχάριος προγραμματιστής μαθαίνει πλέον αναλυτικά και άμεσα τα συστατικά της C++, πλαισιωμένα με πολλά παραδείγματα που αναφέρονται στη λειτουργία και στη χρήση τους. Τα παραδείγματα αυτά στην πραγματικότητα είναι η κωδικοποίηση σε C++ των επεκτάσεων σε διάφορες εφαρμογές, των παραδειγμάτων του δευτέρου κεφαλαίου, που περιγράφουν τους "μηχανισμούς βασικών πράξεων Υπολογιστικής - Πληροφορικής". Έτσι, ο αρχάριος προγραμματιστής μπορεί να κωδικοποιεί με ασφάλεια, άμεσα απλά και αποτελεσματικά σε C++

οποιουσδήποτε αλγορίθμους, αφού έμαθε, στο δεύτερο κεφάλαιο, πως να τους σχεδιάζει, πως να τους συνθέτει και πως να τους διατυπώνει σε Λογικά Διαγράμματα ή σε ΕΑΓ. Τέλος, όπως και προηγουμένως, την ενότητα συμπληρώνουν ερωτήσεις για εξάσκηση και εμπέδωση του θεωρητικού υπόβαθρού της και ασκήσεις για εξοικείωση, εμβάθυνση και εμπλουτισμό των εμπειριών στο προγραμματισμό.

Προτρέπουμε τους αναγνώστες-μελετητές στην ενότητα αυτή να δώσουν ιδιαίτερο βάρος και προσοχή στη κωδικοποίηση του προγράμματος, δηλαδή στην μετάφραση του αλγορίθμου σε C++ και στην υλοποίηση του προγράμματος στον Η/Υ.

Έτσι, αναλυτικά στο τρίτο κεφάλαιο, "**Εισαγωγή στην C++**", καταγράφεται συνοπτικά το ιστορικό της εξέλιξης της C++ και περιγράφονται τα χαρακτηριστικά της C++ ως υψηλής χαμηλού επιπέδου γλώσσας, δηλαδή, τα χαρακτηριστικά της C++ ως γλώσσας κατάλληλης για προγραμματισμό εφαρμογών (υψηλή) αλλά και προγραμματισμό λειτουργικών συστημάτων (χαμηλού επιπέδου). Το κεφάλαιο ολοκληρώνεται με τις καθιερωμένες ερωτήσεις οι οποίες στοχεύουν στο να συμπυκνώσουν και να αναδείξουν τα βασικά στοιχεία του κεφαλαίου.

Στο τέταρτο και τελευταίο κεφάλαιο, "**Γράφοντας Πρόγραμμα στην C++**", ο αρχάριος προγραμματιστής εισάγεται πλέον βήμα προς βήμα στον προγραμματισμό σε C++, βεβαίως με ασφάλεια και επιτυχία, μετά από όσα έχει μάθει μέχρι εδώ. Το κεφάλαιο αυτό είναι το συμπλήρωμα ή η συνέχεια του δεύτερου κεφαλαίου. Θα πρέπει λοιπόν, πρώτα να διαβαστεί, να μελετηθεί, να εμπεδωθεί επαρκώς το δεύτερο κεφάλαιο, διότι μόνο τότε η μελέτη του παρόντος κεφαλαίου και οι κωδικοποιήσεις προγραμμάτων γίνονται ευκολότερα, πληρέστερα και αποτελεσματικά, άρα συνειδητά.

Έτσι στο κεφάλαιο αυτό δίδεται καταρχήν η γενική μορφή προγράμματος στην C++. Ο αναγνώστης-μελετητής, ο οποίος είναι ήδη εξοικειωμένος από το δεύτερο κεφάλαιο με κωδικοποιημένους σε C++ αλγορίθμους, συγκροτεί τη γνώση του προσεγγίζοντας τώρα με σχήματα τη γενική μορφή προγράμματος C++, ενός μοναδικού πηγαίου αρχείου και τον σκελετό απλού προγράμματος C++. Επίσης με τη παρουσίαση και τη διεξοδική ανάλυση των συστατικών ενός δείγματος προγράμματος σε C++, το οποίο αποτελεί μια γενική μορφή προγραμματισμού σε C++, του δίνεται η δυνατότητα να κάνει τη σύνδεσή του και με

τα παραδείγματα των κωδικοποιημένων σε C++ αλγορίθμων του δευτέρου κεφαλαίου.

Ο "έμπειρος" πλέον αναγνώστης-μελετητής ή αρχάριος προγραμματιστής είναι ώριμος να μελετήσει αναλυτικά και αποτελεσματικά το λεξιλόγιο και τα συστατικά της C++, τα οποία ουσιαστικά αντιμετωπίζονται ως ειδικές περιπτώσεις - παραδείγματα των όσων τα κεφάλαια ένα και δύο περιέχουν.

Με τον όρο συστατικά της C++ εννοούμε τα κατασκευάσματα που παριστούν τις ενέργειες τις οποίες περιγράφει η γλώσσα ή τα υπολογιστικά αντικείμενα τα οποία χειρίζεται. Αυτά είναι τα δεδομένα και η περιγραφή τους, οι τελεστές, οι εντολές, οι δηλώσεις, οι οριοθέτες, η δομή προγραμμάτων και η εισαγωγή δεδομένων - εξαγωγή αποτελεσμάτων. Τα συστατικά αυτά παρουσιάζονται με τρόπο αυστηρά τυπικό αλλά γιαυτό απλό, κατανοητό και άμεσα εφαρμόσιμο. Τα συστατικά αυτά πλαισιώνονται με πολλά επιλεγμένα παραδείγματα, τα οποία αναφέρονται στη λειτουργία και στη χρήση τους, αλλά υποβοηθούν και ολοκληρώνουν αυτό που ονομάσαμε βήμα προς βήμα εισαγωγή στον προγραμματισμό με C++. Τα παραδείγματα αυτά είναι προγράμματα (κώδικες) σε C++ ποικίλων εφαρμογών Υπολογιστικής - Πληροφορικής, των οποίων οι πράξεις, στην πραγματικότητα, είναι συνθέσεις ή επεκτάσεις των "βασικών πράξεων" των παραδειγμάτων του δευτέρου κεφαλαίου.

Έτσι, ο αρχάριος προγραμματιστής, μετά την εμπέδωση στην σχεδίαση και διατύπωση σε Λογικά Διαγράμματα ή σε ΕΑΓ οποιωνδήποτε αλγορίθμων, μπορεί να τους κωδικοποιεί με ασφάλεια, άμεσα, απλά και αποτελεσματικά σε C++.

Επιβάλλεται η συστηματική ενασχόληση με τις ερωτήσεις για εξάσκηση και εμπέδωση του θεωρητικού υπόβαθρου της κάθε παραγράφου του κεφαλαίου. Επίσης επιβάλλεται η επίλυση όλων των ασκήσεων για εξοικείωση, εμβάθυνση και εμπλουτισμό των εμπειριών στο προγραμματισμό.

Προτρέπουμε τους αναγνώστες-μελετητές να τολμήσουν, να καθίσουν στον Η/Υ να πληκτρολογήσουν και να τρέξουν όλα τα προγράμματα. Να ξεκινήσουν από τα πολύ απλά παραδείγματα, να επιμείνουν με υπομονή και να συνεχίσουν με επιμονή και πειθαρχία δοκιμάζοντας, αν είναι δυνατόν, όλα τα παραδείγματα και τις άλυτες ασκήσεις.

Σημειώνουμε ότι, κατά την ανάπτυξη των κεφαλαίων, αλλά και στο Παράρτημα ΒΙ, το οποίο αναφέρεται στη χρήση του Η/Υ, οι μελετητές-προγραμματιστές θα βρούν πολλές και χρήσιμες πληροφορίες για τη γραφή, την υλοποίηση στον Η/Υ των προγραμμάτων και οπωσδήποτε τη χρήση του Η/Υ.

Με τα δύο παραρτήματα ολοκληρώνουμε την παρουσίαση της ύλης και τον τρόπο προσέγγισής της.

Το πρώτο παράρτημα"**Συντακτική Περιγραφή της Ελληνικής Αλγοριθμικής Γλώσσας (ΕΑΓ)**", είναι ένας συνοπτικός οδηγός για τη γραφή αλγορίθμων σε ΕΑΓ. Το παράρτημα περιέχει τις εξής παραγράφους:

1. Εισαγωγή,
2. Δομή της Διαδικασίας,
3. Πρόταση Ανάθεσης, 4.Υποθετικές Προτάσεις, 5. Βαθμίδες Επανάληψης,
6. Κλήσεις Διαδικασιών, 7. Εισαγωγή Δεδομένων - Εξαγωγή Αποτελεσμάτων και
8. Παραδείγματα Περιγραφής Αλγορίθμων.

Ο αναγνώστης-μελετητής στις παραγράφους αυτές θα βρεί μια σύντομη αναδρομή στην ιστορία της ΕΑΓ και κατά τρόπο τυπικό και μεθοδικό, με την βοήθεια της πιό απλής και ευρέως διαδεδομένης μεταγλώσσας BNF (Backus-Noar Form), την παρουσίαση όλων των κανόνων της. Σημειώνουμε ότι οι κανόνες της, παρόλο που η ΕΑΓ έχει υλοποιηθεί ως γλώσσα προγραμματισμού, στο παρόν καταγράφονται και παρουσιάζονται μόνο ως κανόνες γλώσσας περιγραφής αλγορίθμων, δηλαδή γλώσσας μη υλοποιήσιμης.

Η χρήση του παραρτήματος ως οδηγού για τη γραφή αλγορίθμων σε ΕΑΓ είναι πολύ απλή, ακόμα και για τους αρχάριους, φυσικά οι πρωτόπειροι θα πρέπει πρώτα να διαβάσουν και να μελετήσουν τα δύο πρώτα κεφάλαια του συγγράμματος, δίνοντας έμφαση στις μεταγλώσσες και στα παραδείγματα αλγορίθμων σε ΕΑΓ αντίστοιχα. Επίσης, στο παράρτημα, στην τελευταία παράγραφο, περιέχονται χρήσιμα παραδείγματα περιγραφής αλγορίθμων, τα οποία είναι επιλεγμένα και συμπληρώνουν την ποικιλία των λυμένων και άλυτων ασκήσεων του παρόντος συγγράμματος.

Στο δεύτερο και τελευταίο παράρτημα "**Πρώτη Γνωριμία με τον Προσωπικό**

Η/Υ - PC", περιλαμβάνονται τα εξής:

1 Γενική Ανασκόπηση του Η/Υ,

2 Το Υλικό,

2.1 Η Μονάδα Συστήματος και η Μνήμη, 2.2 Η Αποθήκευση σε Δίσκο,

2.3 Η Οθόνη, 2.4 Ο Εκτυπωτής, 2.5 Το Πληκτρολόγιο, 2.6 Το Ποντίκι, 2.7 Το Μόντεμ,

3 Το Λογισμικό,

3.1 Τα Προγράμματα και τα Δεδομένα, 3.2 Το Λειτουργικό Σύστημα MS-DOS.

Με το παράρτημα αυτό γίνεται μια προσπάθεια ώστε οι αρχάριοι, στο ξεκίνημά τους, να αποκτήσουν μια πρώτη εικόνα για βασικά και στοιχειώδη πράγματα που αφορούν τον προσωπικό Η/Υ - PC σε όλο το φάσμα του. Σίγουρα η ενημέρωσή τους δεν τελειώνει εδώ, απλώς μόλις αρχίζει. Το παράρτημα αυτό διαβάζεται ανεξάρτητα, πριν όμως οι αρχάριοι καθίσουν στον Η/Υ.

Στο σημείο αυτό, μετά την ανάλυση των κεφαλαίων και παραρτημάτων του συγγράμματος, θα θέλαμε να επιστήσουμε την προσοχή των αναγνωστών-μελετητών σε κάποιες ακόμα διαστάσεις του:

1. στους καθιερωμένους αγγλικούς όρους, μέσα σε παρενθέσεις, που υπάρχουν δίπλα σε όλους τους ελληνικούς όρους του συγγράμματος,

2. στους ορισμούς που δίδονται έμμεσα στο κείμενο ή άμεσα στις υποσημειώσεις, για κάθε έννοια του χώρου της Ε.Η/Υ που αναφέρεται στο σύγγραμμα βοηθητικά,

3. στις οδηγίες, στις συμβουλές και στις πάσης φύσεως χρήσιμες πληροφορίες που παρέχονται και έχουν άμεση σχέση με τον συστηματικό προγραμματισμό και την αποτελεσματική υλοποίησή του.

4. στη βιβλιογραφία, ελληνική και διεθνή, η οποία, για λόγους καλλίτερης χρήσης εκ μέρους των αναγνωστών-μελετητών, χωρίζεται ως προς το περιεχόμενό της σε τρείς ενότητες, με τους τίτλους που ακολουθούν:

(α) **"Συστηματικός Προγραμματισμός και Γλώσσες Προγραμματισμού"**,

(β) **"Γλώσσες Προγραμματισμού C, C++ και UNIX"**,

(γ) **"Εισαγωγικά και Γενικού Περιεχομένου Η/Υ, Εγκυκλοπαίδειες και Λεξικά Η/Υ"**.

Πέραν όμως των παραπάνω, χρήσιμες βιβλιογραφικές πληροφορίες δίδονται επίσης στις

υποσημειώσεις, όπου στο σύγγραμμα απαιτούνται εξαιρετικά για επί τόπου ειδική γνώση και πληροφόρηση, κυρίως για ερευνητικής υφής ή εν γένει επιστημονικού ενδιαφέροντος θέματα.

Τελειώνοντας θέλουμε να τονίσουμε, για μια φορά ακόμα, ότι το σύγγραμμα είναι εισαγωγικό πρώτου επιπέδου, στην παρούσα φάση τουλάχιστον, κρίναμε σκόπιμο, να μην συμπεριλάβουμε τα αρχεία και τις δυνατότητες της γλώσσας C++ για αντικειμενοστρεφή προγραμματισμό, όπως και άλλες λεπτομέρειες που αφορούν ήδη καταγραμμένα στοιχεία της γλώσσας. Η απόπειρα επέκτασης της ύλης ίσως γίνει σε μελλοντική έκδοση του παρόντος, με παράλληλη διαμόρφωση της αντίστοιχης διδασκαλίας.

Η Προοπτική

Με τις παραστάσεις, τις εμπειρίες, τις γνώσεις που αποκτά ο αρχάριος αλλά και τον προβληματισμό που του γεννάται, διαγράφεται αδιαμφισβήτητα η προοπτική ώστε να χρησιμοποιηθούν τα ανωτέρω όχι μόνο ως γνώσεις και εμπειρία εν γένει για προγραμματισμό, αλλά και επιμέρους, ως γνώσεις και εμπειρία για ανάλυση και σχεδίαση της διαδικασιακής στρατηγικής επίλυσης προβλημάτων, ως γνώσεις και εμπειρία για σύνθεση και περιγραφή της διαδικασίας επίλυσης προβλημάτων γενικά με γλώσσες περιγραφής αλγορίθμων αλλά και ειδικά με την ΕΑΓ, τέλος, ως γνώσεις και εμπειρία για κωδικοποίηση των διαδικασιών επίλυσης προβλημάτων σε οποιαδήποτε γλώσσα προγραμματισμού, κυρίως όμως στην C++. Όλα αυτά βεβαίως επ' ωφελεία αυτών καθεαυτών των σπουδών και του επαγγελματικού προσανατολισμού των προαναφερθέντων φοιτητών, πολύ δε περισσότερο των Μαθηματικών Επιστημών και όχι μόνο.

Σωκράτης Δ. Μπαλτζής

ΚΕΦΑΛΑΙΟ 1

ΤΥΠΙΚΕΣ ΓΛΩΣΣΕΣ ΚΑΙ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ

1.1 Εισαγωγή

Ο **προγραμματισμός** (programming) είναι η μέθοδος με την οποία ο άνθρωπος-χρήστης του Η/Υ (user) καθοδηγεί τον Η/Υ για τον αυτόματο υπολογισμό της λύσης ενός προβλήματος ή όπως λέμε της υπολογιστικής επίλυσής του. Η σύνθετη αυτή νοητική διαδικασία έχει σαν βασικό προϊόν ένα πρόγραμμα ή ένα σύνολο προγραμμάτων.

Το **πρόγραμμα** (program) είναι ένας κωδικοποιημένος σε γλώσσα που αποδέχεται/αντιλαμβάνεται ο Η/Υ αλγόριθμος. Δηλαδή αφενός, αλγόριθμος είναι μια διαδικασία (procedure) επίλυσης ενός προβλήματος, που εκφράζει κατά τρόπο ιεραρχημένο και αναλυτικό την περιγραφή των βημάτων για τον αυτόματο ή μηχανικό υπολογισμό της λύσης του· αφετέρου, πρόγραμμα[1] είναι ένας αλγόριθμος μεταφρασμένος (κωδικοποιημένος) σε γλώσσα προγραμματισμού.

Όπως είναι φυσικό και αναμενόμενο, κατά τα διάφορα στάδια της προαναφερθείσας νοητικής διαδικασίας χρησιμοποιούνται κατάλληλες για κάθε περίπτωση επινοηθείσες γλώσσες για την εν γένει έκφραση και τελική επικοινωνία του ανθρώπου-χρήστη του Η/Υ με τον Η/Υ. Σημειώνουμε ότι, οι γλώσσες που επινοούνται από τον άνθρωπο για συγκεκριμένους κάθε φορά σκοπούς, βάσει κανόνων, οι οποίοι όχι μόνο παράγουν[2] (γεννούν) αλλά και ελέγχουν[3] (αναγνωρίζουν) τη γλώσσα, λέγονται **τυπικές ή τεχνητές**

[1] Για τον λόγο αυτόν το πρόγραμμα λέγεται και κώδικας.

[2] Σημαίνει ότι, εφαρμόζοντας τους κανόνες παράγονται λέξεις (οι προσδιοριστές - identifiers), πέραν του λεξιλογίου της γλώσσας (δεσμευμένες λέξεις - reserved words) και οι προτάσεις της γλώσσας (statements). Οτιδήποτε παράγεται από την εφαρμογή των κανόνων είναι στοιχείο της γλώσσας, και αυτό, γιατί στις τυπικές γλώσσες η σύνταξη και η σημασιολογία ταυτίζονται.

[3] Σημαίνει ότι με τη βοήθεια των κανόνων παραγωγής ελέγχουμε αν μια αλυσίδα (λέξη ή πρόταση) ανήκει ή όχι στη γλώσσα, δηλαδή αν είναι σωστά διατυπωμένη. Αρχή ή διαδικασία που εφαρμόζεται σε ένα πρόγραμμα, για τον έλεγχο της ορθότητας των λέξεων του (λεξική ανάλυση) ή της ορθότητας της σύνταξης των προτάσεών του (συντακτική ανάλυση) που συντελείται κατά τη μεταγλώττιση (compilation) ή τη διερμήνευσε (interpretation) του, κατά τη διάρκεια μετατροπής του από πηγαίο πρόγραμμα (source program) σε αντικείμενο πρόγραμμα (object program) για να "τρέξει".

γλώσσες[4] (formal languages). Επισημαίνουμε ότι για κάθε τυπική γλώσσα, έχει εισαχθεί ένα μαθηματικό μοντέλο παραγωγής και ελέγχου της, που λέγεται **γραμματική**[5] (grammar).

Οι κατηγορίες των τυπικών γλωσσών που εμπλέκονται στον προγραμματισμό είναι οι **γλώσσες περιγραφής αλγορίθμων** (languages of algorithms description) οι **μεταγλώσσες** (metalanguages) και τέλος οι **γλώσσες προγραμματισμού** (programmaing languages).

Έτσι στις παραγράφους που έπονται, δίνουμε συνοπτικά και περιεκτικά ορισμούς, θεμελιώδεις έννοιες, βασικές αρχές, συνήθεις ταξινομήσεις και αντιπροσωπευτικά παραδείγματα, για τις γλώσσες περιγραφής αλγορίθμων, τις μεταγλώσσες και τις γλώσσες προγραμματισμού, που θεωρούμε απαραίτητα για την πληρέστερη παρουσίαση και καλλίτερη κατανόηση του αντικειμένου μας, που σημαίνει ευκολία στον χειρισμό και την εφαρμογή του.

1.2 Γλώσσες Περιγραφής Αλγορίθμων

Πριν προχωρήσουμε στη σύντομη παρουσίαση των γλωσσών περιγραφής αλγορίθμων επιβάλλεται να αναλύσουμε τη λέξη αλγόριθμος, αφού η έννοιά της και πολύ σημαντική είναι στην Ε.Η/Υ ή των αυτομάτων ή μηχανοποιημένων υπολογισμών, αλλά και οδήγησε στην επινόηση των γλωσσών αυτών.

- Η λέξη **αλγόριθμος** (algorithm) έχει την έννοια μιας υπολογιστικής διαδικασίας, δηλαδή μιας σειράς πεπερασμένων οδηγιών, για την εκτέλεση ενός υπολογισμού.
- Η σειρά αυτή των οδηγιών, εφαρμοζόμενη σ' ένα συγκεκριμένο σύνολο δεδομένων εισόδου, συνεπάγεται μια πεπερασμένη ακολουθία δράσεων, η ακολουθία αυτή έχει μια μοναδική αρχική δράση.

---

[4] Σε αντίθεση οι φυσικές γλώσσες (natural languages), δηλαδή οι γλώσσες όπως η μητρική μας κλπ., η γραμματική της περιγράφει (η γλώσσα προϋπάρχει της γραμματικής), καθώς επίσης η σύνταξη και η σημασιολογία τους δεν ταυτίζονται, είναι ξεχωριστές οντότητες.

[5] Οι γραμματικές λειτουργούν ως γεννήτριες γλωσσών. Αντίστοιχα με τις γραμματικές εισάγονται και τα μαθηματικά μοντέλα των αποδεκτών των γλωσσών, τα λεγόμενα αυτόματα (automata) και αφηρημένες μηχανές (machines). Η υλοποίηση μιας πολύπλοκης μορφής εξελιγμένης μηχανής, της περίφημης "Μηχανής Turing" (κάθε εξέλιξη της μηχανής αυτής είναι η ίδια η μηχανή) είναι ο γνωστός σε όλους μας σημερινός Η/Υ.

- Κάθε δράση ακολουθείται και πάλι από μια μοναδική δράση, τερματισμό[6] των δράσεων έχουμε, είτε με μια λύση του προβλήματος, είτε με τη δήλωση ότι το πρόβλημα είναι άλυτο.

Σημειώνουμε ότι ένας αλγόριθμος μπορεί να περιγραφεί κατά ποικίλους τρόπους. Οι συστηματικά και τυπικά θεμελιωμένοι τρόποι περιγραφής αλγορίθμων ορίζουν τις γλώσσες περιγραφής τους. Οι γλώσσες αυτές ονομάζονται και **ψευδοκώδικες**[7] (pseudocode), αφού η διατύπωση ενός αλγορίθμου σε ψευδοκώδικα είναι "ίσως" μια μη εκτελέσιμη διαδικασία, αλλά αποτελεί σίγουρα ένα κείμενο, που συνθέτεται από προτάσεις συντακτικά και σημασιολογικά αποδεκτές, καθώς και από λέξεις δεσμευμένες και ειδικά σύμβολα της συγκεκριμένης γλώσσας (η οποία φυσικά στηρίζεται στις αρχές των τυπικών γλωσσών).

Η **Ελληνική Αλγοριθμική Γλώσσα (ΕΑΓ)**[8] είναι μια τέτοιου τύπου γλώσσα (βλπ παράρτημα 1), που χρησιμοποιεί ελληνικές προτάσεις. Ακολουθεί ένα απλό παράδειγμα αλγορίθμου στην ΕΑΓ, που περιγράφει τη διαδικασία αθροίσματος δύο ακεραίων αριθμών:

```
διαδικασια ΑΘΡΟΙΣΜΑ ;
δηλωση (Α, Β, Γ) ακερ ;
αρχη
  διαβασε Α, Β ;
  Γ ← Α + Β ;
  τυπωσε Γ ;
τελος ΑΘΡΟΙΣΜΑ
```

Παρατηρούμε ότι στο παραπάνω κείμενο: (α) Οι άτονες ελληνικές λέξεις οι

---

[6] Μια εξεραιτικά ενδιαφέρουσα παρατήρηση είναι ότι τυπικά στην Ε.Η/Υ γίνεται διάκριση μεταξύ του αλγορίθμου και του προγράμματος. Ένα πρόγραμμα δεν τερματίζει απαραιτήτως, δηλαδή δεν πληροί απαραιτήτως αυτό που η παρούσα πρόταση ορίζει. Για παράδειγμα το λειτουργικό σύστημα (operating system) του υπολογιστή, είναι ένα πρόγραμμα το οποίο δεν τερματίζει, συνεχίζει με φράγχο αναμονής (wait loop) έως ότου εισάγουμε νέα εργασία. Τερματισμό έχουμε μόνο με τη διακοπή παροχής ρεύματος προς τον Η/Υ, δηλαδή το σβήσιμο του Η/Υ ή την καταστροφή του συστήματος.

[7] Σε αναλογία με τον κώδικα.

[8] Η υλοποίηση της ΕΑΓ [3], δηλαδή η προσπάθεια ώστε η ΕΑΓ να χρησιμοποιείται και ως γλώσσα προγραμματισμού βρίσκεται σε εξέλιξη, μέχρι στιγμής μπορούν να τρέξουν χωρίς κανένα απολύτως πρόβλημα όλα τα απλά προγράμματα (χωρίς διαδικασίες και υπορουτίνες), κυρίως σε PC.

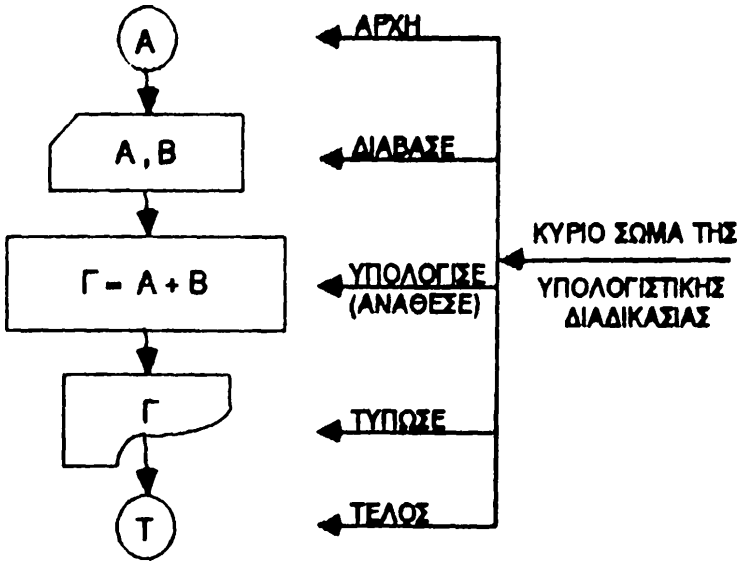
γραμμένες με μικρά έντονα γράμματα είναι οι δευσμευμένες λέξεις της γλώσσας, ενώ αυτές που είναι γραμμένες με κεφαλαία είναι οι προσδιοριστές. Οι προτάσεις της γλώσσας διατυπώνονται βάσει των κανόνων[9] (παραγωγής τους) που τις διέπουν (βλπ παράρτημα 1). (β) Το σώμα της υπολογιστικής διαδικασίας, δηλαδή η ενότητα που οριοθετείται από τις δεσμευμένες λέξεις **αρχη** και **τελος** περιλαμβάνει τρείς βσικές ενέργειες - ενότητες:

i) **διάβασε**, είναι το στάδιο που γίνεται η είσοδος των δεδομένων,

ii) **υπολόγισε** είναι το στάδιο που πραγματοποιούνται οι πάσης φύσεως υπολογισμοί,

iii) **τύπωσε**, είναι το στάδιο που συντελείται η έξοδος των αποτελεσμάτων.

Μια από τις πιό παραστατικές γλώσσες, είναι η **γλώσσα των Διαγραμμάτων Ροής**, ή **Λογικών Διαγραμμάτων** (flow charts). Η γλώσσα αυτή προκειμένου για μεγάλους αλγορίθμους είναι δύσχρηστη, άρα κατά κάποιο τρόπο πρακτικά ξεπερασμένη. Για διδακτικούς σκοπούς εντούτοις ενδείκνυται, γιατί επιτρέπει τη γρήγορη διαπίστωση της σύνθεσης και οργάνωσης ενός αλγορίθμου, και ακόμα την καλή σύνθεσή του σύμφωνα με τις αρχές του συστηματικού προγραμματισμού (βλπ κεφ.2). Το παράδειγμα αλγορίθμου που διατυπώθηκε προηγουμένως στην ΕΑΓ και που περιγράφει τη διαδικασία αθροίσματος δύο ακεραίων αριθμών, δίδεται πάλι, αλλά στη διαγραμματική γλώσσα περιγραφής αλγορίθμων, τα λογικά διαγράμματα ή διαγράμματα ροής του προγράμματος:



Πολλές φορές, πάλι, επιστήμονες ή συγγραφείς σχετικών άρθρων ή βιβλίων διατυπώνουν δική τους ειδική γλώσσα, για την περιγραφή αλγορίθμων, η οποία

[9] Οι κανόνες παραγωγής των προτάσεων είναι αναδρομικοί τύποι, έννοια γνωστή από τα Μαθηματικά.

ανταποκρίνεται στις ειδικές ανάγκες των προβλημάτων ή αλγορίθμων που ασχολούνται.

Επιβάλλεται η διατύπωση των αλγορίθμων σε μια κανονική γενική γλώσσα (βλπ κεφ.2), έστω και μη πραγματοποιημένη, όχι μόνο για λόγους διδακτικούς, αλλά και πρακτικούς. Ο προγραμματισμός σήμερα απαιτεί τη γραφή μεγάλων και πολύπλοκων προγραμμάτων, συστημάτων, γιαυτό επιβάλλεται η εξοικείωσή μας στον σχεδιασμό σωστών προγραμμάτων βάσει αρχών που υπαγορεύονται από τους κανόνες και τις αρχές του συστηματικού προγραμματισμού (βλπ κεφ.2).

Η διατύπωση των αλγορίθμων επιτρέπει στον προγραμματιστή να γράφει και να κοινοποιεί καλά γραμμένα και κανονικά (δομημένα) προγράμματα, χωρίς να υποβάλλει κανένα σε διαρκή εκνευρισμό, που συνεπάγεται η ανάγνωση και κατανόηση προγραμμάτων γραμμένων σε μη κανονικές γλώσσες. Ακόμα, όσο τα προγράμματα γίνονται πιό πολύπλοκα και οι γράφοντες το πρόγραμμα αγνοούν το στάδιο της σχεδίασής του, καταδικάζονται να χάνουν πολύτιμο χρόνο, να μπερδεύονται και στο τέλος να απογοητεύονται, καθώς τα προγράμματά τους δεν θα είναι καθόλου αποτελεσματικά.

Τέλος, όταν η σχεδίαση και ανάπτυξη ενός συστήματος (προγράμματος) συμπληρωθεί, τότε πιό μπορεί να μεταφραστεί στη γλώσσα που πρόκειται να εκτελεστεί.

1.3 Μεταγλώσσες

Μεταγλώσσες, είναι οι γλώσσες που επινοήθηκαν για να περιγράψουν ή να παρουσιάσουν γλώσσες προγραμματισμού και άλλες γλώσσες. Οι τυπικές αυτές γλώσσες έχουν ορολογία και σύμβολα που διαφέρουν από αυτά των γλωσσών που περιγράφουν. Από τις πιό γνωστές μεταγλώσσες για την περιγραφή των γλωσσών προγραμματισμού είναι η B.N.F και τα Συντακτικά Διαγράμματα.

Η **B.N.F.** (Backus Naur Form) είναι ίσως η πιό εύχρηστη και αξιοσημείωτη από τις μεταγλώσσες. Επινοήθηκε από τους J.W. Backus (ΗΠΑ) και P. Naur (Δανία), για την περιγραφή της Algol 60. Η BNF περιέχει μόνο 4 μετασύμβολα <, >, | και ::= . Το ζευγάρι των αγκυλών χρησιμοποιείται για την κατασκευή των μεταγλωσσικών μεταβλητών, οι οποίες αποτελούνται από άτονες απλές ή σύνθετες λέξεις, (οριοθετημένες) κλεισμένες

μεταξύ αυτών. Η κάθετος δηλώνει επιλογή, ενώ το τελευταίο, το σύνθετο σύμβολο σημαίνει ορίζεται ως (βλπ παράρτημα 1).

ΠΑΡΑΔΕΙΓΜΑΤΑ

Οι υποθετικές προτάσεις των : α. **ANS FORTRAN 77**, β. **ANS PASCAL** και γ. **C++**.

α. ANS FORTRAN 77

i) **IF** <λογικη εκφραση> **THEN** ή ii) **IF** <λογικη εκφραση> **THEN**
<αποδοση> <αποδοση 1>
ENDIF **ELSE** <αποδοση 2>
ENDIF

όπου :

<λογικη_εκφραση>::=<απλη_εκφραση><σχεσιακος_τελεστης><απλη_εκφραση>,

<σχεσιακος_τελεστης> ::= **.LT.** | **.LE.** | **.EQ.** | **.NE.** | **.GE.** | **.GT.**,

<απλη_εκφραση> ::= <προσημο><οροσ> |
<προσημο><οροσ><τελεστης><προσημο><οροσ>,

<προσημο> ::=+ | -, <οροσ> ::= . . . , <παραγοντας> ::= . . . , . . . , κλπ.,

<τελεστης> ::= + | - | **.OR.**,

<αποδοση> ::= <συνθετη_προταση>,

<συνθετη_προταση> ::= <προταση><συνθετη_προταση> |
<προταση>,

<προταση> ::= <αναθεση> | <αναγνωση> | <γραφη> | <ανακυκλωση> |
<υποθετικη> | <κληση_υποπρογραμματος> | **stop** | <format> | **end**,

β. ANS PASCAL

i) **if** <λογικη εκφραση> **then** ή ii) **if** <λογικη εκφραση> **then**
<αποδοση> <αποδοση 1>
else <αποδοση 2>

όπου :

<λογικη_εκφραση> ::= <απλη_εκφραση><σχεσιακος_τελεστης><απλη_εκφραση>,

<σχεσιακος_τελεστης> ::= < | <= | = | <> | >= | >,

<απλη_εκφραση> ::= <προσημο><οροσ> |
<προσημο><οροσ><τελεστης><προσημο><οροσ>,

<προσημο> ::= + | -, <ορος> ::= . . . , <παραγοντας> ::= . . . , . . . , κλπ.,

<τελεστης> ::= + | - | or,

<αποδοση> ::=<συνθετη_προταση>,

<συνθετη_προταση> ::= **begin** <προταση> ;<συνθετη_προταση> **end** | . . . |
<προταση> ; ,

<προταση> ::= <αναθεση> | <αναγνωση> | <γραφη> | <ανακυκλωση> |
<υποθετικη> | <κληση_υποπρογραμματος>.

γ. C++

i) **if** <λογικη εκφραση> ή ii) **if** <λογικη εκφραση>
{ <αποδοση> } { <αποδοση 1> }
else
{ <αποδοση 2> }

όπου :

<λογικη_εκφραση> ::= <απλη_εκφραση><σχεσιακος_τελεστης><απλη_εκφραση>,

<σχεσιακος_τελεστης> ::= < | <= | = | != | >= | >,

<απλη_εκφραση> ::= <προσημο><ορος> |
<προσημο><ορος><τελεστης><προσημο><ορος>,

<προσημο> ::= + | -, <ορος> ::= . . . , <παραγοντας> ::= . . . , . . . , <τελεστης> ::= . . . , κλπ.,

<αποδοση> ::= <συνθετη_προταση> ,

<συνθετη_προταση> ::= <προταση> ;<συνθετη_προταση> |
<προταση> ;,

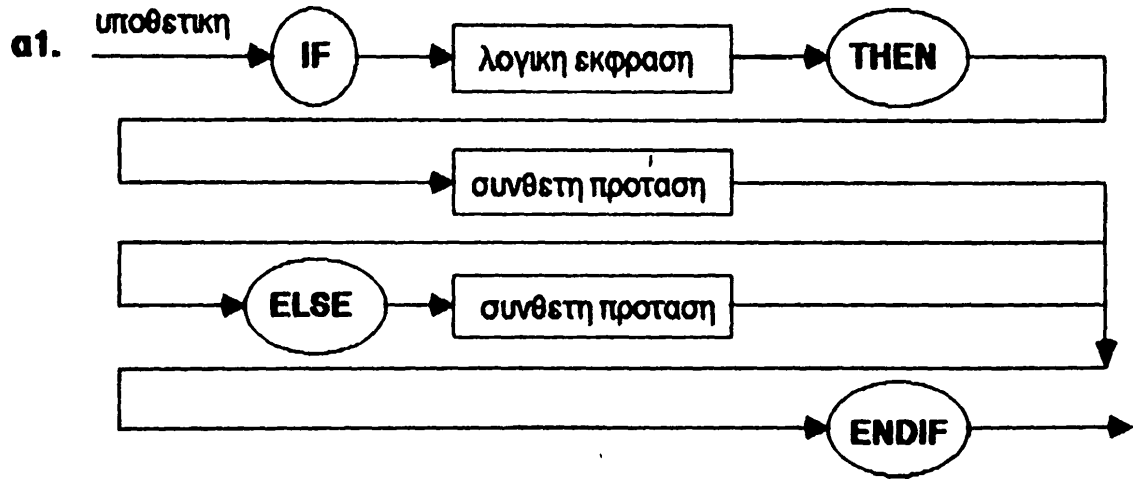
<προταση> ::= <αναθεση> | <αναγνωση> | <γραφη> | <ανακυκλωση> |
<υποθετικη> (βλπ §4.6.1.1 και §4.6.1.2).

Τα **Συντακτικά Διαγράμματα** είναι μια διαγραμματική μεταγλώσσα ισοδύναμη κατά κάποιο τρόπο της B.N.F. Η μέθοδος αυτή περιγραφής γλωσσών διατυπώθηκε από τον Niklaus Wirth, τον επινοητή της Pascal, για την περιγραφή της. Τα Συντακτικά Διαγράμματα αποτελούνται από ορθογώνια παραλληλόγραμμα και κύκλους για τον συμβολισμό των μεταβλητών και δεσμευμένων λέξεων ή ειδικών συμβόλων αντίστοιχα, τα πλαίσια αυτά συνδέονται μεταξύ τους με κατευθυνόμενες γραμμές ή αγκύλες.

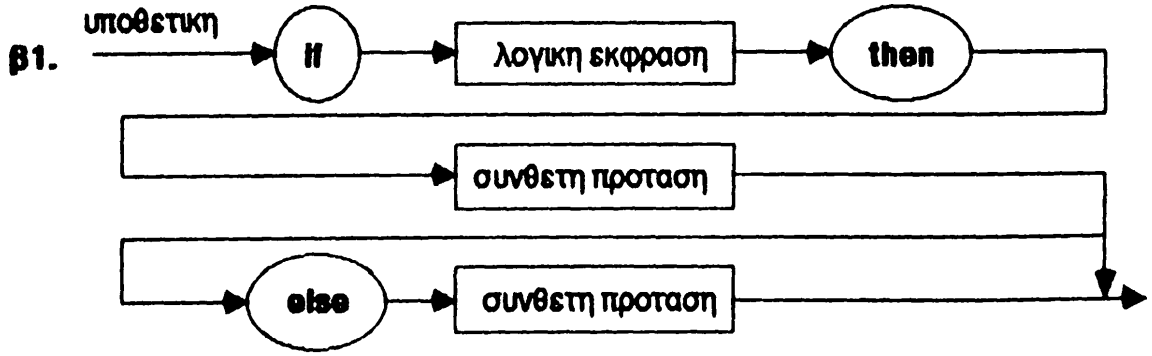
ΠΑΡΑΔΕΙΓΜΑΤΑ

Οι υποθετικές προτάσεις των : α. **ANS FORTRAN 77**, β. **ANS PASCAL** και γ. **C++**.

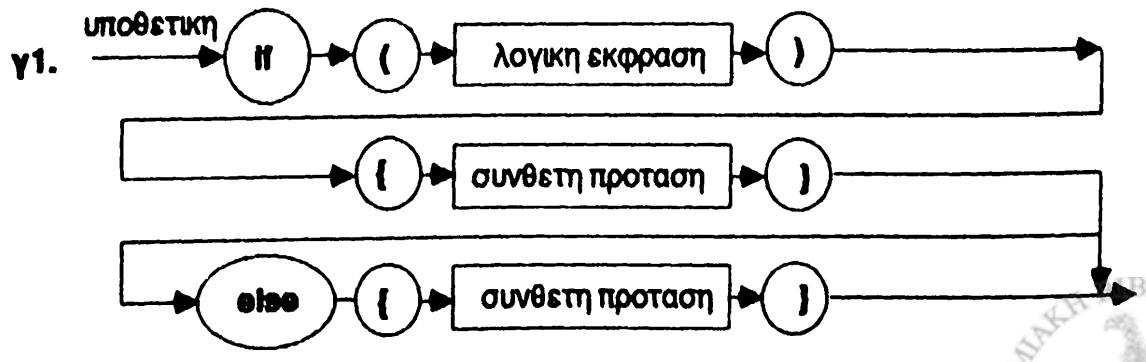
α. ANS FORTRAN 77



β. ANS PASCAL



γ. C++



1.4 Γλώσσες Προγραμματισμού

Είναι αλήθεια ότι από τις τυπικές γλώσσες που έχουν άμεση σχέση με τον προγραμματισμό οι Γλώσσες Προγραμματισμού είναι οι περισσότερο διαδεδομένες μεταξύ της μεγάλης πλειοψηφίας των χρηστών των Η/Υ, αποτελούν δε, για την ώρα τουλάχιστο, το κυριότερο μέσο επικοινωνίας του ανθρώπου με τον Η/Υ. Ας σημειώσουμε ότι η εκρηκτική ανάπτυξη της απλής αριθμομηχανής σε υπολογιστική μηχανή (μηχανή με μνήμη), αλλά και η συνεχιζόμενη εξέλιξη, της όλο και εξυπνότερης μηχανής, σε νοήμονα μηχανή, δρομολόγησαν και προκαλούν συνέχεια την εξέλιξη των μέσων επικοινωνίας του χρήστη και της μηχανής. Έτσι η αναγνώριση φωνής, γραμμάτων και εν γένει προτύπων, όπως και η δυνατότητα κατανόησης φυσικής γλώσσας από τη μηχανή, βάζει σε άλλους δρόμους, νέες εποχές τον προγραμματισμό. Για παράδειγμα, η ρομποτική και όχι μόνο, με τρομερές εφαρμογές στη διαστημική, στην ιατρική, στη βιομηχανία, στον αυτόματο τρόπο οικοδόμησης κτιρίων κλπ., επιβεβαιώνουν του λόγου το αληθές.

1.4.1 Γενική Ανασκόπηση

Οι γλώσσες προγραμματισμού μπορούν να θεωρηθούν ή να ταξινομηθούν τουλάχιστον ως προς δύο βασικές αρχές, ως προς το επίπεδο που ανήκουν και ως προς τη χρήση τους. Είναι συνήθη ότι οι θεωρήσεις αυτές των γλωσσών σχετλιάζονται από τις ιστορικές εξελίξεις του κλάδου, που μοιραία συμπαρασύρουν και τις γλώσσες. Υπάρχουν λοιπόν, τρία επίπεδα γλωσσών προγραμμμτισμού:

1.Οι **γλώσσες κατωτέρου επιπέδου** (low level languages), στις οποίες συμπεριλαμβάνονται οι (α) οι **γλώσσες μηχανής** (machine languages) και (β) οι **συμβολικές γλώσσες** (assembly). Είναι γλώσσες εξαρτημένες από τη μηχανή. Είναι πρωταρχικές και πρωτόγονες, εν γένει, γλώσσες επικοινωνίας του ανθρώπου που προγραμματίζει με τον Η/Υ. Οι γλώσσες αυτές αποτελούνται, οι μεν πρώτες από λέξεις ή προτάσεις που είναι κωδικοί αριθμοί, δηλαδή πρωτόγονα στοιχεία του λεξιλογίου της μηχανής, οι δε δεύτερες από συμβολικές λέξεις αντί για αριθμούς, εξών και το όνομά τους Η καινοτομία, που εισάγεται με τις συμβολικές γλώσσες, είναι ένα ειδικό πρόγραμμα που εφοδιάζεται το σύστημα, και ονομάζεται **συμβολομεταφραστής** (assembler). Το πρόγραμμα αυτό, μεταφράζει στη γλώσσα που η μηχανή καταλαβαίνει τα όσα γράψαμε σε συμβολική γλώσσα. Ο τρόπος με τον οποίο διατυπώνονται, και στις δύο γλώσσες, οι

διεξαγωγές αριθμητικών πράξεων απέχει από τον τρόπο που ο άνθρωπος τις διατυπώνει, εξ ού και ο χαρακτηρισμός κατωτέρου επιπέδου. Οι γλώσσες αυτές έχουν πολλά μειονεκτήματα. Για παράδειγμα, κάθε σειράς ή κατασκευαστικής εταιρίας μηχανή έχει τη δική της γλώσσα μηχανής ή τη συμβολική της γλώσσα, γεγονός που περιόριζε τον κάθε χρήστη στο να δουλεύει σ' έναν μόνον υπολογιστή, και απέκλειε συγχρόνως τη δυνατότητα στα προγράμματα να χρησιμοποιούνται σ' οποιονδήποτε Η/Υ. Εξάλλου η μεγάλη έκταση των προγραμμάτων, ακόμα και αυτών που επέλυαν απλά προβλήματα, σε συνδυασμό με το τεράστιο πλήθος των κωδικών, είχε σαν συνέπεια τις μεγάλες πιθανότητες λαθών, συντακτικών ή λογικών, κατά τον προγραμματισμό, και την εξίσου δύσκολη απολαθοποίησή τους. Το χειρότερο, η παραμικρότερη επέκταση ή βελτίωση του προγράμματος, απαιτούσε αλλαγή της δομής του.

2. Οι **γλώσσες ανωτέρου επιπέδου** (high level Languages), είναι οι γλώσσες που "καταρχήν" έκαναν τον Η/Υ προσπό (φιλικό) στον άνθρωπο που προγραμμάτιζε, αφού γεφύρωσαν το χάσμα μεταξύ του τρόπου επικοινωνίας του και του τρόπου που ο Η/Υ δέχεται τα ερεθίσματα που αντιστοιχούν σε συγκεκριμένα ανθρωπονοήματα-οδηγίες. Σε καμμιά περίπτωση ο άνθρωπος που προγραμμάτιζε δεν έπρεπε να κατεβαίνει στο επίπεδο της μηχανής, αλλά η μηχανή επιβαλλόταν να ανέβει όσο το δυνατόν περισσότερο στο επίπεδο έκφρασης του ανθρώπου. Επινοήθηκαν και εισάχθηκαν λοιπόν, στα "παρασκήνια", οι μεταφραστές, πιό συγκεκριμένα οι μεταγλωττιστές και οι διερμηνείς, ενώ στο "προσκήνιο" κυριάρχησαν οι γλώσσες προγραμματισμού ανωτέρου πιά επιπέδου[10]. Έτσι ο προγραμματισμός με γλώσσες ανωτέρου επιπέδου (ή ακόμα και με συμβολικές) προυποθέτει ένα είδος διασύνδεσης, αντιστοίχισης, μεταξύ αυτών και της γλώσσας μηχανής[11] του Η/Υ, για να τρέξει το πρόγραμμα. Δηλαδή **γράφεται πρώτα ο κώδικας** (πηγαίο πρόγραμμα), στη συνέχεια πραγματοποιείται η **διάγνωση** (diagnostic) (λεξική και συντακτική ανάλυση βλπ υποσημείωση 3), η **απολαθοποίηση** (debuging) και η **μετάφραση**[12] του σε γλώσσα μηχανής (αντικείμενο πρόγραμμα, που θα χρησιμοποιηθεί για

[10] Με τα χαρακτηριστικά που τους προσδόθηκαν οι γλώσσες αυτές ως "ανεξάρτητες μηχανών" τυγχάνουν της ευρύτερης χρησιμοποίησης σε σχέση με τις υπόλοιπες κατηγορίες γλωσσών, σε σημείο που να εννοούμε γλώσσες ανωτέρου επιπέδου όταν μιλούμε για γλώσσες προγραμματισμού.

[11] Η εξέλιξη των μηχανών σήμερα είναι τέτοια ώστε οι γλώσσες μηχανής τους είναι οι ίδιες γλώσσες ανωτέρου επιπέδου!!! Τέτοιο παράδειγμα είναι οι LISP μηχανές, που σχεδιάστηκαν από την Symbolics & XeroxCorporation.

[12] μεταγλώττιση - διερμηνεία

την εκτέλεση του προγράμματος), τέλος το **πρόγραμμα είναι έτοιμο να τρέξει.** Ως παραδείγματα τέτοιων γλωσσών δίνουμε το ελάχιστο και αντιπροσωπευτικότερο, ως προς τη χρήση τους, δείγμα των δημοφιλέστερων γλωσσών όπως είναι 1) η Pascal, η APL και η FORTRAN για επιστημονικές εφαρμογές, 2) η COBOL για επεξεργασία δεδομένων, 3) η SNOBOL και η Icon για επεξεργασία κειμένων (χαρακτήρων), 4) η LISP και η PROLOG για προγραμματισμό προβλημάτων ΤΝ (Τεχνητής Νοημοσύνης), 5) η C, η Ada και η Modula για προγραμματισμό συστημάτων και τέλος 6) η PL/1 ως γενικής χρήσης γλώσσα.

3. Οι **δηλωτικές ή ανωτάτου επιπέδου γλώσσες** (declarative languages). Οι γλώσσες αυτές αναπτύχθηκαν με το σκεπτικό να καταστίσουν εύκολη και γρήγορη την αφομοίωσή τους από επιδέξιους, διαφόρων κλάδων, επαγγελματίες, έτσι ώστε να τις χρησιμοποιούν οι ίδιοι στη δουλειά τους, χωρίς να διαθέτουν ιδιαίτερες γνώσεις στον προγραμματισμό αλλά και χωρίς να έχουν ανάγκη τους προγραμματιστές. Οι δηλωτικές γλώσσες (declarative languages) ως επί το πλείστον λειτουργούν και έχουν εκφραστικά μέσα ανάλογα αυτών της φυσικής, Αγγλικής, Γλώσσας. Έτσι συνιστούν (για την ώρα) το ανώτατο επίπεδο γλωσσών προγραμματισμού. Οι γλώσσες αυτές κυριαρχούνται από προτάσεις (statements) που μας επιτρέπουν να εκφράσουμε μάλλον το "τι θα κάνουμε" παρά το "πώς θα το κάνουμε", γιαυτό και στη βάση τους είναι γλώσσες προστακτικές (command languages). Παραδείγματα δηλωτικών γλωσσών είναι i) η SAS και η SPSS, ii) η NATURAL και η IMS, κά, γλώσσες προσανατολισμένες στη Στατιστική οι πρώτες στη Βάση Δεδομένων οι δεύτερες κλπ.

1.4.2 Θεμελειώδεις Έννοιες Αρχές και Συστατικά των Γλωσσών Ανωτέρου Επιπέδου

Δεν υπάρχει ενιαίος τρόπος ορισμού των Γλωσσών Προγραμματισμού, όμως, οι γλώσσες αυτές, πρέπει να εμφανίζουν κοινά χαρακτηριστικά. Τα χαρακτηριστικά αυτά προσδίδουν στις γλώσσες γνωρίσματα γλωσσών ανωτέρου επιπέδου, καθορίζουν τα συστατικά τους ακόμα και τη δομή τους, δίνοντας τους τη μορφή κανονικών[12] ή όχι γλωσσών. Τέλος τις

[12] Όταν μια γλώσσα επιτρέπει την απευθείας μετάφραση των κανονικών διαγραμμάτων, "κανονικές μονάδες ελέγχου", χωρίς τη χρήση προτάσεων του τύπου "GO TO", τότε η γλώσσα λέγεται κανονική βλ. κεφάλαιο 2). Η FORTRAN V για παράδειγμα είναι εν γένει γλώσσα μη κανονική. Είναι επιτακτική όμως η ανάγκη, για λόγους πάνω απ'όλα πρακτικούς, η κανονοποίηση, η γραφή καλών γραμμένων, δομημένων προγραμμάτων, ιδίως σε γλώσσες μη κανονικές, σε γλώσσες δηλαδή που δεν διαθέτουν φυσικό τρόπο περιγραφής κανονικών σχημάτων.

ταξινομούν από άποψη χρήσης τους. Θα πρέπει να σημειώσουμε ότι, ορισμένα χαρακτηριστικά, είναι δυνατόν να τις κατατάξουν μεταξύ των καλών και βιώσιμων γλωσσών ή όχι, πέραν οποιασδήποτε άλλης υποστήριξης που συντελεί στην καθιέρωση μιας γλώσσας.

Η αρχή λοιπόν που επιβάλλεται, και γύρω από την οποία κινούμαστε για να χαρακτηριστούν οι γλώσσες, ανωτέρου επιπέδου, είναι ότι πρέπει να σχεδιαστούν έτσι ώστε να πλησιάζουν τον άνθρωπο στον τρόπο διατύπωσης της λύσης προβλημάτων. Ακόμα, το πρόγραμμα διατυπωμένο αγνοώντας τη γλώσσα μηχανής, να είναι χρησιμοποιήσιμο με οποιαδήποτε μηχανή[14] . Επί πλέον, ο τρόπος διατύπωσης της λύσης ενός προβλήματος να μοιάζει με αυτόν, που το πρόβλημα διατυπώνεται συνήθως, στον κλάδο της επιστήμης που ανήκει.

Τα **συστατικά** μιας Γλώσσας Προγραμματισμού είναι τα κατασκευάσματα που παριστούν τις ενέργειες που περιγράφει η γλώσσα, ή τα υπολογιστικά αντικείμενα που χειρίζεται. Έτσι τα **δεδομένα και η περιγραφή τους**, οι **τελεστές**, οι **εντολές**, οι **δηλώσεις**, οι **οριοθέτες**, η **δομή προγραμμάτων**, η **εισαγωγή (δεδομένων) / εξαγωγή (αποτελεσμάτων)**, αποτελούν τα συστατικά μιας Γλώσσας Προγραμματισμού.

Τα δεδομένα και η περιγραφή τους: τα δεδομένα των προβλημάτων, τα οποία οι υπολογιστικές διαδικασίες αναλύουν ή συνθέτουν ποικίλλουν ανάλογα από τους κλάδους της επιστήμης που προέρχονται. Τα δεδομένα λοιπόν διακρίνονται από τον τύπο τους. Ο τρόπος με τον οποίο γίνεται ο καθορισμός του τύπου μεταβλητών, ποικίλλει από γλώσσα σε γλώσσα. Οι διάφορες γλώσσες έχουν μεγάλη ή μικρή ποικιλία τύπων δεδομένων, η ποικιλία αυτή εξαρτάται από τον τρόπο που κάθε γλώσσα σχεδιάζεται, και από τους προγραμματιστές που απευθύνεται.

Τελεστές: οι τελεστές είναι ένα από τα μέσα που χρησιμοποιούνται στις γλώσσες προγραμματισμού, για τον μετασχηματισμό ή τον συνδυασμό στοιχείων των δεδομένων. Οι

---

[14] Η ύπαρξη "προτύπων" (standard) γλωσσών, υπαγορευόταν από την επιτακτική ανάγκη να χρησιμοποιούνται γλώσσες, εύκολα σε ποικίλα υπολογιστικά συστήματα, έτσι που να είναι δυνατή η μεταφορά και χρησιμοποίηση των προγραμμάτων από υπολογιστή σε υπολογιστή.

Στις Η.Π.Α. το American National Institute (ANSI) αποτελεί τον εθνικό οργανισμό για εκπόνηση προτύπων σε εθελοντική βάση. Η μεγάλη ανάπτυξη προϊόντων στις Η.Π.Α. οδήγησε σε μια de facto αναγνώριση των προτύπων ANSI σε όλον τον κόσμο, π.χ. ANS FORTRAN, ANS Pascal, κ.λ.π.

Σε Ευρωπαϊκό επίπεδο έχουμε τον Ευρωπαϊκό Οργανισμό Προτυποποίησης CEN και τον Ευρωπαϊκό Οργανισμό Προτυποποίησης στην Ηλεκτροτεχνική CENELEC.

Σε διεθνές επίπεδο οι δραστηριότητες προτυποποίησης συντονίζονται από δύο διεθνείς οργανισμούς, τον Διεθνή Οργανισμό Προτυποποίησης ISO και τη Διεθνή Ηλεκτροτεχνική Επιτροπή IEC.

τελεστές ταξινομούνται σε υπολογιστικούς, σχεσιακούς, λογικούς, σύνδεσης αλυσίδων. Οι τελεστές είναι συνήθως δυικοί και ενθετικοί (infix), δηλαδή γράφονται μεταξύ των στοιχείων που συνδυάζουν. Μπορούν όμως να εμφανίζονται πρίν τις μεταβλητές, οπότε λέγονται προθηματικοί (prefix), ή μετά και τότε λέγονται επιθηματικοί (postfix).

Εντολές: με τον όρο εντολή (command) εννοούμε μια εκτελέσιμη ενέργεια που διατυπώνεται σε μια γλώσσα. Όμως ο όρος αυτός χρησιμοποιείται σήμερα ιδίως για να περιγράψει τις εντολές μηχανής (instructions), ή ακόμα τις εντολές ελέγχου εργασιών, υπολογισμών. Πρέπει να τονίσουμε όλως ιδιαιτέρως ότι, ένα πρόγραμμα αποτελείται από προτάσεις (statments), όπως εξαλλου ένα κείμενο σ'οποιαδήποτε γλώσσα, οι προτασεις αυτές δεν είναι απαραίτητο νάναι εντολές, γιατί απλά δεν είναι πάντα εκτελέσιμες. Μπορεί να αναθέτουν κάποια ή κάποιες τιμές σε μια ή περισσότερες μεταβλητές, ή και να τις περιγράφουν.

Δηλώσεις: με τις δηλώσεις παρέχουμε πληροφορίες στο μεταφραστικό πρόγραμμα του λειτουργικού συστήματος σχετικά με τον τύπο, τις ιδιότητες δηλαδή των μεταβλητών που αντιστοιχούν στα δεδομένα, που πρόκειται να επεξεργαστεί το πρόγραμμα. Υπάρχει και μια κατηγορία πληροφοριών, που παρέχονται από οδηγίες προς τον μεταφραστή, και περιγράφουν τρόπους εισαγωγής ή εξαγωγής δεδομένων, ή άλλες λεπτομέρειες που έχουν σχέση με το όλο περιβάλλον, μέσα στο οποίο λειτουργεί το υπολογιστικό σύστημα.

Οριοθέτες: οι οριοθέτες είναι σύμβολα απλά ή σύνθετα που εξυπηρετούν συντακτικά, γιατί βοηθούν στον καθορισμό διαφόρων μερών της γλώσσας. Για παράδειγμα, το κενό διάστημα είναι ένας συνηθισμένος οριοθέτης, άλλοτε πάλι οι τελεστές θεωρούνται οριοθέτες αφού συμβάλλουν στον καθορισμό της αρχής και του τέλους των προσδιοριστών. Πολλές γλώσσες όπως η PASCAL, PL/1 η C κ.α. χρησιμοποιούν ειδικούς οριοθέτες (π.χ. το ειδικό σύμβολο ;), για να υποδείξουν το τέλος μιας πρότασης.

Δομή Προγραμμάτων: δομή ενός προγράμματος είναι ο τρόπος με τον οποίο συνδυάζονται τα δεδομένα, οι τελεστές και τα άλλα στοιχεία μιας γλώσσας. Είναι ακόμα η μορφή των προτάσεων που εκτελούνται υπό όρους, των προτάσεων επαναληπτικών εκτελέσεων, των σύνθετων προτάσεων (η ύπαρξη δηλαδή ή όχι συγκροτημάτων από απλές προτάσεις που θεωρούνται σαν απλή πρόταση). Τέλος είναι ο τρόπος ορισμού και κλήσης

υποπρογραμμάτων και το αν η γλώσσα επιτρέπει ή όχι τη δημιουργία ιεραρχίας συγκροτημάτων (blocks).

Εισαγωγή / Εξαγωγή: η εισαγωγή δεδομένων / εξαγωγή αποτελεσμάτων δεν έχει άμεση σχέση με τη δομή της γλώσσας, αλλά με την ευχρηστία της. Είναι οι τρόποι εισαγωγής δεδομένων, κατά σειρμούς ή κατά τμήματα, οι τρόποι προσπέλασης σ' ένα αρχείο, διαδοχικά ή άμεσα, για την εισαγωγή ή ανάκληση πληροφοριών, και τέλος οι διάφοροι τρόποι εκτύπωσής τους.

1.4.3 Ταξινόμηση ως προς τη Χρήση τους

Εκατοντάδες διαφορετικές μεταξύ τους γλώσσες προγραμματισμού έχουν επινοηθεί και χρησιμοποιηθεί. Ενώ πολλές από αυτές εξελίσσονται, άλλες, ίσως ακόμα περισσότερες, σχεδιάζονται και εφαρμόζονται. Οι περισσότερες από τις γλώσσες αυτές έχουν επιννοηθεί για να εξυπηρετήσουν ειδικές ανάγκες, δηλαδή για να λύνουν προβλήματα ειδικών κλάδων της επιστήμης. Ταξινομήσεις λοιπόν των γλωσσών προγραμματισμού, μπορούν να γίνουν, όπως προαναφέραμε, από πολλές απόψεις. Ταξινόμηση π.χ. από την άποψη της δομής που επιβάλλουν στα προγράμματα οι γλώσσες αυτές, παρουσιάζει ενδιαφέρον. Όμως είναι πολύ χρήσιμη, για όσους θέλουν να χρησιμοποιήσουν μια γλώσσα για τις υπολογιστικές τους ανάγκες, η ταξινόμηση με βάση τις εφαρμογές για τις οποίες προορίζονται οι γλώσσες. Μια τέτοια ταξινόμηση θα επιχειρήσουμε στην παράγραφο αυτή. Έτσι οι μισές περίπου από τις υπάρχουσες γλώσσες ανήκουν σε έξι κατηγορίες. Οι υπόλοιπες κατανέμονται σε κατηγορίες πολύ ειδικευμένων εφαρμογών.

Γλώσσες Αριθμητικών - Επιστημονικών Υπολογισμών: είναι η παλαιότερη κατηγορία γλωσσών, αφού οι υπολογιστές πρωτοχρησιμοποιήθηκαν για την επίλυση αριθμητικών προβλημάτων, προβλημάτων που προέρχονταν από τις θετικές επιστήμες. Αναφέρουμε τις πιό γνωστές : FORTRAN, ALGOL 60, BASIC, APL/360, EULER. Η προσπάθεια, κατά φυσικότερο τρόπο, περιγραφής και προγραμματισμού παράλληλων αλγορίθμων έδωσε τη γλώσσα OL/2.

Γλώσσες Εμπορικών Εφαρμογών: Η γνωστότερη και πλέον διαδεδομένη είναι η COBOL, τα προγράμματα της έχουν τη μορφή προτάσεων γραμμένων στην Αγγλική

Γλώσσα. Η PL/DB είναι μια αξιόλογη προσπάθεια στην ανάπτυξη αυτής της κατηγορίας των γλωσσών. Η γλώσσα αυτή προορίζεται ειδικότερα για τον χειρισμό βάσεων δεδομένων, έχει δε κανονικού τύπου εκτελέσιμες προτάσεις για αριθμητική, για λογικές αποφάσεις και για τον έλεγχο ανακυκλώσεων.

Γλώσσες για την Επεξεργασία Πινάκων: Η πιό σημαντική και περισσότερο ικανή γλώσσα αυτής της κατηγορίας είναι η LISP 1.5 , έχει απλό αλλά και πολύ κουραστικό συμβολισμό. Όσες άλλες γλώσσες αναπτύχθηκαν προς αυτήν την κατεύθυνση, στην ουσία είναι LISP 1.5 , αλλά με ανωτέρου επιπέδου συμβολισμό. Η SLIP, βασίζεται στη FORTRAN, στην ουσία είναι επέκτασή της που επιτρέπει δυναμική χορήγηση μνήμης και δημιουργία συνδετικών πινάκων, και την εκτέλεση άλλων σχετικών πράξεων.

Γλώσσες για την Επεξεργασία Αλυσίδων: Η περισσότερο γνωστή και διαδεδομένη γλώσσα αυτής της κατηγορίας είναι η SNOBOL4. Αναπτύχθηκε από επιστήμονες των εργαστηρίων της Bell (Bell Telephone Laboratories), περιέχει χειρισμούς και πράξεις που δεν υπάρχουν στις συνηθισμένες γλώσσες προγραμματισμού. Οι χειρισμοί αυτοί είναι κατάλληλοι για την επεξεργασία αλυσίδων χαρακτήρων, γιαυτό χρησιμοποιείται σε ειδικούς τομείς δραστηριότητας και έρευνας όπως, ανάπτυξη μεθόδων μετάφρασης προγραμμάτων, προσομοίωση μηχανών, συμβολικές μαθηματικές επεξεργασίες, σύνταξη και μετάφραση κειμένων, γλωσσολογία, ανάλυση μουσικής. Μια ταχύτερη παραλλαγή της, με χρόνους εκτέλεσης μέχρι 10 φορές μικρότερους, και λιγότερη απαιτούμενη μνήμη, είναι η SPITBOL. Μια άλλη νεότερη προσπάθεια, προς την κατεύθυνση αυτή, είναι η ICON, στηρίζεται κάπως και διατηρεί πολλές από τις έννοιες της SNOBOL4, αλλά είναι πιό αναπτυγμένη και προσχωρημένη σαν γλώσσα ανωτέρου επιπέδου.

Γλώσσες για Συμβολικές και Αλγεβρικές Επεξεργασίες: αναπτύχθηκαν για τη διεξαγωγή αναλυτικών μαθηματικών υπολογισμών. Το αντικείμενο δηλαδή, της επεξεργασίας των γλωσσών αυτών, δεν είναι αριθμητικές εκφράσεις, όπως στις άλλες γλώσσες, αλλά συμβολικές, αλγεβρικές εκφράσεις (μαθηματικοί τύποι). Για αυτόν τον λόγο, οι γλώσσες αυτές έχουν κυρίως αναπτυχθεί από μαθηματικούς, φυσικούς και αστρονόμους. Οι εκφράσεις αυτές (μαθηματικοί τύποι) έχουν πολυπλοκότερες εσωτερικές παραστάσεις από εκείνες των αριθμητικών εκφράσεων και, συνήθως απαιτείται πολύ περισσότερος χώρος αποθήκευσης, επί πλέον κατά τη διάρκεια των αλγεβρικών επεξεργασιών

εμφανίζονται προβλήματα αλγεβρικών απλοποιήσεων. Σημειώνουμε ότι κάθε γλώσσα από αυτές ειδικεύεται σε ορισμένο είδος υπολογισμού, π.χ. στη διεξαγωγή απλών αλγεβρικών πράξεων, ή στον αποτελεσματικό χειρισμό πολυωνύμων και ρητών εκφράσεων, ή στον χειρισμό σειρών ή πινάκων ή τανυστών κλπ. Οι πιό γνωστές και διαδεδομένες γλώσσες είναι: REDUCE, FORMAC, ALTRAN, MASCYMA, SCRATCHPAD και CAMAL.

Γλώσσες Πολλαπλής Χρήσης: Στις αρχές της δεκαετίας του '70, άρχισαν προσπάθειες για την ανάπτυξη γλωσσών που να επιτρέπουν τη διατύπωση προγραμμάτων για όλους τους τομείς εφαρμογών. Έτσι αναπτύχθηκε η PL/1, με τη φιλοδοξία να αντικαταστήσει όλες τις άλλες γλώσσες. Πράγματι, η PL/1 έχει αυτή τη δυνατότητα. Άλλες γλώσσες αυτής της κατηγορίας είναι : η ALGOL 68, η PASCAL, η C και η CPS (για συνδιαλεκτικούς υπολογισμούς). Οι γλώσσες αυτές είναι νεότερες, ανήκουν στην κατηγορία των κανονικών γλωσσών, και έτσι επιτρέπουν και ενθαρρύνουν τον συστηματικό προγραμματισμό.

1.4.4 Αξιολόγηση των Γλωσσών Προγραμματισμού

Με την παράγραφο αυτή κλείνουμε την ενότητα των γλωσσών προγραμματισμού, αφού παρουσιάσαμε συμπυκνωμένα τα απαραίτητα, κατά τη γνώμη μας, στοιχεία για τη δουλειά μας, ενθαρρύνοντας έτσι και βοηθώντας, στην καλλίτερη αντιμετώπιση, αξιολόγηση και επιλογή των γλωσσών προγραμματισμού, κυρίως όμως στη γρήγορη και αποτελεσματική εξοικείωση με αυτές, φυσικά το θέμα δεν εξαντλείται εδώ.

Στην παράγραφο αυτή θα παρουσιάσουμε, και πάλι συνοπτικά, "**αξιόπιστα κριτήρια**" για την αξιολόγηση και σύγκριση γλωσσών προγραμματισμού. Κατά τη διάρκεια του σχεδιασμού της Ada, αναπτύχθηκε ένα, αρκετά εκτεταμένο σύνολο κριτηρίων αξιολόγησης γλωσσών · κριτήρια όμως λεπτομερή, εξειδικευμένα και κατά συνέπεια δυσκολοεφάρμοστα. Τα κριτήρια που θα παρουσιάσουμε εντοπίζουν ολιγάριθμα βασικά χαρακτηριστικά, τα οποία κάποιος ψάχνει ή εντοπίζει σε κάποια γλώσσα προγραμματισμού. Καθένα από τα χαρακτηριστικά πρέπει να είναι ευκολοκατανόητα και όλα μαζί να ορίζουν τι σημαίνει "**καλή γλώσσα**" με την αρκούντως ακριβή έννοια της λέξης, το κυριότερο, να καλύπτουν όλο **το φάσμα** των απόψεων **των προγραμματιστών**, δηλαδή των **σχεδιαστών** (designers), **αυτών που τις υλοποιούν** (implementers) και των **χρηστών**

(users).

Παρακάτω δίνεται πλήρης συνοπτικότατος πίνακας των χαρακτηριστικών, τα οποία και ορίζονται ως κριτήρια για την αξιολόγηση και σύγκριση των γλωσσών προγραμματισμού. Παρ' ότι τα κριτήρια αυτά δεν είναι λεπτομερή, διακρίνονται όμως για την ευρύτητά τους, γεγονός που επιτρέπει την ουσιαστική αξιολόγηση και σύγκριση, τουτέστιν αν μια γλώσσα χαρακτηριστεί ως "**άριστη**" σε όλα ή στα περισσότερα από τα κριτήρια που παραθέτουμε παρακάτω τότε ανεπιφύλακτα η γλώσσα είναι **εξαιρετική**.

Έτσι με τον όρο **εκφραστικότητα** (expressivity) εννοούμε την ικανότητα που έχει η γλώσσα να αντανακλά με σαφήνεια τον προσανατολισμό που της προσέδωσαν οι σχεδιαστές της. Μια εκφραστική γλώσσα : 1. επιτρέπει συμπαγείς εκφράσεις και ενθαρρύνει τη χρήση προτάσεων σχετιζόμενων με τον συστηματικό προγραμματισμό συνήθως της μορφής των "while βρόχων" και "if then else", 2. ενσωματώνει συμβολισμούς που είναι σύμφωνοι με αυτούς που χρησιμοποιούνται στους αντίστοιχους κλάδους για τους οποίους η γλώσσα έχει σχεδιαστεί. Ένας μαθηματικός για παράδειγμα χρησιμοποιώντας FORTRAN ή C θα πρέπει να περιμένει να συναντήσει τις αλγεβρικές εκφράσεις που συναντώνται στη παραδοσιακή Άλγεβρα, 3. χρησιμοποιεί ομοιόμορφα σύμβολα, δηλαδή δεν μεταβάλλεται η έννοιά τους χωρίς λόγο από το ένα πρόγραμμα στο άλλο.

Ο όρος **καλώς ορισμένη** (well-definedness) σημαίνει ότι η σύνταξη και η σημασιολογία της γλώσσας δεν είναι διφορούμενες, είναι εσωτερικά συμβιβαστές και πλήρεις. Ο ορισμός μιας καλώς ορισμένης γλώσσας παρέχει σε αυτούς που την υλοποιούν - την εφαρμόζουν, πλήρη περιγραφή των εκφραστικών τύπων της και τη σημασία τους, έτσι που να είναι δυνατή η γνώση της ακριβούς συμπεριφοράς κάθε έκφρασης πριν την εκτέλεση. Οι καλώς ορισμένες γλώσσες ενθαρρύνουν τη φορητότητα και την αποδεικτικότητα.

Με τον όρο **τύποι δεδομένων και δομών** (data types and structures) εννοούμε την ικανότητα της γλώσσας να υποστηρίζει ποικιλία πρωτόγονων και σύνθετων δεδομένων. Παράδειγμα πρωτόγονων δεδομένων είναι οι **ακέραιοι** (integers), οι **πραγματικοί** (real), οι **αλυσίδες χαρακτήρων** (strings), οι **ενδείκτες** (pointers) κλπ. Αντίστοιχα στα σύνθετα συμπεριλαμβάνονται οι **παρατάξεις** (arrays) και τα **καταχωρήματα** ή **εγγραφές** (records) κατ' αρχήν, αλλά και δεν αποκλείονται οι **δομές δεδομένων** (data structures)

όπως οι **συνδετικοί πίνακες** (linked lists), οι **στοιβάδες** (stacks), οι **ουρές** (queues) και τα **δένδρα** (trees).

Ο όρος **τμηματικοποίηση** ή επιμερισμός (modularity) έχει δύο όψεις: 1. την υποστήριξη της γλώσσας σε υποπρογράμματα, δηλαδή τη δυνατότητα ορισμού ανεξάρτητων διαδικασιών και συναρτήσεων υποπρογραμμάτων και την επικοινωνία με το κυρίως πρόγραμμα διά μέσου παραμέτρων ή καθολικών μεταβλητών, και 2. τη δυνατότητα επέκτασής της, με την έννοια της δυνατότητας που παρέχεται στον προγραμματιστή να ορίσει τελεστές και τύπους δεδομένων. Οι περισσότερες γλώσσες έχουν πολύ καλή υποστήριξη υποπρογραμμάτων αλλά ελάχιστες έχουν τη δυνατότητα επέκτασης όπως μόλις την περιγράψαμε.

Για να αξιολογίσουμε τα **μέσα εισόδου δεδομένων/εξόδου αποτελεσμάτων** (imput / output facilities) μιας γλώσσας εξετάζουμε την υποστήριξή της σε **ακολουθιακά με δείκτη** ή **τυχαίας προσπέλασης αρχεία** καθώς και αυτήν σε **βάσεις δεδομένων** και σε συναρτήσεις ανάκτησης πληροφοριών. Γενικά τα αρχεία αποθηκεύονται στην **περιφερειακή μνήμη (άμεσης προσπέλασης** ή **μαγνητικές ταινίες**), σε αντίθεση με τις **δομές δεδομένων** που αποθηκεύονται στην **κυρία μνήνη**. Ένα αρχείο είναι συνήθως πολύ μεγάλο για να καταχωρηθούν άμεσα στη μνήμη όλα τα καταχωρήματά του, γιαυτό και εφαρμόζονται διαφορετικές τεχνικές προγραμματισμού στην καταχώρηση αρχείων σε σχέση με αυτές που εφαρμόζονται στις δομές δεδομένων.

Η **φορητότητα** (portability) χαρακτηρίζει τις γλώσσες που μπορούν να υλοποιηθούν σε ποικίλους Η/Υ, δηλαδή έχουν σχεδιαστεί έτσι ώστε να είναι εν γένει ανεξάρτητες μηχανών. Οι περισσότερο φορητές γλώσσες μπορούμε να πούμε ότι είναι οι προτυποποιημένες και οι καλώς ορισμένες, πάντα υπάρχουν και οι εξαιρέσεις. Έτσι η ANS FORTRAN (1977) υποστηρίζεται σχεδόν σε όλους τους Η/Υ (mainframes), επειδή έχει προτυποποιηθεί. Επίσης, η C, αν και δεν έχει προτυποποιηθεί, εντούτοις υλοποιείται σε μια μεγάλη ποικιλία μηχανών διότι είναι καλώς ορισμένη. Η BASIC, είναι η λιγότερο φορητή γλώσσα σε σχέσει με τις προαναφερθείσες. Θεωρητικά υλοποιείται σχεδόν σε όλες τις μηχανές, εντούτοις κάθε υλοποίση πραγματοποιείται και με μια λίγο διαφορετική διάλεκτο. Για παράδειγμα οι συναρτήσεις γραφικών της, όλες οι διάλεκτοι δεν υποστηρίζουν τις ίδιες συναρτήσεις. Έτσι αν επιχειφηθεί να μεταφερθεί ένα πρόγραμμα γραμμένο σε BASIC από

μια μηχανή σε άλλη τότε ο προγραμματιστής θα πρέπει να γνωρίζει και τις δύο διαλέκτους για να ξαναγράψει κομμάτια του προγράμματος ανάλογα. Μια πραγματικά φορητή γλώσσα δεν παρουσιάζει τόσες διαλέκτους και κατά συνέπεια ένα πρόγραμμα μπορεί να τρέχει σε οποιαδήποτε μηχανή χωρίς καμμία αλλαγή.

Η **αποτελεσματικότητα** (efficiency) μιας γλώσσας έγγειται στη γρήγορη μεταγλώττιση και εκτέλεση της στις μηχανές που υλοποιείται. Παραδοσιακά η FORTRAN υπήρξε σχετικά αποτελεσματική γλώσσα στα πεδία εφαρμογών της. Οι μεταγλωττιστές είναι αποτελεσματικότεροι των διερμηνέων. Για παράδειγμα ένα πρόγραμμα σε LISP το οποίο μεταγλωττίζεται, εκτελείται πολύ γρηγορότερα από το ίδιο πρόγραμμα που διερμηνεύεται. Φυσικά χρησιμοποιούντες τους διερμηνείς έχουμε άλλα πλεονεκτήματα κατά τη διάρκεια ανάπτυξης του προγράμματος. Είναι ακόμα δεδομένο ότι η αποτελεσματικότητα της μηχανής είναι δευτερεύουσας σημασίας σε σχέση με τον χρόνο που απαιτείται από τον προγραμματιστή για να ολοκληρώσει το πρόγραμμά του.

Μερικές γλώσσες είναι περισσότερο **παιδαγωγικές** (pedagogy) από άλλες. Αυτό οφείλεται στο ότι οι γλώσσες αυτές έχουν μια εσωτερική ευκολία στη διδασκαλία και εκμάθησή τους, στην ύπαρξη καλλίτερων βιβλίων, στο ότι υλοποιούνται σε καλλίτερα περιβάλλοντα ανάπτυξης προγραμμάτων, στο ότι είναι ευρέως γνωστές και χρησιμοποιούνται από τους καλλίτερους προγραμματιστές στον τομέα εφαρμογής τους. Μερικές γλώσσες όπως η BASIC και η PASCAL σχεδιάστηκαν ειδικά σαν παιδαγωγικά εργαλεία. Σήμερα η BASIC κατέχει τα πρωτεία στα σχολεία της στοιχειώδους και μέσης εκπαίδευσης όπου διδάσκεται ο προγραμματισμός, ανάλογα συμβαίνουν για την PASCAL στην ανωτάτη εκπαίδευση, αυτό δεν σημαίνει ότι δεν διδάσκονται άλλες γλώσσες, που κρίνονται για την κάθε περίπτωση καταλληλότερες.

Τέλος η **γενικότητα** (generality) είναι το τελευταίο των 9 κριτηρίων για την αξιολόγηση και σύγκριση των γλωσσών και σημαίνει ότι η γλώσσα χρησιμοποιείται σε ένα ευρύτατο φάσμα εφαρμογών (**γλώσσα πολλαπλής χρήσης** - general purpose language).

Τα εννέα αυτά κριτήρια μας παρέχουν ομοιόμορφη κλίμακα αξιολόγησης των γλωσσών προγραμματισμού, παρόλο που δεν είναι όλα τόσο ποσοτικά και χειροπιαστά όσο αυτό της αποτελεσματικότητας, το οποίο μπορεί να μετρηθεί σε νανοδευτερόλεπτα. Ακόμα τα κριτήρια αυτά αποτελούν πολύ πιό ουσιαστικές και ασφαλείς αρχές επιλογής από το

"χρησιμοποιώ τη Χ γλώσσα γιατί πάντοτε αυτή χρησιμοποιούσα". Το τελευταίο σίγουρα δεν καταρρίπτει την κλασική και αξεπέραστη αρχή "με οποιαδήποτε γλώσσα μπορώ να κάνω οτιδήποτε", συμπληρώνουμε ότι αρκεί η γλώσσα να σου εξασφαλίζει έστω τις ελάχιστες δυνατότητες γιαυτό που θέλεις να κάνεις, και εσύ σαν προγραμματιστής να μην είσαι τσαρλατάνος, αλλά να έχεις σίγουρα γνώσεις, ικανότητα και ένα μηχάνημα με ανάλογες των στόχων σου προδιαγραφές.

ΚΡΙΤΗΡΙΑ ΑΞΙΟΛΟΓΗΣΗΣ & ΣΥΓΚΡΙΣΗΣ Γ.Π.	PASCAL	FORTRAN	C	PROLOG
1. ΕΚΦΡΑΣΤΙΚΟΤΗΤΑ	ΚΑΛΗ	ΕΠΑΡΚΗΣ	ΚΑΛΗ	ΚΑΛΗ
2. ΚΑΛΩΣ ΟΡΙΣΜΕΝΗ	ΑΡΙΣΤΗ	ΚΑΛΗ	ΑΡΙΣΤΗ	ΑΡΙΣΤΗ
3. ΤΥΠΟΙ ΔΕΔΟΜΕΝΩΝ ΚΑΙ ΔΟΜΩΝ	ΕΠΑΡΚΗΣ	ΕΠΑΡΚΗΣ	ΚΑΛΗ	ΚΑΛΗ
4. ΤΜΗΜΑΤΙΚΟΠΟΙΗΣΗ (ΕΠΙΜΕΡΙΣΜΟΣ)	ΚΑΛΗ	ΚΑΛΗ	ΑΡΙΣΤΗ	ΚΑΛΗ
5. ΜΕΣΑ ΕΙΣΑΓΩΓΗΣ / ΕΞΑΓΩΓΗΣ	ΚΑΛΗ	ΚΑΛΗ	ΚΑΛΗ	ΚΑΛΗ
6. ΦΟΡΗΤΟΤΗΤΑ	ΑΡΙΣΤΗ	ΚΑΛΗ	ΑΡΙΣΤΗ	ΚΑΚΗ
7. ΑΠΟΤΕΛΕΣΜΑΤΙΚΟΤΗΤΑ	ΚΑΛΗ	ΑΡΙΣΤΗ	ΑΡΙΣΤΗ	ΕΠΑΡΚΗΣ
8. ΠΑΙΔΑΓΩΓΙΚΗ	ΚΑΛΗ	ΚΑΛΗ	ΕΠΑΡΚΗΣ	ΚΑΚΗ
9. ΓΕΝΙΚΟΤΗΤΑ	ΕΠΑΡΚΗΣ	ΕΠΑΡΚΗΣ	ΚΑΛΗ	ΚΑΚΗ

ΠΙΝΑΚΑΣ 1.4.2: Αξιολόγηση και σύγκριση βάσει των εννέα κριτηρίων των γλωσσών, PASCAL, FORTRAN, C και PROLOG. Με τους όρους ΑΡΙΣΤΗ, ΚΑΛΗ, ΕΠΑΡΚΗΣ και ΚΑΚΗ αποδίδουμε τους αντίστοιχους αγγλικούς όρους EXCELLENT, GOOD, FAIR και POOR.

1.5 Ασκήσεις (Ερωτήσεις για Επανάληψη και Εμπέδωση)

1. Τι καλείται προγραμματισμός, τι πρόγραμμα, τι κώδικας και τι ψευδοκώδικας.
2. (α) Ποιές λέγονται τυπικές γλώσσες και πώς παράγονται, (β) ποιές κατηγορίες τυπικών γλωσσών εμπλέκονται στον προγραμματισμό, αναφέρετε αντίστοιχα παραδείγματα.
3. Διατυπώστε την έννοια του αλγορίθμου.
4. Ποιές βασικές ενέργειες - ενότητες περιλαμβάνονται στο σώμα μιας υπολογιστικής διαδικασίας.
5. Ποιά είναι τα τρία επίπεδα των γλωσσών προγραμματισμού, αναφέρατε παραδείγματα.
6. Τι καλούμε συστατικά μιας γλώσσας προγραμματισμού και να αναφερθούν.
7. Ταξινόμηση των γλωσσών προγραμματισμού ως προς τη χρήση τους.
8. Αναφέρατε τα κριτήρια αξιολόγησης των γλωσσών προγραμματισμού.

ΚΕΦΑΛΑΙΟ 2

ΠΕΡΙ ΣΥΣΤΗΜΑΤΙΚΟΥ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

2.1 Εισαγωγή

Είναι δεδομένο ότι, αν θέλουμε να λύσουμε ή να επεξεργαστούμε γενικά ένα πρόβλημα με τη βοήθεια Η/Υ, θα πρέπει να γράψουμε το ανάλογο πρόγραμμα. Σήμερα με την ανάπτυξη και τη διάδοση των Η/Υ, τα προβλήματα που επεξεργαζόμαστε είναι μεγάλα και σύνθετα, με αποτέλεσμα να απαιτείται η γραφή ανάλογων, μεγάλων και σύνθετων προγραμμάτων, συστημάτων, δηλαδή προγραμμάτων χιλιάδων γραμμών. Εκ των πραγμάτων υποχρεωνόμαστε πριν τη γραφή του προγράμματος να σχεδιάσουμε τη στρατηγική επίλυσης του προβλήματος, για να προκύψει καλλίτερα σχεδιασμένο πρόγραμμα, που θα μπορεί να ολοκληρωθεί, να ελεχθεί, να διορθωθεί και να τροποποιηθεί πιό εύκολα. Το μοντέλο αυτό επίλυσης του προβλήματος, που συνθέτεται βάσει κανόνων[18] και προδιαγραφών, είναι ο γνωστός μας αλγόριθμος και βεβαίως περιγράφεται με κάποια γλώσσα περιγραφής αλγορίθμων. Επίσης είναι γνωστό ότι ο κωδικοποιημένος (μεταφρασμένος) αλγόριθμος σε κάποια γλώσσα προγραμματισμού λέγεται πρόγραμμα.

Για να αποκτήσει κάποιος την ικανότητα και τη δυνατότητα να προγραμματίζει σωστά και χωρίς περιορισμούς ως προς το σύνθετο και το πολύπλοκο των προβλημάτων ή ως προς τις ιδιαιτερότητες κάποιας γλώσσας προγραμματισμού, θα πρέπει από την αρχή να διδαχθεί να προγραμματίζει πειθαρχώντας στη σχεδίαση του αλγορίθμου και στις αρχές του συστηματικού προγραμματισμού ακόμα και στα μικρά και απλά προβλήματα, αυτό εξάλλου αποτελεί την καλλίτερη εμπέδωση και τον ασφαλέστερο τρόπο για να εξελιχθεί σε καλό προγραμματιστή.

Έτσι, θεωρήσαμε όχι μόνο από διδακτικής του προγραμματισμού άποψης αλλά και πρακτικής, ότι είναι πολύ βασικό να προηγηθεί της γραφής προγραμμάτων, ο τρόπος σύνθεσης και έκφρασης του αντίστοιχου αλγορίθμου, πράγμα που προυποθέτει τη "γνώση" και την "εξοικείωση" με τα βασικά εργαλεία έκφρασης και επικοινωνίας όλων των φάσεων

[18] Οι κανόνες αυτοί είναι γνωστοί ως αρχές και κανόνες του συστηματικού προγραμματισμού.

του προγραμματισμού, τα εργαλεία αυτά είναι οι γλώσσες περιγραφής αλγορίθμων, οι μεταγλώσσες και οι γλώσσες προγραμματισμού (αντικείμενο του προηγουμένου κεφαλαίου).

Στο κεφάλαιο αυτό, παρουσιάζουμε τις αρχές και τους κανόνες του συστηματικού προγραμματισμού καθώς και τη μεθοδολογία ανάπτυξης ενός προγράμματος, που ανάγεται, (α) στον εντοπισμό, στην κατανόηση και στην εν γένει διευκρίνιση των ζητουμένων του προβλήματος, (β) στη σχεδίαση της στρατηγικής επίλυσής του, (γ) στον καθορισμό των κύριων δομών δεδομένων του, αν υπάρχουν (η αναφορά γίνεται για την πληρότητα της παρουσίασης και μόνο) και τέλος (δ) στην κωδικοποίησή του, στη μετάφραση δηλαδή του αλγορίθμου στη γλώσσα προγραμματισμού που θα επιλέξουμε.

2.2 Βασικές Έννοιες και Αρχές του Συστηματικού Προγραμματισμού

Η ανάπτυξη των υπολογιστών δρομολογείται από τα μέσα της δεκαετίας του '50 και ακολουθεί όλο και πιό επιταχυνόμενους έως εκκρηκτικούς ρυθμούς, ιδίως από το τέλος της δεκαετίας του '60 και εντεύθεν. Οι υπολογιστές πλέον, μπορούν να ανταποκρίνονται με επιτυχία στις ανάγκες όλο και περισσότερων τομέων της ανθρώπινης δραστηριότητας. Αυτή η επιτελούμενη διείσδηση στη ζωή μας, δημιούργησε όπως ήταν φυσικό νέες ανάγκες, νέες προοπτικές αλλά και νέα προβλήματα.

Από τις κοσμογονικές αυτές εξελίξεις του κλάδου που επιτελούνται στην πορεία, όχι μόνο δεν ξέφυγε ο προγραμματισμός αλλά βρέθηκε στο επίκεντρο του ενδιαφέροντος και των ανακατατάξεων, διότι αντανακλά τις αυξανόμενες απαιτήσεις μας από τους Η/Υ και τις συνέπειες για την ικανοποίησή τους, δηλαδή τον ανταγωνισμό ανάμεσα στην εξέλιξη του **λογισμικού** (software) και σε αυτή του **μηχανικού** (hardware).

Μέσα λοιπόν στη δίνη των εξελίξεων και προς το τέλος της δεκαετίας του '60, αρχίζει η υποχώρηση της μέχρι τότε νοοτροπίας, που την εξέφραζε η **Τέχνη του Προγραμματισμού** παραχωρώντας βαθμιαία τη θέση στη νέα αναβαθμισμένη και πειθαρχημένη σε προδιαγραφές νοοτροπία της **Επιστήμης του Προγραμματισμού**, η οποία και ολοκληρώνεται προς το τέλος της δεκατίας του '70, με την καθιέρωση εντελώς νέων νοοτροπιών στα περί του προγραμματισμού. Αυτές συνοψίζονται στις αρχές και

κανόνες του **Συστηματικού Προγραμματισμού** (Systematic or Structured Programming), συμπαρασύροντας και άλλους κλάδους της διαμορφούμενης Επιστήμης των Ηλεκτρονικών Υπολογιστών πχ. τον σχεδιασμό γλωσσών κλπ.

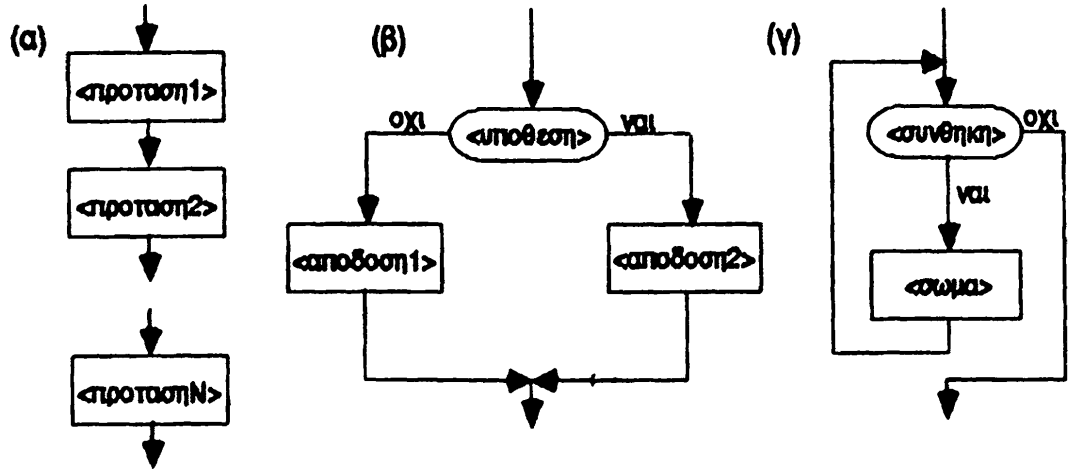
Έτσι τα μεγάλα σε έκταση προγράμματα, "συστήματα", για μακροχρόνια χρήση, θα μπορούν πλέον να συνθέτονται με τη συμμετοχή πολλών επιστημόνων και τεχνικών, με οργάνωση, μεθοδικότητα ασφάλεια, και να συντηρούνται, δηλαδή να αναθεωρούνται και να επεκτείνονται, εύκολα.

Θα μπορούσαμε να συνεχίσουμε για την ιστορία του προγραμματισμού γράφοντας και άλλα ενδιαφέροντα, φοβούμαι όμως μην απομακρυνθούμε από τον σκοπό μας, το "συστηματικά προγραμματίζειν". Έτσι αρκούμαστε στη θεμελιώδη και σημαντικότατη για τον προγραμματισμό επισήμανση του Bohm και του Jacobini που απέδειξαν [16] ότι, **ένα λογικό διάγραμμα, που παριστά έναν σωστό αλγόριθμο δηλαδή μια υπολογιστική διαδικασία, οσοδήποτε πολύπλοκο και αν είναι, μπορεί να περιγραφεί πλήρως σαν καθετισμός τριών μόνο κατασκευασμάτων, των λεγόμενων κανονικών βαθμίδων ή μονάδων ελέγχου ροής λογικών διαδικασιών**.

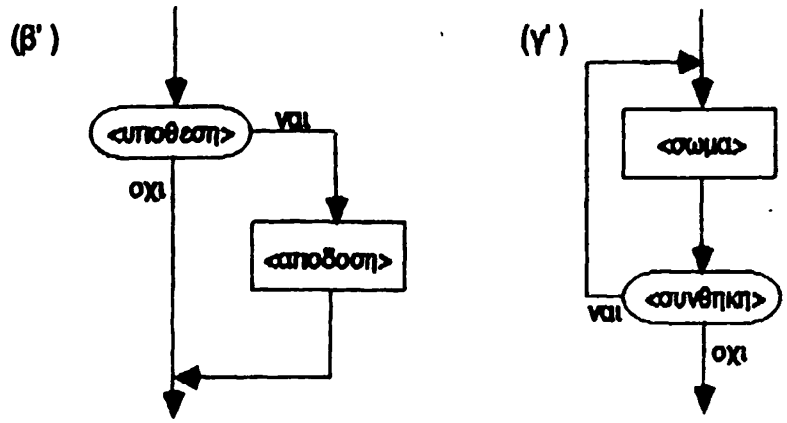
Τις κανονικές βαθμίδες ή μονάδες ελέγχου ροής λογικών διαδικασιών παριστούμε σε διαγραμματική μορφή στα Σχήματα 2.1.1 α, β, γ και 2.1.2 β', γ', το Σχήμα 2.1.3 παριστά μια ακόμα παραλλαγή της βαθμίδας (γ). Το θεμελιώδες χαρακτηριστικό των βαθμίδων είναι ότι έχουν, ή εφαρμοζόμενες πρέπει να έχουν, μια μόνο είσοδο και μια μόνο έξοδο.

Ένα **διάγραμμα ροής** που αποτελείται από κανονικές βαθμίδες λέγεται **κανονικό**, ένα **πρόγραμμα** που αντιστοιχεί σε κανονικό διάγραμμα λέγεται **κανονικό ή δομημένο** (structured). Μια **γλώσσα** λέγεται **κανονική** όταν διαθέτει απλές ή σύνθετες προτάσεις, οι οποίες λέγονται και προγραμματικά σχήματα, που επιτρέπουν την άμεση γραφή κανονικών προγραμμάτων, άλλως εάν δηλαδή η γραφή κανονικών προγραμματικών σχημάτων επιτυγχάνεται κατά έμμεσο τρόπο, η γλώσσα λέγεται μη κανονική. Αυτά, και όχι μόνο, συνθέτουν τον Συστηματικό Προγραμματισμό. Στο θέμα αυτό θα επανέλθουμε παρακάτω.

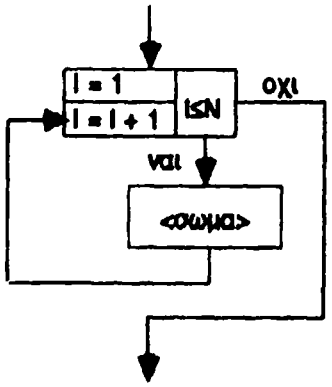
[16] Böhm, C., and Jacobini, G. Flow diagrams, Turing machines and languages with only two formation rules. Communications of the ACM, Vol 9, No 5 (May 1966) pp. 336 - 371.



ΣΧΗΜΑ 2.1.1 : Κανονικές Βαθμίδες Ροής Λογικών Διαδικασιών.
(α) Απλής αλληλουχίας προτάσεων, (β) Εκλογής και (γ) Ανακύκλωσης



ΣΧΗΜΑ 2.1.2 : Παραλλαγές των Κανονικών Βαθμίδων Ροής Λογικών Διαδικασιών. (β') Εκλογής και (γ') Ανακύκλωσης



ΣΧΗΜΑ 2.1.3 : Παραλλαγή Ανακύκλωσης, (ανακυκλώσεις γνωστού πλήθους)

ΠΑΡΑΔΕΙΓΜΑ 2.2.1 Να περιγραφούν οι κανονικές βαθμίδες ροής προγράμματος σε **ΕΑΓ** και **C++**.

Απόδειξη: Στην πραγματικότητα οι κανονικές βαθμίδες είναι μόνο δύο, της εκλογής β, β' και της ανακύκλωσης γ, γ'.

Α. Η **βαθμίδα 2.1.1(α)** παριστά ένα **ακολουθιακό σχήμα προτάσεων (αλληλουχία προτάσεων)**, που κάθε πρόταση της είναι δυνατόν να είναι, μια απλή πρόταση ή μια κανονική βαθμίδα ή κιβωτισμός κανονικών βαθμίδων (βλπ § 1, 2, 3 των παραδειγμάτων από Π2.2.2 έως Π2.2.8).

Β. Η **βαθμίδα 2.1.1(β)** και η **παραλλαγή της 2.1.2(β')** που θα περιγράψουμε παριστούν **επιλογή**, οι <αποδοση 1>, <αποδοση 2> και <αποδοση> μπορεί να είναι απλές ή σύνθετες προτάσεις ή κανονικές βαθμίδες ή κιβωτισμοί κανονικών βαθμίδων.

(Ι) **ΕΑΓ**,

1. στη βαθμίδα 1(β) αντιστοιχεί το κανονικό προγραμματικό σχήμα της ΕΑΓ:

```
εαν <υποθεση>
  τοτε   <αποδοση 1>
  αλλως <αποδοση 2> ;        (βλπ §1 του Π2.2.3)  και
```

2. στη βαθμίδα 2(β') αντιστοιχεί το κανονικό προγραμματικό σχήμα της ΕΑΓ:

```
εαν <υποθεση>
  τοτε <αποδοση> ;           (βλπ §1 των Π2.2.7 και Π2.2.8).
```

(ΙΙ) **C++**,

1. για τη **βαθμίδα 2.1.1(β)** έχουμε το κανονικό προγραμματικό σχήμα :

```
if (<υποθεση>)
  { <αποδοση 1> ; }
else
  { <αποδοση 2> ; }          (βλπ §3 του Π2.2.3)  και
```

2. για τη 2(β') το αντίστοιχο :

```
if (<υποθεση>)
  { <αποδοση> ; }            (βλπ §1 των Π2.2.7 και Π2.2.8).
```

Γ. Τέλος περιγράφεται η **βαθμίδα ανακύκλωσης 2.1.1(γ)** και οι **παραλλαγές της 2.1.2(γ')** και **2.1.3**, οι 2.1.1, 2.1.2 παριστούν αγνώστου πλήθους υπό συνθήκη ανακυκλώσεις, ενώ η 2.1.3 γνωστού πλήθους ανακυκλώσεις. Το <σωμα> μπορεί να είναι απλή ή σύνθετη προτάση ή κανονική βαθμίδα ή κιβωτισμός κανονικών βαθμίδων.

(i) **ΕΑΓ**,

1. στη **βαθμίδα 2.1.1(γ)** αντιστοιχεί το κανονικό προγραμματικό σχήμα της ΕΑΓ:

εφοσον <συνθήκη> **επαναλαβε**
<σωμα> ; (βλπ §1 των Π2.2.4, Π2.2.5 και Π2.2.7),

2. στη **βαθμίδα 2.1.2(γ')** αντιστοιχεί το κανονικό προγραμματικό σχήμα της ΕΑΓ:

επαναλαβε
<σωμα>
εως <συνθήκη> ; και

3. για την παραλλαγή 2.1.3, γνωστού πλήθους ανακυκλώσεων, προβλέπεται το κανονικό προγραμματικό σχήμα :

για J ← 1 **εως** N **επαναλαβε**
<σωμα> ; (βλπ §1 των Π2.2.6 και Π2.2.8),

(ii) **C++**,

1. για τη **βαθμίδα 2.1.1(γ)** έχουμε το κανονικό προγραμματικό σχήμα κατά άμεσο τρόπο :

while (<συνθηκη>)
{ <σωμα> ;} (βλπ §3 των Π2.2.4, Π2.2.5 και Π2.2.7),

2. για τη **2.1.2(β')** το αντίστοιχο κανονικό προγραμματικό σχήμα:

do
{ <σωμα> ; }
while (<συνθηκη>) ; και

3. για τη **2.1.3** έχουμε το κανονικό προγραμματικό σχήμα:

for (<αρχικη_εκφραση> ;<εκφραση_ελεγχου> ;<εκφραση_μετρητης>)
{ <σωμα> ; } (βλπ §3 των Π2.2.6 και Π2.2.8).

Παρατηρήσεις :

1. Το **κανονικό** ή **δομημένο πρόγραμμα** είναι το αποτέλεσμα της εφαρμογής των αρχών

και των κανόνων του συστηματικού προγραμματισμού στη γραφή του προγράμματος,

2. Ένα **δομημένο πρόγραμμα** συνίσταται από :

α. ακολουθία απλών προτάσεων και κανονικών βαθμίδων (κιβωτισμένων ή μη, με ειδιάκριτα τα επίπεδα κιβωτισμού) επιλογής και ανακύκλωσης, **για να είναι εύκολο στον έλεγχο και εντοπισμό τυχών σφαλμάτων και ευκολοσυντηρούμενο,**

β. προσεκτικά στοιχειοθετημένα σχόλια όπου δεί, **για να είναι ευανάγνωστο,**

γ. μεταβλητές (προσδιοριστές) που με συνέπεια και φαντασία παριστούν ή αποδίδουν την έννοια στην οποία αντιστοιχούν, **για να είναι ευκολονόητο,**

3. Σας προτρέπουμε να προγραμματίζετε με μεθοδικότητα, υπομονή και πειθαρχία στις αρχές του συστηματικού προγραμματισμού, έτσι αποφεύγετε τον εκνευρισμό και τις απογοητεύσεις. Επιβάλλονται καλογραμμένα προγράμματα, η δοξασία ότι **"αρκεί να τρέχει"** είναι πέρα για πέρα λάθος, δεν εκφράζει διαφορετική επιστημονική αντίληψη, αλλά κάποιο προσωπικό πρόβλημα, κάποια προσωπική ιδιοσυγκρασία, ή ίσως και κάποιο ψυχολογικό πρόβλημα. Σημειωτέον ότι η "Ψυχολογία του προγραμματισμού" έχει αποτελέσει αντικείμενο μελέτης. Προσοχή λοιπόν !!!

4. Στο εξής η οποιαδήποτε και οπουδήποτε αναφορά μας σε πρόγραμμα θα σημαίνει δομημένο πρόγραμμα, κάθε απόκλιση απορρίπτεται.

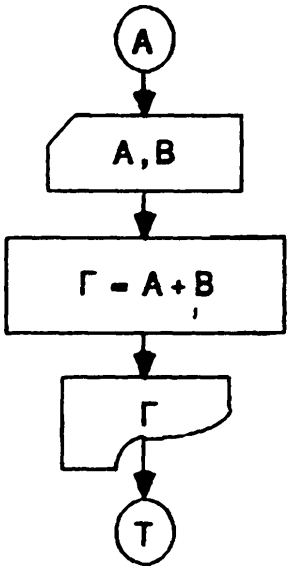
ΠΑΡΑΔΕΙΓΜΑ 2.2.2

Να περιγραφεί η διαδικασία **αθροίσματος δύο ακεραίων αριθμών** (διαβάζει δύο ακεραίους, υπολογίζει το άθροισμά τους και το τυπώνει) χρησιμοποιώντας την **ΕΑΓ**, το **λογικό διάγραμμα** ή **διάγραμμα ροής** και τη C++:

1. σχεδίαση του αλγορίθμου σε **ΕΑΓ**,

```
διαδικασια ΑΘΡΟΙΣΜΑ1 ;
δηλωση (Α, Β, Γ) ακερ ;
αρχη
  διαβασε Α, Β ;
  Γ ← Α + Β ;
  τυπωσε Γ ;
τελος ΑΘΡΟΙΣΜΑ1
```

2. σχεδίαση του αλγορίθμου με **Λογικό Διάγραμμα ή Διάγραμμα Ροής,**



Σημείωση: Η υπολογιστική διαδικασία που περιγράφεται με το παραπάνω διάγραμμα ροής αποτελείται από μια ακολουθία απλών προτάσεων (πρβλ Σχήμα 2.1.1(α)). Αναλυτικά αποτελείται από μια πρόταση εισόδου (**διάβασε**), μια πρόταση ανάθεσης (Γ ← A + B) και μια πρόταση εξόδου (**τύπωσε** ή **γράψε**).

3. μετάφραση ή κωδικοποίηση του αλγορίθμου σε **C++**,

```
// όνομα αρχείου: ATHROIS1.CPP [17]
 # include <iostream.h>
 main( )
 {
    // δηλώσεις
  int A,B,C ;
    // είσοδοςδεδομένων
  cin>>A>>B ;
    // υπολογισμός αθροίσματος
  C = A + B ;
    // εξαγωγή αποτελέσματος
  cout<<' το άθροισμα είναι : '<<C ;
  return(0);
 }
```

[17] // με το αυτό το σύνθετο σύμβολο στην αρχή της γραμμής γράφονται τα σχόλια στη C, προτάσεις δηλαδή που δεν αναγνωρίζονται από τον Η/Υ, αλλά είναι για διευκόλυνση του προγραμματιστή.

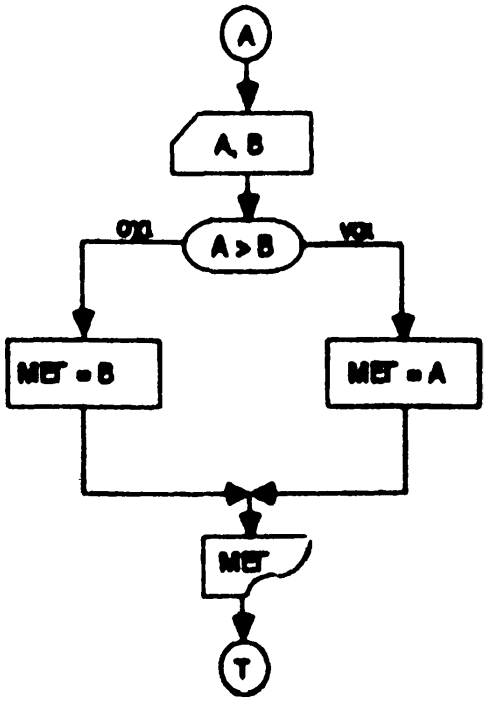
ΠΑΡΑΔΕΙΓΜΑ 2.2.3

Να περιγραφεί η διαδικασία **εύρεσης του μεγίστου δύο δοθέντων ακεραίων αριθμών** (διαβάζει δύο ακεραίους, βρίσκει τον μέγιστο και τον τυπώνει) χρησιμοποιώντας την **ΕΑΓ**, το **λογικό διάγραμμα** ή **διάγραμμα ροής** και τη C++:

1. σχεδίαση του αλγορίθμου σε **ΕΑΓ**,

```
διαδικασια ΕΥΡΜΕΓ1;
 /* δηλώσεις */ [18]
δηλωση (Α, Β, ΜΕΓ) ακερ;
αρχη
    /* εισαγωγή δεδομένων */
 διαβασε Α, Β;
   /* εύρεση μεγίστου */
 εαν Α > Β
     τοτε ΜΕΓ ← Α
     αλλως  ΜΕΓ ← Β;
   /* εξαγωγή αποτελέσματος */
 γραφε ΜΕΓ;
τελος ΕΥΡΕΣΜΕΓ1
```

2. σχεδίαση του αλγορίθμου με **Λογικό Διάγραμμα** ή **Διάγραμμα Ροής**.



[18] /* ... */ μέσα στα σύμβολα αυτά γράφονται τα σχόλια στην ΕΑΓ.

Σημείωση: Η υπολογιστική διαδικασία που περιγράφεται με το παραπάνω διάγραμμα ροής αποτελείται από μια ακολουθία απλών προτάσεων και βαθμίδων (πρβλ Σχήμα 2.1.1(α)). Αναλυτικά αποτελείται από μια πρόταση εισόδου (**διάβασε**), μια κανονική βαθμίδα εκλογής (πρβλ Σχήμα 2.1.1(β)) (**υπολόγισε**) και μια πρόταση εξόδου (**τύπωσε** ή **γράψε**).

3. μετάφραση ή κωδικοποίηση του αλγορίθμου σε **C++** :

```
// όνομα αρχείου: EYRMEG1.CPP
 # include <iostream.h>
 main( )
 {
      // δηλώσεις
  int A,B,MEG ;
      // εισαγωγή δεδομένων
  cin >>A>>B ;
      // εύρεση μεγίστου
  if (A > B)
    { MEG = A ; }
  else
    { MEG = B ; }
      // εξαγωγή αποτελέσματος
  cout<<'ο μέγιστος είναι : '<<MEG ;
  return(0);
 }
```

ΠΑΡΑΔΕΙΓΜΑ 2.2.4

Να περιγραφεί η διαδικασία **καταμέτρησης, αγνώστου πλήθους, ακεραίων αριθμών** χρησιμοποιώντας την **ΕΑΓ**, το **λογικό διάγραμμα** ή **διάγραμμα ροής** και τη C++:

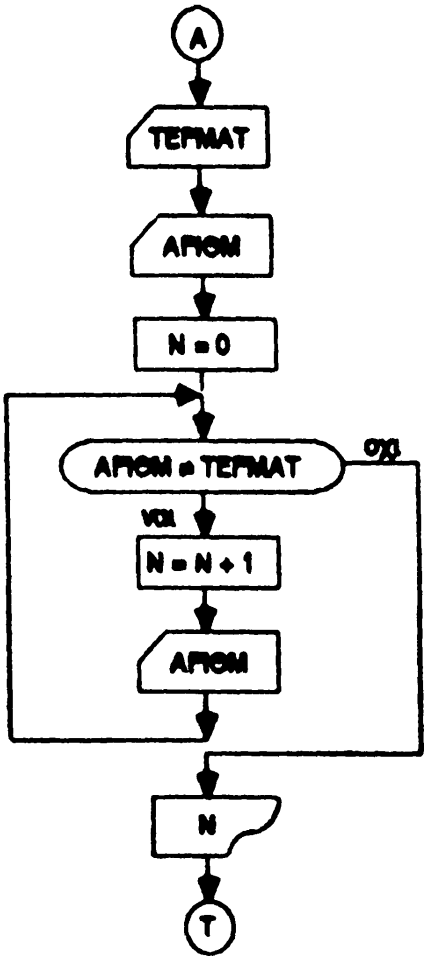
1. σχεδίαση του αλγορίθμου σε **ΕΑΓ** ,

```
διαδικασια ΚΑΤΑΜΕΤΡΗΣΗ ;
/* δηλώσεις */
δηλωση (ΤΕΡΜΑΤ, ΑΡΙΘΜ, Ν) ακερ ;
αρχη
    /* εισαγωγή δεδομένων */
 διαβασε ΤΕΡΜΑΤ;
 διαβασε ΑΡΙΘΜ ;
  /* ανάθεση αρχικής τιμής στον καταμετρητή[19] */
```

[19] Ο καταμετρητής ορίζεται με αναδρομική σχέση (αναδρομικός ορισμός) N=0, N=N+1.

```
N ← 0;
  /* υπολογισμός πλήθους */
εφοσον ΑΡΙΘΜ ≠ ΤΕΡΜΑΤ επαναλαβε
      (N ← N + 1 ;
  /* εισαγωγή νέων δεδομένων */
       διαβασε ΑΡΙΘΜ ; )
  /* εξαγωγή αποτελέσματος */
τυπωσε ' το πλήθος των αριθμών είναι : ', N ;
τελος ΚΑΤΑΜΕΤΡΗΣΗ
```

2. σχεδίαση του αλγορίθμου με **Λογικό Διάγραμμα ή Διάγραμμα Ροής.**



Σημείωση: Η υπολογιστική διαδικασία που περιγράφεται με το παραπάνω διάγραμμα ροής αποτελείται από μια ακολουθία απλών προτάσεων και βαθμίδων (πρβλ Σχήμα 2.1.1(α)). Αναλυτικά αποτελείται από μια πρόταση εισόδου (διάβασε), μια κανονική βαθμίδα ανακύκλωσης (πρβλ Σχήμα 2.1.1(γ)) (υπολόγισε) και μια πρόταση εξόδου (τύπωσε ή

γράψε).

3. μετάφραση ή κωδικοποίηση του αλγορίθμου σε **C++** :

```
// όνομα αρχείου: KATAMETR.CPP
# include <iostream.h>
main( )
{
     // δηλώσεις
int TERMAT, ARITHM, N ;
     // εισαγωγή δεδομένων
cin >>TERMAT ;
cin >>ARITHM ;
     // ανάθεση αρχικής τιμής στον καταμετρητή
N = 0 ;
     // υπολογισμός πλήθους
while ( ARITHM != TERMAT )
    {
     N = N + 1;
     // εισαγωγή νέων δεδομένων
      cin >>ARITHM ;
    }
     // εξαγωγή αποτελέσματος
cout<<' το πλήθος των αριθμών είναι : '<<N ;
return(0);
}
```

ΠΑΡΑΔΕΙΓΜΑ 2.2.5

Να περιγραφεί η διαδικασία **αθροίσματος, αγνώστου πλήθους, ακεραίων αριθμών** χρησιμοποιώντας την **ΕΑΓ**, το **λογικό διάγραμμα** ή **διάγραμμα ροής** και τη C++:

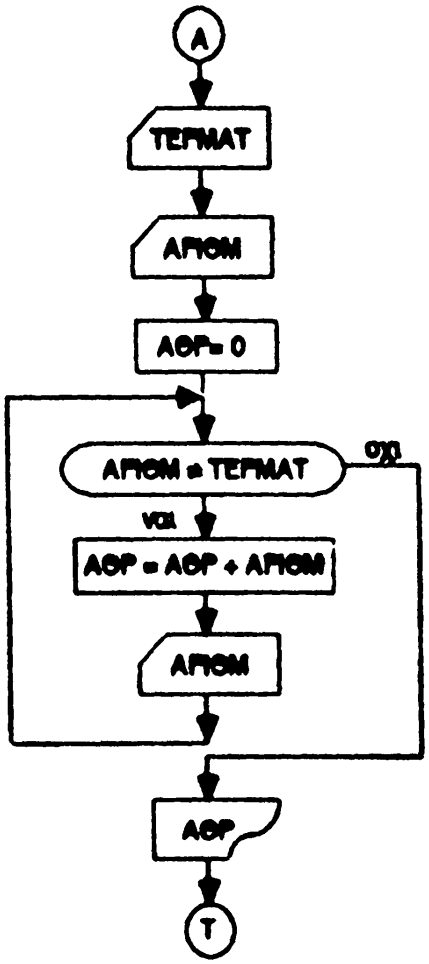
1. σχεδίαση του αλγορίθμου σε **ΕΑΓ** ,

```
διαδικασια ΑΘΡΟΙΣΜΑ2 ;
 /* δηλώσεις */
δηλωση (ΤΕΡΜΑΤ, ΑΡΙΘΜ, ΑΘΡ) ακερ ;
αρχη
    /* εισαγωγή δεδομένων */
  διαβασε ΤΕΡΜΑΤ;
  διαβασε ΑΡΙΘΜ ;
   /* ανάθεση αρχικής τιμής στον αθροιστή[20] */
```

[20] Ο αθροιστής ορίζεται με αναδρομική σχέση (αναδρομικός ορισμός) ΑΘΡ=0, ΑΘΡ=ΑΘΡ+ΑΡΙΘΜ.

```
ΑΘΡ ← 0;
  /* υπολογισμός αθροίσματος */
εφοσον ΑΡΙΘΜ ≠ ΤΕΡΜΑΤ επαναλαβε
      (ΑΘΡ ← ΑΘΡ + ΑΡΙΘΜ ;
  /* εισαγωγή νέων δεδομένων */
       διαβασε ΑΡΙΘΜ ; )
  /* εξαγωγή αποτελέσματος */
τυπωσε ' το άθροισμα είναι : ', ΑΘΡ ;
τελος ΑΘΡΟΙΣΜΑ2
```

2. σχεδίαση του αλγορίθμου με **Λογικό Διάγραμμα ή Διάγραμμα Ροής.**



Σημείωση: Η υπολογιστική διαδικασία που περιγράφεται με το παραπάνω διάγραμμα ροής αποτελείται από μια ακολουθία απλών προτάσεων και βαθμίδων (πρβλ παράδειγμα 2.2.4).

3. μετάφραση ή κωδικοποίηση του αλγορίθμου σε **C++** :

```
// όνομα αρχείου: ATHROIS2.CPP
 # include <iostream.h>
 main( )
 {
      // δηλώσεις
  int TERMAT, ARITHM, ATHR ;
      // εισαγωγή δεδομένων
  cin >>TERMAT ;
  cin >>ARITHM ;
      // ανάθεση αρχικής τιμής στον αθροιστή
  ATHR = 0 ;
      // υπολογισμός αθροίσματος
  while ( ARITHM != TERMAT )
      {
       ATHR = ATHR + ARITHM;
      // εισαγωγή νέων δεδομένων
       cin >>ARITHM ;
      }
      // εξαγωγή αποτελέσματος
  cout<<' το άθροισμα είναι : '<<ATHR ;
  return(0);
 }
```

ΠΑΡΑΔΕΙΓΜΑ 2.2.6

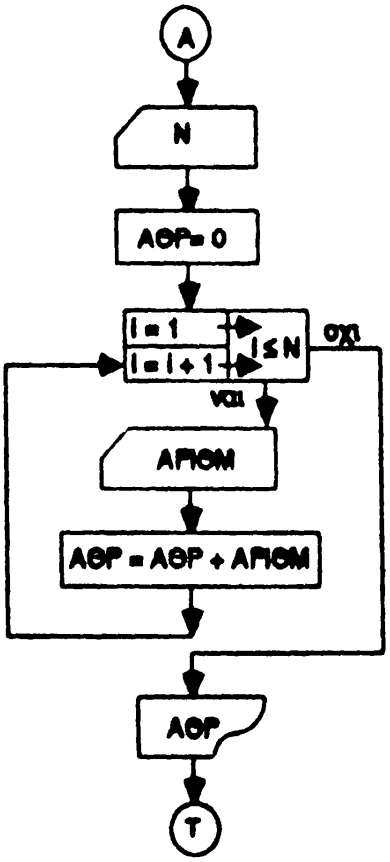
Να περιγραφεί η διαδικασία **αθροίσματος, γνωστού πλήθους, ακεραίων αριθμών** χρησιμοποιώντας την **ΕΑΓ**, το **λογικό διάγραμμα** ή **διάγραμμα ροής** και τη C++:

1. σχεδίαση του αλγορίθμου σε **ΕΑΓ**,

```
διαδικασια ΑΘΡΟΙΣΜΑ3 ;
 /* δηλώσεις */
δηλωση (N, I, ΑΡΙΘΜ, ΑΘΡ) ακερ ;
αρχη
    /* εισαγωγή δεδομένων */
  διαβασε N;
    /* ανάθεση αρχικής τιμής στον αθροιστή */
  ΑΘΡ ← 0;
    /* υπολογισμός αθροίσματος */
  για I ← 1 εως N επαναλαβε
       ( διαβασε ΑΡΙΘΜ ; /* εισαγωγή δεδομένων */
```

```
        ΑΘΡ ← ΑΘΡ + ΑΡΙΘΜ ; )
    /* εξαγωγή αποτελέσματος */
    τυπωσε 'το άθροισμα είναι : ', ΑΘΡ ;
τελος ΑΘΡΟΙΣΜΑ3
```

2. σχεδίαση του αλγορίθμου με **Λογικό Διάγραμμα** ή **Διάγραμμα Ροής.**



Σημείωση: Η υπολογιστική διαδικασία που περιγράφεται με το παραπάνω διάγραμμα ροής αποτελείται από μια ακολουθία απλών προτάσεων και βαθμίδων (πρβλ Σχήμα 2.1.1(α)). Αναλυτικά αποτελείται από μια πρόταση εισόδου (**διάβασε**), μια κανονική βαθμίδα ανακύκλωσης (ανακυκλώσεις γνωστού πλήθους) (πρβλ Σχήμα 2.1.3) (**υπολόγισε**) και μια πρόταση εξόδου (**τύπωσε** ή **γράψε**).

3. μετάφραση ή κωδικοποίηση του αλγορίθμου σε **C++** :

```
// όνομα αρχείου: ATHROIS3.CPP
# include <iostream.h>
main( )
{
```

```
    // δηλώσεις
int N, I, ARITHM, ATHR ;
    // εισαγωγή δεδομένων
cin >>N ;
    // ανάθεση αρχικής τιμής στον αθροιστή
ATHR = 0 ;
    // υπολογισμός αθροίσματος[21]
for ( I = 1; I <= N; I++ )
    {
    // εισαγωγή δεδομένων
     cin >>ARITHM ;
     ATHR = ATHR + ARITHM;
    }
    // εξαγωγή αποτελέσματος
cout<<' το άθροισμα είναι : '<<ATHR ;
return(0);
}
```

ΠΑΡΑΔΕΙΓΜΑ 2.2.7

Να περιγραφεί η διαδικασία **εύρεσης μεγίστου, αγνώστου πλήθους, ακεραίων αριθμών** χρησιμοποιώντας την **ΕΑΓ**, το **λογικό διάγραμμα** ή **διάγραμμα ροής** και τη C++:

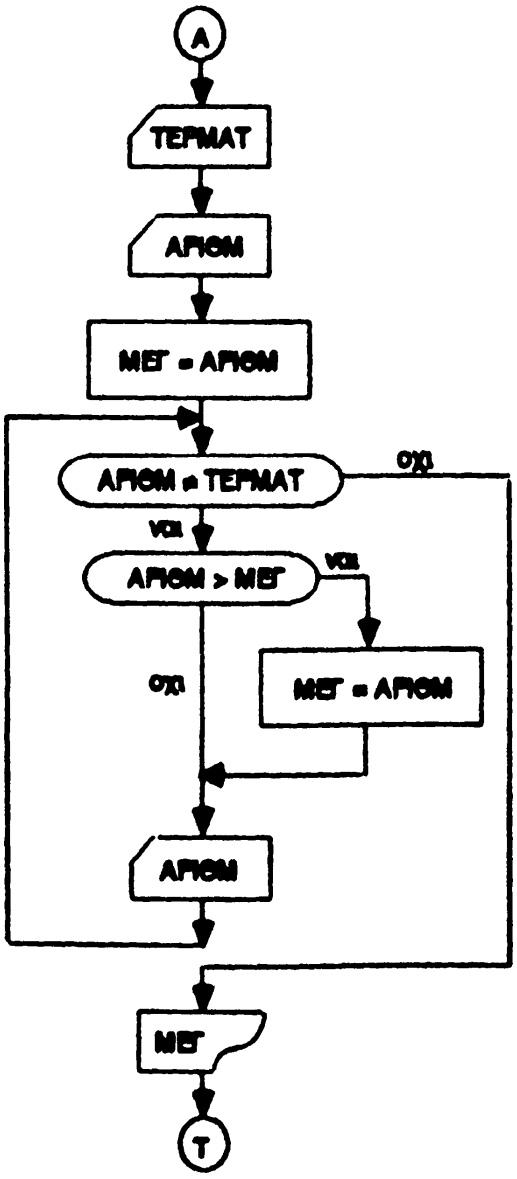
1. σχεδίαση του αλγορίθμου σε **ΕΑΓ**,

```
διαδικασια ΕΥΡΜΕΓ2 ;
 /* δηλώσεις */
δηλωση (ΤΕΡΜΑΤ, ΑΡΙΘΜ, ΜΕΓ) ακερ ;
αρχη
    /* εισαγωγή δεδομένων */
διαβασε ΤΕΡΜΑΤ;
διαβασε ΑΡΙΘΜ ;
  /* ανάθεση αρχικής τιμής στον μέγιστο */
ΜΕΓ ← ΑΡΙΘΜ;
  /*έλεγχος συνέχισης διαδικασίας εύρεσης μεγίστου */
εφοσον ΑΡΙΘΜ ≠ ΤΕΡΜΑΤ επαναλαβε
  /* εύρεση μεγίστου */
     (εαν ΑΡΙΘΜ > ΜΕΓ
```

[21] I++ έκφραση ισοδύναμη της I = I + 1, όπου ++ είναι τελεστής της C (βλπ §4.4.2.1 & πίνακα 4.4.2)

```
                τοτε ΜΕΓ ← ΑΡΙΘΜ;
                /* εισαγωγή νέων δεδομένων */
              διαβασε ΑΡΙΘΜ ;)
            /* εξαγωγή αποτελέσματος */
          τυπωσε ' ο μέγιστος είναι : ', ΜΕΓ ;
        τελος ΕΥΡΜΕΓ2
```

2. σχεδίαση του αλγορίθμου με Λογικό Διάγραμμα ή Διάγραμμα Ροής.



Σημείωση: Η υπολογιστική διαδικασία που περιγράφεται με το παραπάνω διάγραμμα ροής

αποτελείται από μια ακολουθία απλών προτάσεων και βαθμίδων (πρβλ Σχήμα 2.1.1(α)). Αναλυτικά αποτελείται από δύο προτάσεις εισόδου και μια πρόταση ανάθεσης (**διάβασε - εισαγωγή δεδομένων**), μια κανονική βαθμίδα ανακύκλωσης (ανακυκλώσεις αγνώστου πλήθους) (πρβλ Σχήμα 2.1.1(γ)), το σώμα της ανακύκλωσης αποτελείται[22] από μια βαθμίδα εκλογής (με μια απόδοση) και μια πρόταση εισόδου (**υπολόγισε**), τέλος και από μια πρόταση εξόδου (**τύπωσε** ή **γράψε - εξαγωγή αποτελεσμάτων**).

3. μετάφραση ή κωδικοποίηση του αλγορίθμου σε **C++** :

```
// όνομα αρχείου: EYRMEG2.CPP
 # include <iostream.h>
 main( )
 {
      // δηλώσεις
  int TERMAT, ARITHM, MEG ;
      // εισαγωγή δεδομένων
  cin >>TERMAT ;
  cin >>ARITHM ;
      // ανάθεση αρχικής τιμής στον μέγιστο
  MEG = ARITHM ;
      // έλεγχος συνέχισης διαδικασίας εύρεσης μεγίστου
  while ( ARITHM != TERMAT )
     {
      // εύρεση μεγίστου
      if ( ARITHM > MEG )
       {
         MEG = ARITHM ;
       }
     // εισαγωγή νέων δεδομένων
      cin >>ARITHM ;
     }
     // εξαγωγή αποτελέσματος
  cout<<' ο μέγιστος είναι : '<<MEG ;
  return(0);
 }
```

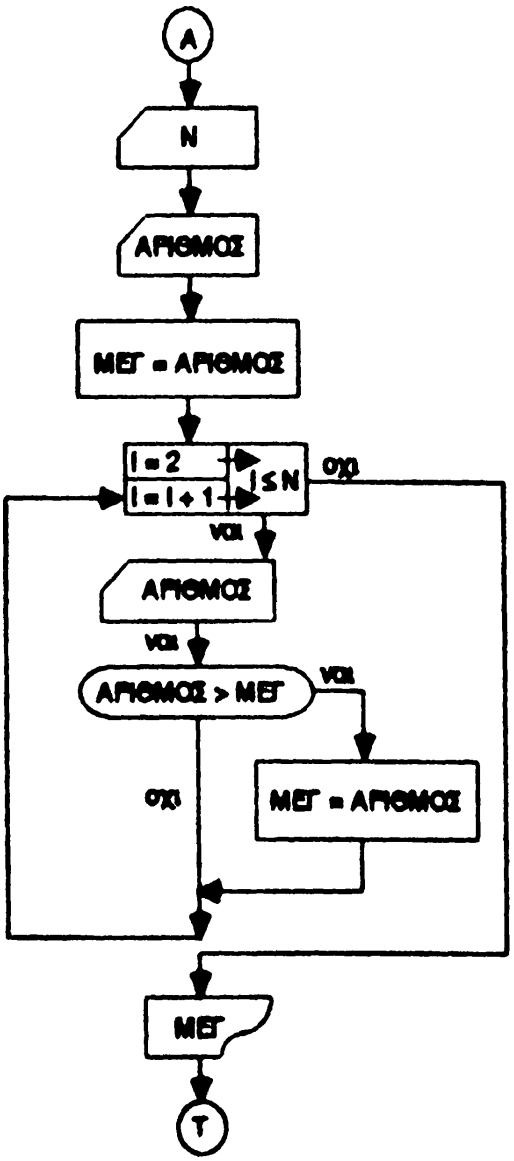
ΠΑΡΑΔΕΙΓΜΑ 2.2.8

Να περιγραφεί η διαδικασία **εύρεσης μεγίστου, γνωστού πλήθους, ακεραίων**

[22] Η πρόταση *εκλογής* (βαθμίδα εκλογής) είναι καθωπισμένη στη πρόταση ανακύκλωσης εισόδου (βαθμίδα ανακύκλωσης).

αριθμών χρησιμοποιώντας το λογικό διάγραμμα ή διάγραμμα ροής, την ΕΑΓ και τη C++:

1. σχεδίαση του αλγορίθμου με **Λογικό Διάγραμμα ή Διάγραμμα Ροής,**



Σημείωση: Η υπολογιστική διαδικασία που περιγράφεται με το παραπάνω διάγραμμα ροής αποτελείται από μια ακολουθία απλών προτάσεων και βαθμίδων (πρβλ Σχήμα 2.1.1(α)). Αναλυτικά αποτελείται από δύο προτάσεις εισόδου και μια πρόταση ανάθεσης (**διάβασε - εισαγωγή δεδομένων**), μια κανονική βαθμίδα ανακύκλωσης (ανακυκλώσεις γνωστού

πλήθους) (πρβλ Σχήμα 2.1.3), το σώμα της ανακύκλωσης αποτελείται[22] από μια πρόταση εισόδου και από μια βαθμίδα εκλογής (με μια απόδοση) (**υπολόγισε**), τέλος και από μια πρόταση εξόδου (**τύπωσε** ή **γράψε - εξαγωγή αποτελεσμάτων**).

2. σχεδίαση του αλγορίθμου σε **ΕΑΓ** ,

```
διαδικασια ΕΥΡΜΕΓ3 ;
 /* δηλώσεις */
δηλωση (N, I, ΑΡΙΘΜ, ΜΕΓ) ακερ ;
αρχη
    /* εισαγωγή δεδομένων */
  διαβασε N;
  διαβασε ΑΡΙΘΜ ;
    /* ανάθεση αρχικής τιμής στον μέγιστο */
  ΜΕΓ ← ΑΡΙΘΜ;
    /* έλεγχος συνέχισης διαδικασίας εύρεσης μεγίστου */
  για I ← 2 εως N επαναλαβε
        ( διαβασε ΑΡΙΘΜ ; /* εισαγωγή νέων δεδομένων */
  /* εύρεση μεγίστου */
        εαν ΑΡΙΘΜ > ΜΕΓ
           τοτε ΜΕΓ ← ΑΡΙΘΜ; )
    /* εξαγωγή αποτελέσματος */
  τυπωσε ' ο μέγιστος είναι : ', ΜΕΓ ;
τελος ΕΥΡΜΕΓ3
```

3. μετάφραση ή κωδικοποίηση του αλγορίθμου σε **C++** :

```
// όνομα αρχείου: EYRMEG3.CPP
 # include <iostream.h>
 main( )
 {
      // δηλώσεις
  int N, I, ARITHM, MEG ;
      // εισαγωγή δεδομένων
  cin >>N ;
  cin >>ARITHM ;
      // ανάθεση αρχικής τιμής στον μέγιστο
```

[22] Η πρόταση εκλογής (βαθμίδα εκλογής) είναι καθωπισμένη στην πρόταση ανακύκλωσης για (βαθμίδα ανακύκλωσης).

```
MEG = ARITHM ;
    // έλεγχος συνέχισης διαδικασίας εύρεσης μεγίστου
for ( I = 2; I <= N; I++ )
    {
    // εισαγωγή δεδομένων
     cin >>ARITHM ;
    // εύρεση μεγίστου
     if ( ARITHM > MEG )
      {
        MEG = ARITHM ;
      }
    }
    // εξαγωγή αποτελέσματος
cout<<' ο μέγιστος είναι : '<<MEG ;
return(0);
}
```

2.3 Μεθοδολογία Ανάπτυξης Προγράμματος

Ένα πρόγραμμα είναι ένας κωδικοποιημένος σε γλώσσα προγραμματισμού αλγόριθμος, δηλαδή μια διαδικασία επίλυσης κάποιου προβλήματος που εκφράζει κατά τρόπο ιεραρχημένο και αναλυτικό την περιγραφή των βημάτων για τον αυτόματο υπολογισμό της λύσης του προβλήματος ή όπως λέμε της υπολογιστικής επίλυσής του.

Έτσι, η σύνθεση προγραμμάτων, ο προγραμματισμός δηλαδή, είναι άρρηκτα συνδεδεμένος και με τη μαθηματική λογική και δεοντολογία, από την οποία εκπορεύεται και ολοκληρώνει στην πράξη.

Η μεθοδολογία λοιπόν του προγραμματισμού, ανάγεται σε πρώτη φάση στη μεθοδολογία περιγραφής της σύνθεσης των βημάτων της υπολογιστικής απόδειξης, στην περίπτωσή μας της υπολογιστικής επίλυσης του προβλήματος, και σε δεύτερη φάση στην κωδικοποίησή της.

Η πρώτη φάση είναι και το δυσκολότερο μέρος του προγραμματισμού και ίσως το πιό ακαθόριστο ως προς τη διατύπωση γενικών οδηγιών για την εξεύρεση μεθόδου επίλυσης ενός προβλήματος.

Γενικά η σύνθεση της απόδειξης κάποιου προβλήματος εξαρτάται από την ανάλυση των δεδομένων του. Όσο η ανάλυση γίνεται προσεκτικότερη, μεθοδικότερη, πληρέστερη και ακριβέστερη τόσο η απόδειξη γίνεται προφανέστερη. Η γνώση λοιπόν των ιδιοτήτων των δεδομένων αλλά και η κατανόηση των ζητουμένων κάποιου προβλήματος, οδηγούν στη σχεδίαση της στρατηγικής της απόδειξής του.

Στα προβλήματα υπολογιστικής - πληροφορικής, όμως, δεν αρκούν μόνο τα παραπάνω για τον σχεδιασμό της στρατηγικής τής επίλυσής τους, πρέπει να λαμβάνονται υπόψη και άλλα στοιχεία για τα δεδομένα, αφού γίνεται μηχανικός χειρισμός τους.

Για παράδειγμα, τα στοιχεία αυτά, για τα δεδομένα κάποιου αριθμητικού προβλήματος μπορούν να ταξινομηθούν σε τρείς κατηγορίες :

1. **στην εισαγωγή τους**, αναλυτικότερα,

1α. τρόπος εισαγωγής των δεδομένων (από άλλο πρόγραμμα ή υποπρόγραμμα, είναι εναποθηκευμένα στη μνήμη του υπολογιστή ή εισάγονται από το πληκτρολόγιο),

1β. σε περίπτωση εισαγωγής τους από το πληκτρολόγιο είναι γνωστού ή αγνώστου πλήθους, στη δεύτερη περίπτωση πως θα επισημαίνεται το τέλος τους, επίσης το είδος των δεδομένων (ακέραιοι κλπ.),

1γ. είναι πολλοί ή λίγοι αριθμοί (δεδομένα) δηλαδή πάνω από 1000 ή κάτω από 100,

1δ. ποιός καθορίζει τη μορφή που θα δίνονται οι αριθμοί,

2. **στα σφάλματα εισαγωγής, εξαγωγής και επεξεργασίας τους**,

2α. το είδος του σφάλματος που μπορεί να γίνει κυρίως κατά την εισαγωγή,

2β. τον τρόπο που θα ενεργήσουμε σε περίπτωση σφάλματος, θα συνεχίσουμε ή θα τερματίσουμε την εκτέλεση,

2γ. το είδος των πληροφοριών σφαλμάτων που πρέπει να εκτυπωθούν και

2δ. αν υπάρχει θέμα υπερχείλισης,

3. **στην εξαγωγή τους**,

3α. τρόπος παρουσίασης των αποτελεσμάτων,

3β. είδος των πινάκων που θα γίνει η εκτύπωση (πχ. με πολλές ή λίγες στήλες κλπ.) και

3γ. εξαγωγή δεδομένων.

Άλλοι παράγοντες που επηρεάζουν και εξαρτώνται από αυτή καθεαυτή τη σχεδίαση

της στρατηγικής, είναι και οι δομές των δεδομένων των προβλημάτων. Για παράδειγμα από τη στρατηγική επίλυσης κάποιου προβλήματος εξαρτάται αν θα χρησιμοποιήσουμε, για την κατασκευή ενός αρχείου, γραμμικό πίνακα διαδοχικής χορήγησης μνήμης, ή ένα συνδετικό γραμμικό πίνακα ή ένα δυαδικό δένδρο.

Η κωδικοποίηση είναι η τελευταία και η πιό μηχανική φάση του προγραμματισμού, δεν απαιτεί τη χρήση ευφυίας, σίγουρα όμως απαιτεί γνώση και προσοχή. Έχουμε ήδη καλύψει από τις προηγούμενες παραγράφους το θέμα αυτό. Δεν είναι ανάγκη να επαναλάβουμε τα περί πειθαρχίας και υπομονής, μόνο ότι χρειάζεται προσοχή:
1. στον σωστό καθορισμό των προγραμματικών σχημάτων κατά τη σύνθεση του αλγορίθμου, για να μην υπάρξουν δυσκολίες στην κωδικοποίηση, παρά τις τυχούσες ιδιορυθμίες της γλώσσας προγραμματισμού που χρησιμοποιούμε και
2. στην προσιδιοποίηση, ότι ίσως χρειαστούν αρκετές επαναλήψεις στον σχεδιασμό του αλγορίθμου και στο γράψιμο του αντίστοιχου προγράμματος, έως ότου προκύψει η επιθυμητή τελική μορφή.

Σημείωση : Σχόλια για τους αρχαρίους.

Θέλουμε να επισημάνουμε ότι, πράγματι, υπάρχει η παραπλανητική τάση, δυστυχώς όχι μόνο μεταξύ των αρχαρίων, κατά το γράψιμο του προγράμματος να αγνοούν τον καθορισμό των στόχων και τον σχεδιασμό του προγράμματος, δηλαδή να μην γράφουν πρώτα τον αλγόριθμο. Αυτό βέβαια, δεν δημιουργεί πρόβλημα όταν, στην αρχή τουλάχιστο, τα προγράμματα είναι απλά χωρίς ιδιαίτερες απαιτήσεις, οπότε είναι μάλλον εύκολο να υπάρχει όλη η διαδικασία στο μυαλό και αν γίνει κάποιο λάθος, να εντοπίζεται και να διορθώνεται εύκολα. Όμως, όσο τα προγράμματα γίνονται πιό σύνθετα τόσο η διαδικασία αυτή αποτυγχάνει και καταδικάζει τους προγραμματίζοντες στο να χάνουν πολύτιμο χρόνο, στο να μπερδεύονται και τέλος να εκνευρίζονται και να απογοητεύονται καθώς τα προγράμματά τους δεν είναι αποτελεσματικά. Επίσης, υπάρχει μεγαλύτερη ευχέρεια όταν προηγείται ο αλγόριθμος του προγράμματος, να γυρίζετε πίσω, όσες φορές χρειάζεται, στα προηγούμενα βήματα που ακολουθήσατε, είτε γιατί ανακαλύψατε ότι στη σχεδίαση υπάρχει κάποιο λάθος, είτε γιατί υπάρχει καλλίτερος τρόπος να το ή να τα προσεγγίσετε.

Η παρότρυνσή μας λοιπόν είναι, εξασκηθείτε και αναπτύξτε την ικανότητα να αναλύσετε διεξοδικά το πρόβλημα και να σχεδιάζετε σωστά τον αλγόριθμο πριν γράψετε το (αποτελεσματικό) πρόγραμμα. Για τη σχεδίαση του αλγορίθμου και την κωδικοποίησή του

χρησιμοποιείτε ανεπιφύλακτα μολύβι και χαρτί. Κάντε το, και θα κερδίσετε χρόνο και ικανοποίηση.

Με το παράδειγμα που θα ακολουθήσει θα προσπαθήσουμε να δώσουμε κάποιες κατευθύνσεις για το πώς μπορεί η πρώτη φάση του προγραμματισμού να ξεπερνιέται σχετικά εύκολα, μέχρι να αποκτηθεί η απαιτούμενη εμπειρία.

ΠΑΡΑΔΕΙΓΜΑ 2.3.1 Να γραφεί πρόγραμμα σε C++ που υπολογίζει και τυπώνει τον μέσον όρο κάποιων θετικών πραγματικών αριθμών, το πολύ δώδεκα ψηφίων.

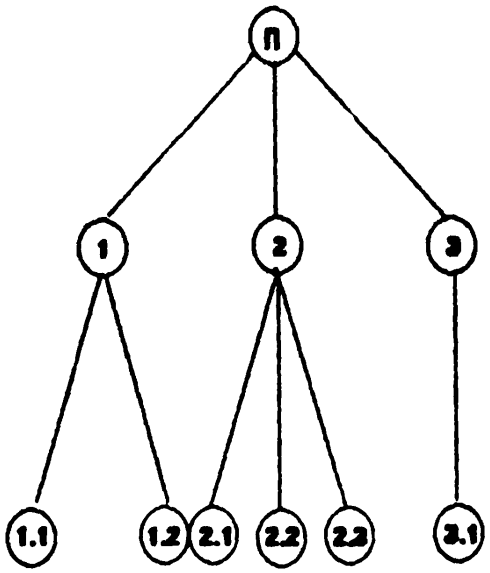
Παρατηρήσεις :

1. η διαδικασία της επινόησης και σχεδίασης της λύσης πρέπει να χωριστεί τελείως από την περιγραφή της λύσης αυτής σε κάποια γλώσσα προγραμματισμού, στην περίπτωσή μας C++.
2. η σχεδίαση της λύσης πρέπει να γίνει πρώτα με βάση ένα πρωτόγονο υπολογιστικό μοντέλο,
3. η προσοχή μας θα επικεντρώνεται πρώτα στο τι θέλουμε να κάνουμε και μετά στο πως θα το κάνουμε. Το τι θέλουμε να κάνουμε θα το περιγράφουμε πρώτα μονολεκτικά με τίτλους, που θα αντιστοιχούν σε εντολές ανωτέρου επιπέδου πχ., βρήτε, αντιστρέψετε, κα. Οι εντολές αυτές θα αναλύονται στη συνέχεια σε εντολές κατωτέρου επιπέδου και σταδιακά θα καθορίζουν το πως θα γίνει, με σχήματα που θα πειθαρχούν στη γραφή του συστηματικού προγραμματισμού. Έτσι αναπτύσσεται ο αλγόριθμος.

Απόδειξη : Σύμφωνα με αυτά που υποστηρίξαμε διαμορφώνουμε και την απόδειξη.

Α. **"τι κάνουμε"** (βλπ και Σχήμα 2.3.1) ,

1. εισαγωγή δεδομένων,
   1.1 διάβασε ΤΕΡΜΑΤΙΣΤΗ
   1.2 διάβασε ΑΡΙΘΜΟΥΣ
2. υπολογισμός ΜΕΣΟΥ ΟΡΟΥ,
   2.1 υπολόγισε ΑΘΡΟΙΣΜΑ
   2.2 υπολόγισε ΠΛΗΘΟΣ ΑΡΙΘΜΩΝ
   2.3 υπολόγισε ΜΕΣΟ ΟΡΟ
3. εξαγωγή αποτελέσματος.
   3.1 τύπωσε ΜΕΣΟ ΟΡΟ



ΣΧΗΜΑ 2.3.1 : Είναι το δένδρο ανάπτυξης της λύσης του προβλήματος. Η λύση του προβλήματος αναλύεται από πάνω προς τα κάτω. Η ρίζα του δένδρου είναι εντολη ανωτάτου επιπέδου που λέει:

"να λυθεί το πρόβλημα του υπολογισμού του μέσου όρου".

Κάθε στάθμη του δένδρου είναι και μια περαιτέρω ανάλυση. Οι τερματικοί κόμβοι του μπορούν να γραφούν στην ΕΑΓ και τελικά στη γλώσσα εκτέλεσης.

Β. "πως το κάνουμε",

διαβασε ΤΕΡΜΑΤ

διαβασε ΑΡΙΘΜ

J = 1

S = 0

εφοσον ΑΡΙΘΜ ≠ ΤΕΡΜΑΤ **επαναλαβε**
    S = S + ΑΡΙΘΜ
    διαβασε ΑΡΙΘΜ
    J = J + 1

N = J - 1

ΜΕΣΟΡ = S / N

τυπωσε N, ΜΕΣΟΡΟΣ

Γ. "ο αλγόριθμος σε ΕΑΓ",

διαδικασια ΕΥΡΕΣΗΣ_ΜΕΣΟΡΟΥ;

```
/* δηλώσεις */
δηλωση  (ΑΡΙΘΜ, S, ΜΕΣΟΡ) πραγματικ;
δηλωση (ΤΕΡΜΑΤ, N, J) ακερ;

αρχη
/* εισαγωγή δεδομένων */
 διαβασε ΤΕΡΜΑΤ;
 διαβασε ΑΡΙΘΜ;

/* ανάθεση αρχικών τιμών στους καταμετρητή & αθροιστή */
 J ← 1;
 S ← 0;

/* έλεγχος συνέχισης της διαδικασίας άθροισης/καταμέτρησης */
 εφοσον ΑΡΙΘΜΟ ≠ ΤΕΡΜΑΤ επαναλαβε

/* υπολογισμός αθροίσματος */
      ( S ← S + ΑΡΙΘΜ;

/* εισαγωγή νέων δεδομένων */
       διαβασε ΑΡΙΘΜ;

/* καταμέτρηση νέων δεδομένων */
       J ← J + 1; )

/* υπολογισμός πλήθους */
 N ← J - 1;

/* υπολογισμός μέσου όρου */
 ΜΕΣΟΡ ← S / N;

/* εξαγωγή αποτελεσμάτων */
 γραψε N, ΜΕΣΟΡ;

τελος ΕΥΡΕΣΗ_ΜΕΣΟΡΟΥ
```

Δ. "κωδικοποίηση σε C++"

```
// όνομα αρχείου: MESOROS2.CPP
# include <iostream.h>
main( )
{
      // δηλώσεις
 float TERMAT, N, J ;
```

```
int TERMAT, ARITHM, MEG ;
      // εισαγωγή δεδομένων
cin >>TERMAT ;
cin >>ARITHM ;
    // ανάθεση αρχικών τιμών στους καταμετρητή & αθροιστή
J = 1 ;
S = 0 ;
    // έλεγχος συνέχισης της διαδικασίας άθροισης/καταμέτρησης
while ( ARITHM != TERMAT )
 {
      // υπολογισμός αθροίσματος
   S = S + ARITHM ;
      // εισαγωγή νέων δεδομένων
   cin >>ARITHM ;
      // καταμέτρηση νέων δεδομένων
    J = J +1;
 }
      // υπολογισμός πλήθους
N = N - 1;
      // υπολογισμός μέσου όρου
MESOR = S / N ;
      // εξαγωγή αποτελέσματος
 cout<<' ο μέσος όρος είναι : '<<MESOR ;
 return(0);
}
```

Σημείωση: Στο σημείο αυτό θα θέλαμε να κάνουμε μια ακόμα σημαντική επισήμανση, συνήθως λοιπόν, κατά τη σχεδίαση του αλγορίθμου ακολουθούμε μια διαδικασία καταμερισμού του σε ενότητες. Η διαδικασία αυτή λέγεται **βελτιστοποίηση κατά βήματα** (step-wise refinement) και προβλέπει τον χωρισμό του προβλήματος σε υποπροβλήματα μέχρις ότου η λύση σε καθένα από αυτά να είναι προφανής. Αυτό κάνει την αντιμετώπιση του προβλήματος πολύ πιό εύκολη, αφού και πάλι μια ενότητα μπορεί να χωριστεί σε άλλες υποενότητες κ.ο.κ.. Για παράδειγμα ένας αλγόριθμος μπορεί να χωριστεί σε τρείς κύριες λειτουργίες, την εισαγωγή δεδομένων (διάβασε), την επεξεργασία τους (υπολόγισε) και τέλος την εξαγωγή των αποτελεσμάτων (τύπωσε). Είναι λοιπόν λογικό να χωρίζεται, να οργανώνεται ο αλγόριθμος καταρχήν σε τρείς ενότητες, όπως έχουμε ξαναναφέρει (§1.2), αντίστοιχες με αυτές τις τρείς λειτουργίες. Έτσι στον προηγούμενο

αλγόριθμο (πχ. 2.3.1), στη σχεδίασή του (Α."τι κάνουμε") η ενότητα 2, "υπολογισμός ΜΕΣΟΥ ΟΡΟΥ", αναλύεται στην ενότητα 2.1, "υπολόγισε ΑΘΡΟΙΣΜΑ", στην ενότητα 2.2, "υπολόγισε ΠΛΗΘΟΣ ΑΡΙΘΜΩΝ" και τέλος στην ενότητα 2.3, "υπολόγισε ΜΕΣΟ ΟΡΟ".

2.4 Ασκήσεις

1α. Να επεκταθεί η διαδικασία **αθροίσματος δύο ακεραίων αριθμών**, έτσι ώστε να μπορεί να αθροίζει όσα ζεύγη ακεραίων θέλουμε. Η διαδικασία θα διακόπτεται αν ο ένας από τους δύο αριθμούς είναι ίσος με μηδέν. Η περιγραφή της διαδικασίας να γίνει χρησιμοποιώντας την **ΕΑΓ**, το **λογικό διάγραμμα** ή **διάγραμμα ροής** και τη C++.

1β. Να ξανασχεδιαστεί η προηγούμενη διαδικασία έτσι ώστε να διακόπτεται όταν ένας **διακόπτης** (ΔΙΑΚΟΠΤ) παίρνει τη τιμή μηδέν.

1.γ Να επεκταθεί η διαδικασία **αθροίσματος δύο ακεραίων αριθμών**, έτσι ώστε να μπορεί να αθροίζει γνωστού πλήθους (Ν) ζεύγη ακεραίων. Η περιγραφή της διαδικασίας να γίνει χρησιμοποιώντας την **ΕΑΓ**, το **λογικό διάγραμμα** ή **διάγραμμα ροής** και τη C++.

2α. Να επεκταθεί η διαδικασία **εύρεσης του μεγίστου δύο δοθέντων ακεραίων αριθμών**, έτσι ώστε να μπορεί να συγκρίνει όσα ζεύγη ακεραίων θέλουμε. Η διαδικασία θα διακόπτεται αν ο ένας από τους δύο αριθμούς είναι ίσος με μηδέν. Η περιγραφή της διαδικασίας να γίνει χρησιμοποιώντας την **ΕΑΓ**, το **λογικό διάγραμμα** ή **διάγραμμα ροής** και τη C++.

2β. Να ξανασχεδιαστεί η προηγούμενη διαδικασία έτσι ώστε να διακόπτεται όταν ένας **διακόπτης** (ΔΙΑΚΟΠΤ) παίρνει τη τιμή μηδέν.

2γ. Να επεκταθεί η διαδικασία **εύρεσης του μεγίστου δύο δοθέντων ακεραίων αριθμών**, έτσι ώστενα μπορεί να σγκρίνει γνωστού πλήθους (Ν) ζεύγη ακεραίων. Η περιγραφή της διαδικασίας να γίνει χρησιμοποιώντας την **ΕΑΓ**, το **λογικό διάγραμμα** ή **διάγραμμα ροής** και τη C++.

3. Να περιγραφεί η διαδικασία **εύρεσης του μεγίστου τριών δοθέντων ακεραίων**

αριθμών χρησιμοποιώντας την **ΕΑΓ**, το **λογικό διάγραμμα** ή **διάγραμμα ροής** και τη C++.

4. Να περιγραφεί η διαδικασία **αθροίσματος**, από **άγνωστο πλήθος** δοθέντων **ακεραίων αριθμών** και **καταμέτρησης του πλήθους τους**, χρησιμοποιώντας την **ΕΑΓ**, το **λογικό διάγραμμα** ή **διάγραμμα ροής** και τη C++.

5. Να περιγραφεί η διαδικασία **αθροίσματος**, από **γνωστό πλήθος** δοθέντων **ακεραίων αριθμών** και **υπολογισμού του μέσου όρου τους**, χρησιμοποιώντας την **ΕΑΓ**, το **λογικό διάγραμμα** ή **διάγραμμα ροής** και τη C++.

6α. Να περιγραφεί η διαδικασία **εύρεσης μεγίστου** από **άγνωστο πλήθος** δοθέντων **ακεραίων αριθμών**, **εντοπισμού της τάξης του μεγίστου** και **καταμέτρησης ή υπολογισμού του πλήθους των δοθέντων αριθμών**, χρησιμοποιώντας την **ΕΑΓ**, το **λογικό διάγραμμα** ή **διάγραμμα ροής** και τη C++.

6β. Να περιγραφεί η διαδικασία **εύρεσης μεγίστου** από **γνωστό πλήθος** δοθέντων **ακεραίων αριθμών** και **εντοπισμού της τάξης του ελαχίστου** χρησιμοποιώντας την **ΕΑΓ**, το **λογικό διάγραμμα** ή **διάγραμμα ροής** και τη C++.

6γ. Να περιγραφεί η διαδικασία **εύρεσης μεγίστου** και **ελαχίστου** από **άγνωστο πλήθος** δοθέντων **ακεραίων αριθμών**, **εντοπισμού της τάξης του μεγίστου** και **ελαχίστου** και τέλος **καταμέτρησης ή υπολογισμού του πλήθους των δοθέντων αριθμών**, χρησιμοποιώντας την **ΕΑΓ**, το **λογικό διάγραμμα** ή **διάγραμμα ροής** και τη C++.

7α. Να περιγραφεί η διαδικασία **καταμέτρησης ή υπολογισμού του πλήθους των θετικών από άγνωστο πλήθος δοθέντων ακεραίων αριθμών**, χρησιμοποιώντας την **ΕΑΓ**, το **λογικό διάγραμμα** ή **διάγραμμα ροής** και τη C++.

7β. Να περιγραφεί η διαδικασία **καταμέτρησης ή υπολογισμού του πλήθους των αρνητικών από γνωστό πλήθος δοθέντων ακεραίων αριθμών**, χρησιμοποιώντας

την **ΕΑΓ**, το **λογικό διάγραμμα** ή **διάγραμμα ροής** και τη C++.

7γ. Να περιγραφεί η διαδικασία **καταμέτρησης ή υπολογισμού του πλήθους των θετικών και των αρνητικών από άγνωστο πλήθος δοθέντων ακεραίων αριθμών**, χρησιμοποιώντας την **ΕΑΓ**, το **λογικό διάγραμμα** ή **διάγραμμα ροής** και τη C++.

7δ. Να περιγραφεί η διαδικασία **καταμέτρησης ή υπολογισμού του πλήθους των θετικών και των αρνητικών από γνωστό πλήθος δοθέντων ακεραίων αριθμών**, χρησιμοποιώντας την **ΕΑΓ**, το **λογικό διάγραμμα** ή **διάγραμμα ροής** και τη C++.

8α. Να περιγραφεί η διαδικασία **καταμέτρησης ή υπολογισμού του πλήθους των αρτίων από άγνωστο πλήθος δοθέντων ακεραίων αριθμών**, χρησιμοποιώντας την **ΕΑΓ**, το **λογικό διάγραμμα** ή **διάγραμμα ροής** και τη C++.

8β. Να περιγραφεί η διαδικασία **καταμέτρησης ή υπολογισμού του πλήθους των περιττών από γνωστό πλήθος δοθέντων ακεραίων αριθμών**, χρησιμοποιώντας την **ΕΑΓ**, το **λογικό διάγραμμα** ή **διάγραμμα ροής** και τη C++.

8γ. Να περιγραφεί η διαδικασία **καταμέτρησης ή υπολογισμού του πλήθους των αρτίων και των περιττών από άγνωστο πλήθος δοθέντων ακεραίων αριθμών**, χρησιμοποιώντας την **ΕΑΓ**, το **λογικό διάγραμμα** ή **διάγραμμα ροής** και τη C++.

8δ. Να περιγραφεί η διαδικασία **καταμέτρησης ή υπολογισμού του πλήθους των θετικών και των περιττών από γνωστό πλήθος δοθέντων ακεραίων αριθμών**, χρησιμοποιώντας την **ΕΑΓ**, το **λογικό διάγραμμα** ή **διάγραμμα ροής** και τη C++.

9. Να περιγραφεί η διαδικασία **καταμέτρησης ή υπολογισμού του πλήθους των αρτίων θετικών από άγνωστο πλήθος δοθέντων ακεραίων αριθμών**, χρησιμοποιώντας την **ΕΑΓ**, το **λογικό διάγραμμα** ή **διάγραμμα ροής** και τη C++.

10. Να περιγραφεί η διαδικασία **εύρεσης μεγίστου** από **άγνωστο πλήθος** δοθέντων **ακεραίων αριθμών που ανήκουν στο διάστημα [125,350] , εντοπισμού της τάξης του μεγίστου** και **καταμέτρησης ή υπολογισμού του πλήθους: 1. όλων**

των δοθέντων αριθμών και **2. των αριθμών που βρίσκονται στο διάστημα [125,350]** , χρησιμοποιώντας την **ΕΑΓ**, το **λογικό διάγραμμα** ή **διάγραμμα ροής** και τη C++.

11. Να περιγραφεί η διαδικασία **υπολογισμού του μέσου όρου ακεραίων αριθμών μεγαλυτέρων του 299,** οι οποίοι δίδονται από **γνωστό πλήθος ακεραίων αριθμών,** χρησιμοποιώντας την **ΕΑΓ**, το **λογικό διάγραμμα** ή **διάγραμμα ροής** και τη C++.

ΚΕΦΑΛΑΙΟ 3

ΕΙΣΑΓΩΓΗ ΣΤΗΝ C++

3.1 Εισαγωγή

Η C++ έχει βαθιά τις ρίζες της στην C, η ιστορία της λοιπόν, είναι άρρηκτα συνδεδεμένη με αυτή της C, αφού η C είναι υποσύνολο της C++. Οποιαδήποτε ιστορική αναδρομή της C++ περνά μέσα από την αντίστοιχη της C. Η γλώσσα προγραμματισμού C σχεδιάστηκε καταρχήν για να ενθαρρύνει την οικονομία στις εκφράσεις σε μια μεγάλη ποικιλία εφαρμογών. Η πρώτη εφαρμογή της C ήταν ως γλώσσα προγραμματισμού συστημάτων, αφού χρησιμοποιήθηκε πάνω από το 90% για τη γραφή του κώδικα του λειτουργικού συστήματος UNIX. Οι εφαρμογές της διαδίδονται ευρέως και δυναμικά, αφενός διότι οι τελεστές της, οι τύποι δεδομένων της καθώς και οι συναρτήσεις βιβλιοθήκης της είναι ασυνήθιστα πλούσιοι, αφετέρου λόγω της μεγάλης φορητότητάς της. Καθώς η C είναι η γλώσσα που γράφτηκε το UNIX, η χρησιμοποίηση αυτή καθεαυτή του UNIX, στο ξεκίνημα της C, στην επεξεργασία δεδομένων και στον επιστημονικό προγραμματισμό αναμφίβολα επαύξησαν τη χρήση της C ακόμα και στους τομείς αυτούς. Η C++ η οποία είναι εξέλιξη της C, διαθέτει επιπλέον ευελιξία, αρθρωτή δόμηση και εν γένει νοοτροπία πραγματικής γλώσσας αντικειμενοστρεφούς προγραμματισμού. Η C++ κυριάρχησε σε μικρό χρονικό διάστημα στον χώρο της ανάπτυξης λογισμικού τόσο στο UNIX όσο και στο MS-DOS καθώς και στο OS/2. Πολλοί επαγγελματίες προγραμματιστές προβλέπουν ότι η C++ θα καταλάβει τη θέση της C στη βιομηχανία λογισμικού.

3.2 Σύντομη Ιστορική Αναδρομή της C++

Οποιαδήποτε αναφορά στην εξέλιξη της C++ περνά μέσα από την αναφορά στην εξέλιξη της C. Η ιστορική εξέλιξη της C είναι παράλληλη με αυτή του UNIX, όπως προαναφέρθηκε στην Εισαγωγή.

Το 1967, η Bell Laboratories αναζήτησε μια παραλλαγή για το λειτουργικό σύστημα Multics του υπολογιστή PDP-7. Έτσι γράφτηκε σε συμβολική γλώσσα μια αυθεντική έκδοση του λειτουργικού συστήματος UNIX. Την εποχή εκείνη μια πειραματική

γλώσσα αναπτυσσόταν από τον Kenneth Thompson. Η γλώσσα αυτή ονομάστηκε B[23] και σχεδιάστηκε βάσει της συχγρόνου τότε γλώσσας προγραμματισμού συστημάτων BCPL[24].

Το **1972**, η γλώσσα C σχεδιάστηκε ως μια επέκταση της B · η πρωταρχική διαφορά ήταν ότι η C είχε μια επεκτεταμένη συλλογή πρότυπων τύπων, αφού η B ουσιαστικά υστερούσε ή μάλλον ήταν χωρίς τύπους.

Αμέσως μετά, το **1973**, το UNIX επεκτάθηκε ουσιαστικά αφεαυτού, περισσότερο δε από το 90% αυτής της νέας έκδοσης γράφτηκε σε C. Εξαιτίας αυτής της απελευθέρωσης από τη συμβολική γλώσσα (assembly), το UNIX, ,κατ' επέκταση και η C, έγιναν επαρκώς φορητές. Το τελευταίο είχε σαν αποτέλεσμα να υιοθετηθούν γρήγορα από ποκίλα κύρια συστήματα και σχεδόν αμέσως άρχισε να διαδίδεται η χρήση τους. Η γλώσσα C και η βιβλιοθήκη των συναρτήσεών της έχουν ήδη μια σταθερή πορεία. Όλα τα υποσύνολα των επαυξήσεων προσαρτήθηκαν στη βιβλιοθήκη με τρόπο που να εξασφαλίζεται η παραπέρα συμβατότητα με τα υπάρχοντα προγράμματα.

Το **19976-1977** , το UNIX μεταφέρεται στο σύστημα VAX, έτσι μια νέα έκδοση αναπτύχθηκε ανεξάρτητα στο Πανεπιστήμιο της California στο Berkeley.

Στα **τέλη της δεκαετίας του 1970 αρχές της δεκαετίας του 1980**, η C και το UNIX εξαπλώθηκαν περαιτέρω σε διάφορες μηχανές, η ποικιλία των οποίων διέτρεχε τους μεγάλους (κεντρικούς) υπολογιστές (mainframes) μέχρι τους μικρουπολογιστές (micros). Η C υποστηρίχθηκε ανεξάρτητα και έξω από το περιβάλλον UNIX, μεταγλωττιστές για πολλές μηχανές ήταν πλέον διαθέσιμοι και μάλιστα κάτω από τα δικά τους λειτουργικά συστήματα. Η C είναι αποδοτική (μερικές φορές καλείται υψηλή χαμηλού επιπέδου γλώσσα, λόγω της ταχύτητας εκτέλεσης), είναι ευέλικτη και περιέχει τα σωστά συστατικά γλώσσας, που τις επιτρέπουν να συντηρείται (βλπ Πίνακα 1.4.2 σελ 40).

Μια εξαιρετική και ενδιαφέρουσα επισκόπιση για την καταγωγή και τα χαρακτηριστικά του UNIX και της C μπορεί να βρεί ο μελετητής σε μια σειρά άρθρων στο Bell Systems Journal[25], ενώ το πρότυπο κείμενο αναφορά της C είναι των Keringham και Ritchie [26].

---

[23] S.C. Johnson and B.W. Kernighan, "The Programming Language B", Computer Science Tech. Rep. No. 8, Bell Laboratories, Murray Hill, N.J., January 1973.

[24] M. Richards, "BCPL: A Tool for Compiler Writing and Systems Programming." Proc AFIPS SJCC 34:557-566 (1966).

[25] D.M.Ritchie, et al., "The C Programming Language," Bell Systems Tech. J. 57(6): 1991-2019 (August 1978).

[26] B.W. Kernighan and D.M.Ritchie, The C Programming Language, Prentice-Hall, Englewood Cliffs, N.J., 1978. (Αποκλειστικότητα για την Ελληνική Γλώσσα, Η Γλώσσα Προγραμματισμού C - ANSI C, ΚΛΕΙΔΑΡΙΘΜΟΣ, δεύτερη έκδοση 1990).

Το **1980**, ο Bjourn Stroustrup, που εργαζόταν στην American Telephone and Telegraph Incorporated (AT&T), προχώρησε την C κατά ένα βήμα, πρόσθεσε δυνατότητες για να αντισταθμίσει ορισμένα από τα λάθη της. Οι δυνατότητες[27] εντοπίζονται στο ότι μπορεί ο προγραμματιστής να δημιουργήσει τους δικούς του τύπους δεδομένων καθώς και ειδικούς τελεστές και συναρτήσεις που τους επεξεργάζονται. Έτσι διαμορφώθηκε μια νέα προσέγγιση στις δομές δεδομένων, ο προσανατολισμός στον αντικειμενοστρεφή προγραμματισμό[28]. Έτσι προέκυψε η C++, μια "καλλίτερη C", όπως χαρακτηρίζεται η C++. Η

---

[27] Οι δυνατότητες αυτές επεκτάθηκαν και σε άλλες γλώσσες, π.χ. στην Smalltalk

[28] **Αντικειμενοστρεφής προγραμματισμός** (ΑΝΤΚ.Π, object oriented programming) είναι μια σχετικά καινούργια μέθοδος σχεδίασης και ανάπτυξης εφαρμογών. Ένας καινούργιος τρόπος σκέψης για την επίλυση προβλημάτων με Η/Υ. Είναι μια μέθοδος που μιμείται τον τρόπο με τον οποίο σκέπτεται ο άνθρωπος και διαχωρίζει τα πράγματα μεταξύ τους, έτσι ο ΑΝΤΚ.Π αντί να προσπαθεί να προσαγάγει το πρόβλημα σε κάτι που είναι οικείο στον Η/Υ προτρέπει τον Η/Υ να προσαρμοστεί στο πρόβλημα.

Σημειώνουμε για την πληρότητα ότι ο γνωστός "παραδοσιακός" προγραμματισμός λέγεται **διαδικασιακός προγραμματισμός** (ΔΔΚ.Π, procedural programming). Υπενθυμίζουμε ότι στον ΔΔΚ.Π καθορίζεται η ακολουθία από υπολογιστικά βήματα τα οποία περιγράφουν τον τρόπο με τον οποίο λύνεται ένα πρόβλημα (αλγοριθμικός τρόπος). Στον ΔΔΚ.Π το πρόγραμμα αποτελείται από μια ακολουθία εντολών που περιγράφουν το πώς λύνεται ένα πρόβλημα και εκτελούνται μια-μια με τη σειρά που έχουν γραφεί.

Στον ΑΝΤΚ.Π εξετάζεται το πρόβλημα για να προσδιοριστούν οι ανεξάρτητες οντότητες, που συσχετίζονται με το πρόβλημα. Τις ανεξάρτητες αυτές οντότητες (ενότητες) δεν τις διαχωρίζουμε με γνώμονα τον τρόπο οργάνωσής τους στον Η/Υ, αλλά τις ξεχωρίζουμε επειδή παρουσιάζουν μια φυσική συνοχή και διαθέτουν συγκεκριμένα όρια, που τις διαφοροποιούν από το υπόλοιπο πρόβλημα. Οι οντότητες αυτές οργανώνονται σαν αντικείμενα (objects) στο πρόγραμμα του Η/Υ. Η αντιστοιχία μεταξύ αυτών των οντοτήτων, που συνθέτουν το φυσικό πρόβλημα, με τα αντικείμενα του προγράμματος, που επιλύει το πρόβλημα πρέπει να είναι ένα προς ένα. Στην ουσία γίνεται μια προσομοίωση.

Βασικός στόχος του ΑΝΤΚ.Π είναι η βελτίωση της παραγωγικότητας των προγραμματιστών. Η χρήση του ΑΝΤΚ.Π μειώνει την απόσταση μεταξύ της φάσης σχεδίασης μιας εφαρμογής και της υλοποίησής της.

Στον ΑΝΤΚ.Π μοντελοποιούμε κατ'ευθείαν την εφαρμογή, ενώ με τις παραδοσιακές μεθόδους βασιζόμαστε σε διαγράμματα ροής δεδομένων ή σε μαθηματικές λογικές.

Στον ΑΝΤΚ.Π τα προγράμματα λειτουργούν ανταλλάσσοντας "μηνύματα", μεταξύ ενεργών αντικειμένων.

Για παράδειγμα μια εμπορική εφαρμογή ΑΝΤΚ.Π θα μπορούσε να θεωρήσει τους "πελάτες" σαν αντικείμενα που στέλνουν τα μηνύματα "χρέωση" και "πίστωση" προς τα αντικείμενα "λογαριασμοί". Τα αντικείμενα "λογαριασμοί" μπορούν να συνεργάζονται και να ενημερώνουν άλλα αντικείμενα όπως "ταμείο", "υπόλοιπα", "λογαριασμοί προς εξόφληση".

Στον ΑΝΤΚ.Π δημιουργούμε καινούργιους τύπους δεδομένων που τους ονομάζουμε κλάσεις και τους "διδάσκουμε" τον τρόπο με τον οποίο πρέπει να ανταποκρίνονται σε κάποια μηνύματα. Μια κλάση της διδάσκουμε πώς να ενεργεί, όταν δέχεται ένα συγκεκριμένο μήνυμα, δημιουργώντας μια αντίστοιχη "μέθοδο". Ο προγραμματιστής μπορεί να δηλώσει μεταβλητές αυτής της κλάσης. Προς τις μεταβλητές αυτές, που είναι πλέον αντικείμενα, μπορούμε να στείλουμε συγκεκριμένα μηνύματα και η λογική που ισχύει είναι: στέλνω ένα μήνυμα προς ένα αντικείμενο και αφήνω το αντικείμενο να ξεδιαλύνει πώς θα το διαχειριστεί.

Ο ΑΝΤΚ.Π δεν βλέπει τα αντικείμενα σαν παθητικά δεδομένα, αλλά σαν ένα συνδυασμό μιας εσωτερικής κατάστασης του αντικειμένου και των μεθόδων, που την επηρεάζουν (διαχειρίζονται).

Στον ΑΝΤΚ.Π τα δεδομένα δεν "κυκλοφορούν" ελεύθερα στο σύστημα, παρά είναι προστατευμένα από τυχαίες μεταβολές. Στο σύστημα κυκλοφορούν "μηνύματα" και όχι δεδομένα.

C++ διαθέτει ευελιξία και αρθρωτή δόμηση[29] όπως μια πραγματική γλώσσα αντικειμενοστρεφούς προγραμματισμού.

Τελειώνοντας συνοψίζουμε, η C++ εξελίχθηκε από μια σχεδόν άγνωστη γλώσσα προγραμματισμού, που η χρήση της περιοριζόταν σε περιβάλλον UNIX, σε ένα πολύ δημοφιλές εργαλείο ανάπτυξης λογισμικού. Κυριάρχησε σε μικρό χρονικό διάστημα στον χώρο της ανάπτυξης λογισμικού τόσο στο UNIX όσο και στο MS-DOS καθώς και στο OS/2. Πολλοί επαγγελματίες προγραμματιστές προβλέπουν ότι η C++ θα καταλάβει τη θέση της C στη βιομηχανία λογισμικού.

3.3 Τα Χαρακτηριστικά της C++ ως Υψηλή Χαμηλού Επιπέδου Γλώσσας

Η C++ είναι μια "μικρή" (αποδοτική, δυνατή και δημοφιλής), δομημένη γλώσσα προγραμματισμού. Έχει λιγότερες από 46 δεσμευμένες λέξεις. Για να αντισταθμίσει το μικρό λεξιλόγιό της, έχει πάρα πολλούς τελεστές, όπως +, -, && κλπ.. Ο μεγάλος αριθμός τελεστών της C++ μπορεί να οδηγήσει τους προγραμματιστές να γράψουν κρυπτογραφικά προγράμματα με λίγες μόνο γραμμές κώδικα, πράγμα το οποίο επισημαίνουμε για να αποθαρρύνουμε και να προειδοποιήσουμε, "να μην παρεκλείνεται από τις αρχές του συστηματικού προγραμματισμού (βλπ κεφ 2, παρατηρήσεις, σελ 30), είναι σημαντικότερο να κάνετε προγράμματα αναγνώσιμα, από το να εξοικονομείτε μερικά bytes. Σας προτρέπουμε να χρησιμοποιείτε τους τελεστές στην πλήρη τους έκταση και να κρατάτε τα προγράμματα αναγνώσιμα".

Ο μεγάλος αριθμός των τελεστών της C++ (είναι σχεδόν όσες και οι δεσμευμένες λέξεις) απαιτεί σωστή χρήση του Πίνακα προτεραιότητας των τελεστών (βλπ §4.4 "Τελεστές της C++" και §4.4.3 "Σειρά Προτεραιότητας Τελεστών στις Διάφορες Παραστάσεις"). Οι περισσότερες γλώσσες προγραμματισμού έχουν τέσσερα ή πέντε μόνο επίπεδα προτεραιότητας, η C++, αντίθετα, έχει 16, με τα οποία όπως είναι φυσικό πρέπει να

[29] Είναι βασική αρχή του **αρθρωτού προγραμματισμού** (ΑΡΘΡ.Π - modular programming). Ο ΑΡΘΡ.Π μπορεί να οριστεί σαν ένα είδος οργάνωσης του προγράμματος σε μικρά και ανεξάρτητα μέρη που καλούνται **ενότητες** (modules), η συμπεριφορά τους καθορίζεται από ένα σύνολο κανόνων. Ο ΑΡΘΡ.Π μπορεί να χρησιμοποιηθεί για να διαιρεθεί ένα πρόβλημα σε συστήματα, ένα σύστημα σε προγράμματα, ένα πρόγραμμα σε ενότητες. Βασικός στόχος του ΑΡΘΡ.Π είναι η διάσπαση του προβλήματος σε μικρότερα απλούστερα μέρη, τα οποία είναι ανεξάρτητα ή αυτόνομα, με τον τρόπο αυτόν επιτυγχάνεται η μείωση της πολυπλοκότητας του προβλήματος (βλπ κεφ 2, σημείωση, σελ 67).

εξοικειωθούμε.

Ένα άλλο χαρακτηριστικό της C++ είναι ότι δεν έχει προτάσεις εισόδου / εξόδου (εισ/εξ). Η C++ δεν έχει εντολές που εκτελούν εργασίες εισ/εξ. Αυτός είναι ένας από τους κύριους λόγους φορητότητας της C++. Οι προτάσεις εισ/εξ των περισσοτέρων γλωσσών περιορίζουν τη γλώσσα με το συγκεκριμένο υλικό. Για παράδειγμα, η BASIC, έχει σχεδόν 20 προτάσεις/εντολές εισ/εξ, ορισμένες από τις οποίες γράφουν στην οθόνη, στον εκτυπωτή, σε μόντεμ κλπ. Σημειώνουμε ότι, αν γραφεί ένα πρόγραμμα BASIC σε μικροϋπολογιστή, υπάρχουν πολλές πιθανότητες να μην μπορέσει να εκτελεστεί σε κεντρικό υπολογιστή χωρίς βασικές τροποποιήσεις.

Η εισ/εξ στην C++ γίνεται μέσω της χρήσης τελεστών και κλήσεων συναρτήσεων. Με κάθε μεταγλωττιστή της C++ δίνεται μια βιβλιοθήκη πρότυπων συναρτήσεων εισ/εξ. Οι συναρτήσεις εισ/εξ είναι ανεξάρτητες υλικού αφού λειτουργούν σε κάθε συσκευή, σε κάθε Η/Υ που ακολουθεί το πρότυπο της AT&T[∞].

Στο επόμενο κεφάλαιο θα αναπτύξουμε αναλυτικά τα συστατικά της C++, δηλαδή τα κατασκευάσματα που παριστούν τις ενέργειες που περιγράφει η γλώσσα ή τα υπολογιστικά αντικείμενα που χειρίζεται και που εντοπίζονται στα δεδομένα και στην περιγραφή τους, στις δηλώσεις, στους τελεστές, στις εντολές, στους οριοθέτες, στη δομή προγραμμάτων και τέλος στην εισαγωγή (δεδομένων) /εξαγωγή (αποτελεσμάτων) (βλπ § 1.4.2).

3.4 Ασκήσεις (Ερωτήσεις)

1. Τι γνωρίζεται για τον διαδικασιακό προγραμματισμό.
2. Τι γνωρίζεται για τον αντικειμενοστρεφή προγραμματισμό
3. Τι γνωρίζεται για τον αρθρωτό προγραμματισμό.
4. Ποιό είναι το όνομα μιας από τις γλώσσες προγραμματισμού από τις οποίες προήλθε η C;

[∞] Η C++ έχει οριστεί από την AT&T έστε να υπάρχει συμβατότητα ανάμεσα στις διάφορες εκδόσεις της.

5. Ποιά δεκαετία αναπτύχθηκε η C++;

6. Αληθές ή ψευδές; Η C++ είναι γνωστή σαν "καλλίτερη C".

7. Ποιές είναι οι παραπάνω δυνατότητες που διαθέτει η C++ σε σχέση με την C;

8. Αναφέρατε και άλλα χαρακτηριστικά της C++.

9. Είναι αληθές ή ψευδές ότι: Η C++ είναι πολύ μεγάλη και δεν χωρά σε πολλούς μικροϋπολογιστές;

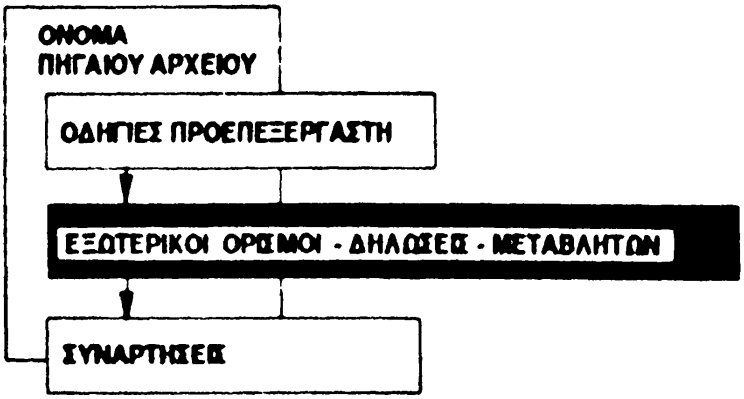
10. Τι είναι σημαντικότερο αναγνώσιμα προγράμματα C++ ή κρυπτογραφημένα σύμφωνα με τις δυνατότητες τις C++ και εξοικονόμηση μερικών bytes και γιατί;

ΚΕΦΑΛΑΙΟ 4

ΓΡΑΦΟΝΤΑΣ ΠΡΟΓΡΑΜΜΑ ΣΤΗΝ C++

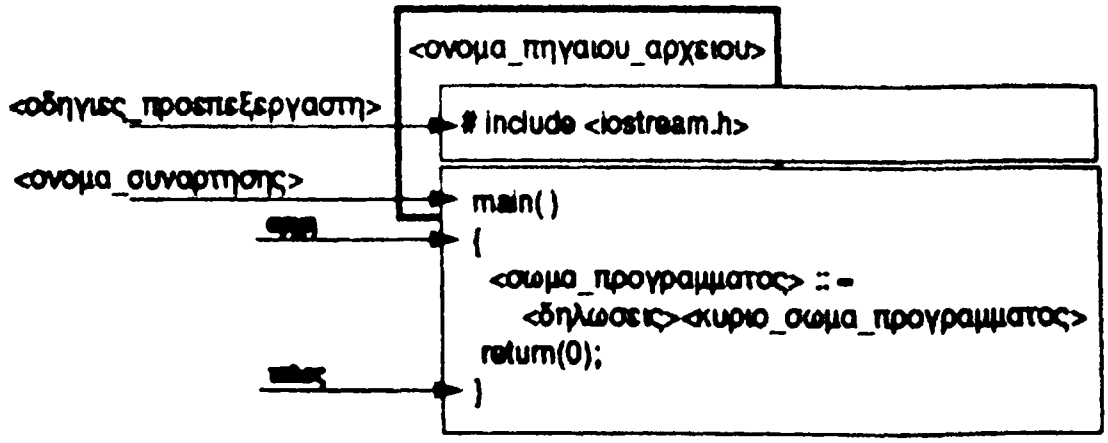
4.1 Γενική Μορφή Προγράμματος στην C++

Η απλούστερη μορφή ενός προγράμματος C++, το οποίο βρίσκεται ολόκληρο σε ένα και μοναδικό πηγαίο αρχείο αποτελείται από τρείς ενότητες: **1. οδηγίες προεπεξεργαστή, 2. εξωτερικοί ορισμοί (δηλώσεις) μεταβλητών** [1] και **3. συναρτήσεις** (βλπ Σχήμα 4.1.1).



ΣΧΗΜΑ 4.1.1: Γενική μορφή προγράμματος C++ ενός μοναδικού πηγαίου αρχείου.

Παραθέτουμε για παράδειγμα τον σκελετό του τύπου των προγραμμάτων του κεφ 2.



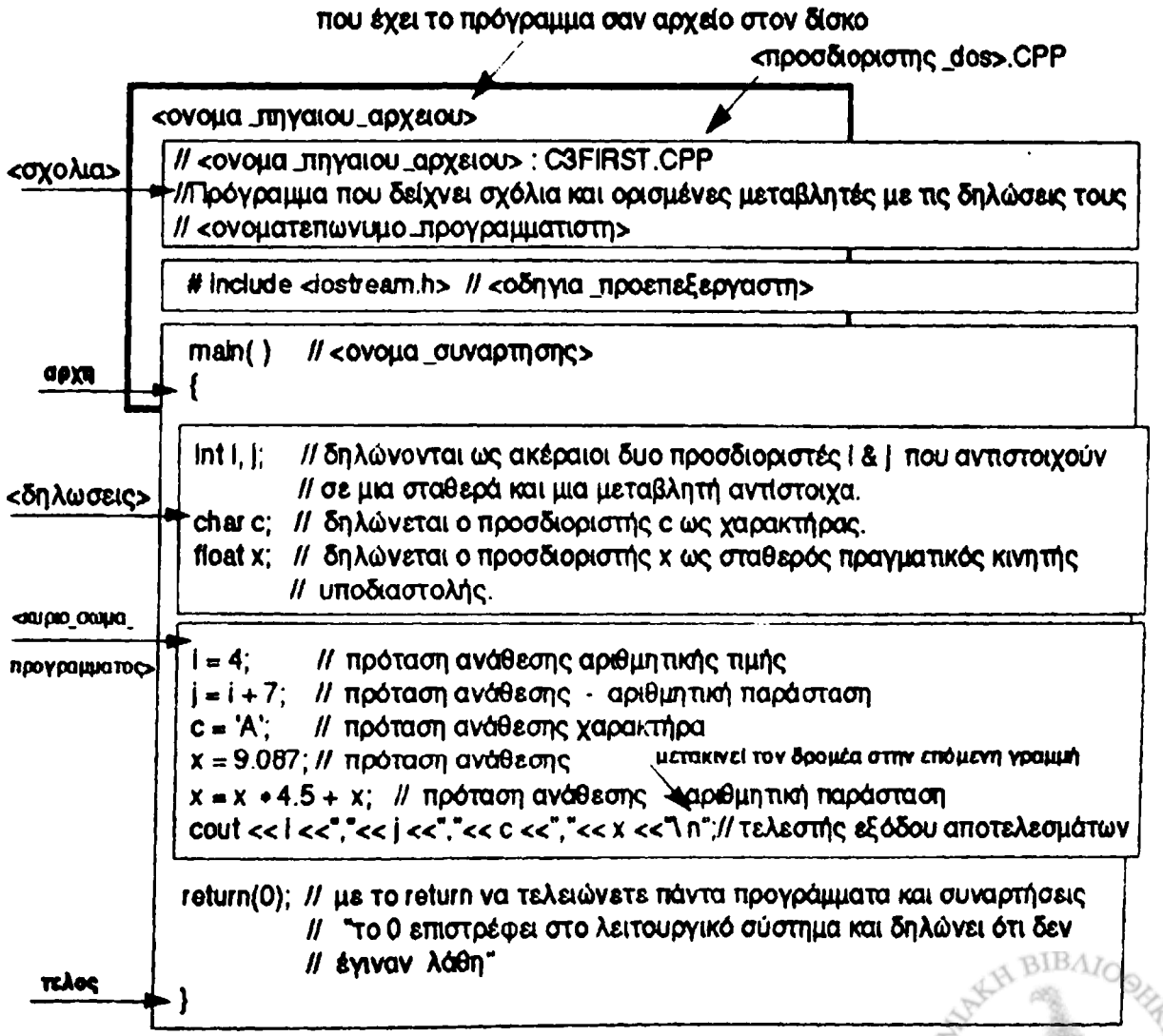
ΣΧΗΜΑ 4.1.2: Σκελετός απλού προγράμματος C++.

[1] Η ενότητα αυτή εμφανίζεται σε ορισμένες περιπτώσεις (περίπτωση ύπαρξης καθολικών μεταβλητών). περισσότερες λεπτομέρειες βλπ στην § 4.10

Για να κάνουμε περισσότερο σαφή τα προηγούμενα σχήματα αλλά και καλλίτερη την σύνδεσή τους με τα παραδείγματα του κεφ 2, παραθέτουμε το παρακάτω παράδειγμα.

ΠΑΡΑΔΕΙΓΜΑ 4.1.1

Δίδεται ένα δείγμα προγράμματος σε C++, το οποίο αποτελεί μια γενική μορφή προγραμματισμού σε C++. Στο παράδειγμα αυτό διακρίνονται: (α) τα μέρη του προγράμματος, όπως αυτά παρουσιάζονται στο Σχήμα 4.1.2 και (β) κάποια από τα πολύ βασικά συστατικά της γλώσσας, όπως: 1. τα σχόλια και ο τρόπος που εμφανίζονται, 3.οι δηλώσεις, με τρείς τύπους δεδομένων, 4. οι προτάσεις ανάθεσης, εκ των οποίων η δεύτερη το δεξί μέλος της είναι αριθμητική παράσταση και τέλος 5. ο τελεστής εξόδου αποτελεσμάτων.



Περαιτέρω εξήγηση του παραπάνω προγράμματος- δείγματος :

1. **Το <ονομα_πηγαιου _αρχειου>** : Το πηγαίο πρόγραμμα της C++ είναι καταχωρημένο ως αρχείο με το όνομα C3FIRST.CPP. Το όνομα του αρχείου είναι προσδιοριστής του MS-DOS[32] και ακολουθεί τους γνωστούς κανόνες προσδιοριστών του[33] με την επέκταση .CPP (<προσδιοριστης_dos>.CPP). Σαν προσδιοριστής του DOS μπορεί να γραφεί με κεφαλαία ή πεζά γράμματα του λατινικού αλφαβήτου.

2. **Τα <σχολια> στην C++** : Το ευανάγνωστο ενός προγράμματος βελτιώνεται με την πρόσθεση σχολίων στο κείμενο του προγράμματος. Τα σχόλια είναι μηνύματα που εξηγούν τι κάνει το πρόγραμμα, στο σημείο του προγράμματος όπου έχουν εισαχθεί. Τα σχόλια αρχίζουν πάντοτε με το σύνθετο σύμβολο // και τελειώνουν στο τέλος της γραμμής. Τα σχόλια μπορούν να γράφονται με πεζά ή κεφαλαία γράμματα, αφού αγνοούνται από τον μεταγλωττιστή της C++ (βλπ παρατηρήσεις 4.1.1 παρακάτω). Τα σχόλια μπορούν να βρίσκονται στην ίδια γραμμή με προτάσεις της γλώσσας. Σε ένα πρόγραμμα τα σχόλια επιβάλλονται τόσο όσο πρέπει να αποφεύγονται τα περιττά σχόλια.

Τα σχόλια που περιέχονται στις τρείς πρώτες γραμμές του προγράμματος δεν είναι τα μοναδικά, υπάρχουν και άλλα παρακάτω στο πρόγραμμα. Τα σχόλια των τριών πρώτων γραμμών:

```
// <ονομα_πηγαιου_αρχειου> : C3FIRST.CPP
// Πρόγραμμα που δείχνει σχόλια & μεταβλητές με τις δηλώσεις τους
// <ονοματεπωνυμο_προγραμματιστη>
```

[32] Το MS-DOS (το λειτουργικό συστημα δίσκου της Microsoft) είναι ένα σύστημα που επιτρέπει στα προγράμματα της C++ να αλληλοεπιδρούν με το υλικό. Το MS-DOS (που συνήθως καλείται και DOS), είναι πάντα φορτωμένο στην RAM, όταν ανάβετε τον Η/Υ σας. Το DOS ελέγχει και άλλα προγράμματα εκτός των δίσκων. Χρησιμοποιείται ώστε να επικοινωνούν τα προγράμματά σας με όλο το υλικό του Η/Υ, περιλαμβανομένων της οθόνης, του πληκτρολογίου και του εκτυπωτή.

"Δεν χρειάζεται να είστε ειδικοί στο DOS, ή ακόμη να ξέρετε περισσότερες από μερικές απλές εντολές του για να είστε καλοί με τα PC. Το DOS όμως, κάνει ορισμένα πράγματα τα οποία δεν μπορεί να κάνει η C++, όπως για παράδειγμα η μορφοποίηση δίσκων, αντιγραφή αρχείων σε δίσκους κλπ. Είναι αλήθεια ότι καθώς θα εξοικειώνεστε με τον Η/Υ σας, θα βρίσκεστε στην ανάγκη να αναβαθμίζετε τις γνώσεις για το DOS. Για την καλλίτερη, τότε, εκμάθηση και χρήση του, διαβάστε ένα βιβλίο (ή να συμβουλεύεστε έναν μικρό οδηγό [24]) Γρήγορης Εκμάθησης του MS-DOS."

[33] Οι κανόνες για τη δημιουργία ονομάτων αρχείων ή καταλόγων του DOS είναι.

1. Τα ονόματα (προσδιοριστές) μπορούν να έχουν μήκος από έναν έως οκτώ χαρακτήρες.
2. Τα ονόματα (προσδιοριστές) μπορούν να έχουν μια επέκταση μέχρι και τρείς χαρακτήρες. Η επέκταση ακολουθεί πάντα το όνομα και διαχωρίζεται από αυτό με μια τελεία. Οι επεκτάσεις χρησιμοποιούνται για να προσδιορίσουν διαφορετικούς τύπους αρχείων.
3. Τα ονόματα (προσδιοριστές) μπορούν να περιέχουν οποιοδήποτε συνδυασμό γραμμάτων ή αριθμών ή των ακολούθων ειδικών χαρακτήρων - ' ! @ # - $ % ^ & ( ) - { } ' με τον περιορισμό ότι ο πρώτος χαρακτήρας θα είναι πάντα αριθμητικός.
4. Τα ονόματα (προσδιοριστές) δεν μπορούν να περιέχουν κανά διαστήματα.

περιέχουν το όνομα του αρχείου, εξηγούν τον σκοπό του προγράμματος και περιλαμβάνουν το ονοματεπώνυμο του προγραμματιστή, ίσως αν χρειαστεί και άλλα στοιχεία του.

3. **Η <οδηγια_προεπεξεργαστη>** : Η πρόταση **#include <iostream.h>** καλείται οδηγία προεπεξεργαστή, δεν είναι πρόταση της C++, αλλά μια οδηγία που δίνει εντολή στον μεταγλωττιστή της C++ να φορτώση ένα αρχείο από τον δίσκο (ανάμεσα) στο πρόγραμμα. Σκοπό δε έχει τη διασφάλιση της σωστής λειτουργίας της εξόδου που παράγεται με τον cout.

4. **Οι <αγκυλες> και η <συναρτηση> main( )** : Όλα τα προγράμματα της C++ πρέπει να περιέχουν τις δύο γραμμές:

```
main( )
{
```

Οι προτάσεις που ακολουθούν την main( ) εκτελούνται πρώτες. Η ενότητα ενός προγράμματος C++ που αρχίζει με την **main( )** και ακολουθείται από **{** λέγεται **κύρια σνάρτηση.** Η συνάρτηση αυτή δεν χρειάζεται να είναι η πρώτη, συνήθως όμως είναι. Σε κάθε **αριστερή αγγύλη {** , που ισοδυναμεί με **αρχη**, αντιστοιχεί και μια **δεξιά αγγύλη }**, που ισοδυναμεί με **τελος.** Ένα πρόγραμμα C++ είναι στην πραγματικότητα μια συλλογή συναρτήσεων (μικρών ενοτήτων κώδικα). Η συνάρτηση main( ) είναι απαραίτητη αλλά και η πρώτη συνάρτηση που εκτελείται.

Στο παράδειγμά μας, όλο το πρόγραμμα είναι main( ), κύρια συνάρτηση. Οι προτάσεις του προγράμματος βρίσκονται μεταξύ του ζεύγους των αγγύλων { } και αποτελούν το σώμα του προγράμματος. Όλες οι προτάσεις (εκτελέσιμες) του προγράμματός μας πρέπει και τελειώνουν με ερωτηματικό ; για να ξέρει η C++ ότι η πρόταση τελείωσε. Σημειώνουμε ότι οι γραμμές της main( ) και των αγγίστρων δεν τελειώνουν με ερωτηματικό, για τον απλό λόγο ότι οι γραμμές αυτές ορίζουν την αρχή και το τέλος της συνάρτησης και δεν εκτελούνται. Επίσης στο τέλος των σχολίων δεν βάζουμε ποτέ ερωτηματικό, επειδή, όπως προαναφέραμε, τα σχόλια αγνοούνται από τον μεταγλωττιστή της C++.

5. **Οι <δηλωσεις>** :

```
int i, j;   // δηλώνονται ως ακέραιοι δυο προσδιοριστές i & j που αντιστοιχούν
            // σε μια σταθερά και μια μεταβλητή αντίστοιχα.
```

char c; // δηλώνεται ο προσδιοριστής c ως χαρακτήρας.

float x; // δηλώνεται ο προσδιοριστής x ως σταθερός πραγματικός κινητής
// υποδιαστολής (δηλαδή με δεκαδικό μέρος).

Οι δηλώσεις των μεταβλητών περιγράφουν τα δεδομένα που χρησιμοποιούνται στο σώμα του προγράμματος. Οι δηλώσεις των μεταβλητών περιγράφουν τον τρόπο αποθήκευσης των δεδομένων του προγράμματος ανάλογα με τον τύπο τους.

Είναι γνωστό ότι ένα πρόγραμμα επεξεργάζεται δεδομένα και παράγει αποτελέσματα, εκτελώντας τις προτάσεις (εντολές) του εκάστοτε υπολογισμού. Τα δεδομένα παρίστανται με προσδιοριστές και είναι μεταβλητές και σταθερές.

6. Το **<κυριο_σωμα_προγραμματος>** :

```
i = 4;        // πρόταση ανάθεσης αριθμητικής τιμής
j = i + 7;    // πρόταση ανάθεσης - αριθμητική παράσταση
c = 'A';      // πρόταση ανάθεσης χαρακτήρα (προσοχή στα εισαγωγικά)
x = 9.067; // πρόταση ανάθεσης
x = x + 4.5 + x;  // πρόταση ανάθεσης - αριθμητική παράσταση
cout << i <<",,"<< j <<",,"<< c <<",,"<< x <<"\ n"; // τελεστής εξόδου αποτ/σμάτων
```

Όταν το πρόγραμμα φθάσει στην τελευταία γραμμή, εκτυπώνει τα περιεχόμενα των τεσσάρων μεταβλητών στην οθόνη. Η έξοδος από την γραμμή αυτή είναι: 4, 11, A, 49.978499. Η cout δεν είναι πρόταση της C++, η C++ δεν έχει προτάσεις εισ/εξ. Η cout είναι ένας τελεστής, που η περιγραφή του για τον μεταγλωττιστή υπάρχει στο αρχείο #include με όνομα iostream.h και στέλνει έξοδο στην οθόνη. Το "\ n" ισοδυναμεί με το **endl** και σημαίνει ότι μετακινεί τον δρομέα στην επόμενη γραμμή.

7. Η **<προταση> return** :

```
return(0); // με το return να τελειώνετε πάντα προγράμματα και συναρτήσεις
           // "το 0 επιστρέφει στο λειτουργικό σύστημα και δηλώνει ότι δεν
           // έγιναν λάθη"
```

Η εντολή return απλώς δίνει το μήνυμα ότι η συνάρτηση αυτή (main( )) τελείωσε. Η C++ επαναφέρει τον έλεγχο στο σημείο που βρισκόταν πριν αρχίσει η συνάρτηση. Στην περίπτωσή μας, υπάρχει μια μόνο συνάρτηση, έτσι ο έλεγχος απονέρχεται στο DOS ή στο περιβάλλον επεξεργασίας της C++. Η C++ χρειάζεται μια τιμή επιστροφής, για τον λόγο αυτόν επιστρέφουμε το 0 (return(0)) στο λειτουργικό σύστημα. Αν δεν χρησιμοποιούνται μεταβλητές επιστροφής στο λειτουργικό σύστημα, δεν υπάρχει ιδιαίτερη χρησιμότητα αυτής της τιμής. Στην πραγματικότητα, πολλές προτάσεις return είναι προαιρετικές. Στις

περιπτώσεις αυτές η C++ ξέρει πότε έφθασε το τέλος του προγράμματος και χωρίς την πρόταση αυτή. Είναι όμως προγραμματιστικά σωστό να γράφεται μια return στο τέλος κάθε συνάρτησης, συμπεριλαμβανομένης και της main( ). Σε ορισμένες συναρτήσεις, όταν επιστρέφονται τιμές, απαιτείται η γραφή της return, αν λοιπόν συνηθίσουμε να την χρησιμοποιούμε (την return), δεν θα την ξεχνάμε όταν χρειάζεται πραγματικά.

Παρατηρήσεις 4.1.1:

1. Απαιτούνται **ευανάγνωστα προγράμματα**, όρο πιό ευανάγνωστο είναι το πρόγραμμα, τόσο πιό γρήγορα εντοπίζονται όσα χρειάζονται αλλαγή. Τα προγράμματα που είναι απλά, ευανάγνωστα, γεμάτα ελεύθερο χώρο (με διαχωριστικές γραμμές και διαστήματα) και χωρίς δυσνόητα και δυσανάγνωστα τεχνάσματα είναι τα πλέον ελκυστικά.

"Χρησιμοποιείται πολύ ελεύθερο χώρο. Παρατηρήστε ότι οι πρώτες γραμμές του C3FIRST.CPP αρχίζουν στην πρώτη στήλη, αλλά το σώμα του προγράμματος είναι πιό μέσα. Έτσι ο προγραμματιστής μπορεί να κοιτάξει κατευθείαν αυτό που τον ενδιαφέρει. Όταν γράφετε προγράμματα που περιέχουν αρκετές ενότητες, η χρήση ελεύθερου χώρου βοηθά το μάτι να ακολουθήσει και να αναγνωρίσει την επόμενη ενότητα με εσοχή.

Αν εργάζεστε σαν προγραμματιστής ίσως χρειαστεί να αλλάξετε τον κώδικα κάποιου άλλου ή άλλοι να αλλάξουν τον δικό σας. Στα τμήματα προγραμματισμού εταιριών λένε ότι οι προγραμματιστές που θα μείνουν γράφουν ευανάγνωστα προγράμματα, αλλά και ψάχνουν να βρούν προγραμματιστές που γράφουν προγράμματα για το μέλλον."

2. **Βάζεται σχόλια όπως προχωρείτε**, τότε είστε περισσότερο γνώστες της λογικής του προγράμματος την ώρα που το πληκτρολογείτε. Αν αναβάλλετε την εισαγωγή των σχολίων μέχρι το τέλος του προγράμματος, το πιθανότερο είναι ότι δεν θα τα προσθέσετε ποτέ. Τα σχόλια βοηθούν ακόμα, στο να ξαναγυρίσετε και να δείτε τι κάνατε, όταν ήδη γράφετε τις επόμενες ενότητες του προγράμματος χωρίς να αποκωδικοποιήσετε τον προηγούμενο κώδικά σας.

3. **Κεφαλαία και πεζά γράμματα** στην C++. **Χρησιμοποιείται κεφαλαία γράμματα στην C++ με μεγάλη προσοχή.** Τα κεφαλαία και πεζά γράμματα είναι σημαντικότερα στην C++ από ότι στις άλλες γλώσσες.Το C3FIRST.CPP, π.χ., είναι γραμμένο με κεφαλαία γράμματα, ενώ όλη η γλώσσα C++ είναι γραμμένη με πεζά γράμματα. Για παράδειγμα οι δεσμευμένες λέξεις int, char, return, είναι γραμμένες με πεζά γράμματα, αν πληκτρολογηθούν με κεφαλαία γράμματα ο μεταγλωττιστής της C++ θα παράγει λάθη και δεν θα μεταγλωττίσει το πρόγραμμα. Καμμία πρόταση της C++ δεν γράφεται με κεφαλαία

γράμματα, εκτός και αν χρησιμοποιηθούν για ορισμένες λέξεις και μηνύματα που στέλνονται στην οθόνη, στον εκτυπωτή ή σε αρχεία του δίσκου.

4. Στον προγραμματισμό ο Η/Υ διαβάζει κάθε γραμμή του πηγαίου προγράμματος, ξεκινώντας από την πρώτη και φθάνοντας μέχρι την τελευταία, έως ότου δηλαδή ολοκληρώσει όλες τις προτάσεις. Κατόπιν το πηγαίο πρόγραμμα[34] μεταγλωττίζεται και στη

---

[34] **ΕΠΕΞΕΡΓΑΣΤΗΣ** : Πληκτρολογείτε τον πηγαίο κώδικα στην μνήμη του Η/Υ, χρησιμοποιώντας τον επεξεργαστή προγράμματος. Αφού πληκτρολογήσετε τον πηγαίο κώδικα C++, πρέπει να τον αποθηκεύσετε σε αρχείο δίσκου πριν μεταγλωττίσετε και εκτελέσετε το πρόγραμμα. Οι περισσότεροι μεταγλωττιστές της C++ περιμένουν τα πηγαία προγράμματα να είναι αποθηκευμένα σε αρχεία με επεκτάσεις .CPP.

Πολλοί μεταγλωττιστές της C++ περιλαμβάνουν ενσωματωμένο επεξεργαστή. Δύο από τους δημοφιλέστερους μεταγλωττιστές που ακολουθούν το πρότυπο AT&TC++ 2.1, είναι οι Borland και Microsoft C/C++ 7.7. Τα δυο αυτά προγράμματα εκτελούνται σε πλήρως ολοκληρωμένα περιβάλλοντα, οπότε ο χρήστης δεν χρειάζεται να βρει ένα ξεχωριστό επεξεργαστή ή να μάθει πολλές εντολές του μεταγλωττιστή. Δηλαδή με τους μεταγλωττιστές αυτούς, απλώς πληκτρολογεί το πρόγραμμα στον επεξεργαστή και μετά με ένα βήμα επιλέγει τη σωστή επιλογή από το παράθυρο που μεταγλωττίζει και εκτελεί το πρόγραμμα.

Στην αντίθετη περίπτωση όμως, που δεν υπάρχει ολοκληρωμένο περιβάλλον, ο χρήστης θα πρέπει να εκκινήσει έναν επεξεργαστή, να πληκτρολογήσει το πρόγραμμα, να αποθηκεύσει το πρόγραμμα σε δίσκο, να βγει από τον επεξεργαστή , να εκτελέσει το πρόγραμμα μεταγλώττισης και μόνο μετά να εκτελέσει το μεταγλωττισμένο πρόγραμμα από το λειτουργικό σύστημα.

Στην περίπτωση λοιπόν, που δεν υπάρχει ολοκληρωμένο περιβάλλον σαν αυτά της Borland και της Microsoft C/C++ 7.7, πρέπει να βρεθεί ένας επεξεργαστής κειμένου. Οι επεξεργαστές κειμένου μπορούν να δουλέψουν σαν επεξεργαστές, αλλά πρέπει ο χρήστης πρώτα να μάθει πως να αποθηκεύει και να φορτώνει αρχεία σε μορφή κειμένου ASCII. Μερικές φορές είναι ευκολότερο να χρησιμοποιηθεί ένας επεξεργαστής από το να χρησιμοποιηθεί ένας επεξεργαστής κειμένου σαν απλός επεξεργαστής.

Στα PC, το DOS 5 παράγεται με έναν απλό επεξεργαστή πλήρους οθόνης, τον EDIT. Αυτός προσφέρει εντολές σε παράθυρα και πλήρεις δυνατότητες ελέγχου δρομέα. Ο EDIT είναι ένα απλό, ευχρηστο πρόγραμμα και είναι ό,τι πρέπει για επεξεργαστής για αρχάριους.

Ένας άλλος επεξεργαστής είναι ο EDLIN, διατίθεται σε προηγούμενες εκδόσεις του DOS. Ο EDLIN είναι ένας επεξεργαστής γραμμής, δεν παρέχει έλεγχο πλήρους οθόνης και απαιτεί από τους χρήστες να μάθουν κάποιες κρυπτογραφικές εντολές. Το πλεονέκτημα του EDLIN είναι ότι υπάρχει σε όλα τα PC με DOS πριν από το 5.

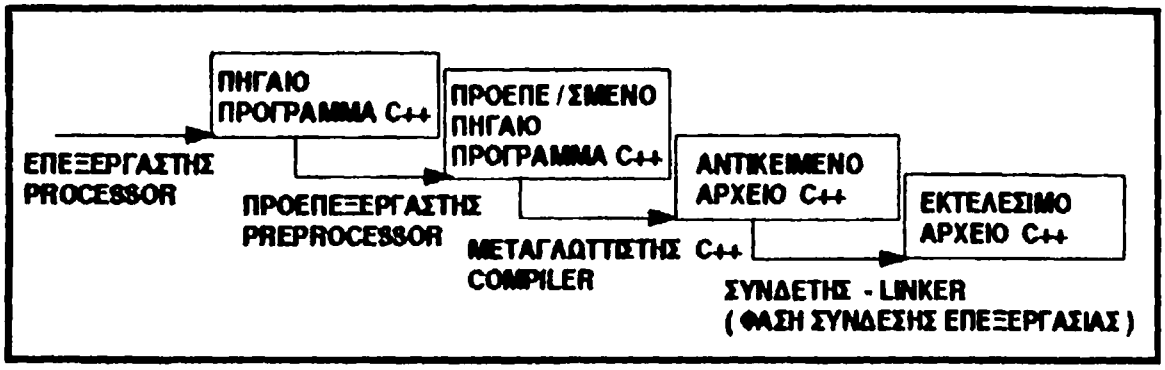
Τα περισσότερα συστήματα UNIX περιέχουν τον επεξεργαστή vi. Αυτός είναι για τον UNIX ό,τι ο EDLIN για το DOS και υπάρχει σε όλα τα συστήματα UNIX στον κόσμο.

Οι χρήστες κεντρικών υπολογιστών έχουν στη διάθεσή τους άλλους επεξεργαστές, π.χ. τον ISPF. Σε οποιαδήποτε άλλη περίπτωση μινιυπολογιστή ή κεντρικού υπολογιστή, πρέπει ο χρήστης να ενημερωθεί σε ποιόν επεξεργαστή έχει πρόσβαση.

ΜΕΤΑΓΛΩΤΤΙΣΤΗΣ Αφού πληκτρολογηθεί και διορθωθεί ο πηγαίος κώδικας του προγράμματος, πρέπει το πρόγραμμα να μεταγλωττιστεί. Η διαδικασία που ακολουθείτε εξαρτάται από την έκδοση της C++και τον Η/Υ που χρησιμοποιείται. Οι χρήστες της Borland και της Microsoft C/C++ , αρκεί να πιέσουν το Alt-R για να μεταγλωττίσουν και να εκτελέσουν τα προγράμματά τους. Όταν μεταγλωττίζονται προγράμματα στα περισσότερα PC, ο μεταγλωττιστής παράγει τελικά ένα εκτελέσιμο αρχείο με όνομα ίδιο με του αρχείου του πηγαίου κώδικα, αλλά με την επέκταση .EXE. Το μεταγλωττισμένο αυτό αρχείο μπορεί να εκτελείται από την προτροπή του DOS πληκτρολογώντας σκέτο το όνομά του.

Οι χρήστες του UNIX ίσως χρειαστεί να χρησιμοποιήσουν τον μεταγλωττιστή cfront. Οι περισσότεροι μεταγλωττιστές cfront συνήθως μετατρέπουν τον κώδικα C++ σε κώδικα C. Κατόπιν ο κώδικας C μεταγλωττίζεται από τον μεταγλωττιστή C του συστήματος. Αυτός παράγει ένα εκτελέσιμο αρχείο, το όνομα του οποίου είναι εξ ορισμού A.OUT. Μετά μπορεί να εκτελεστεί το αρχείο A.OUT από την προτροπή του UNIX. Οι χρήστες κεντρικών Η/Υ έχουν συνήθως προτυπες διαδικασίες για μεταγλώττιση πηγαίων προγραμμάτων C++ και αποθήκευση αποτελεσμάτων σε ειδικά σημεία.

συνέχεια αρχίζει η εκτέλεση του αντικείμενου προγράμματος (βλπ Σχήμα 4.1.3 - πρβλ υποσημείωση 3 §1.1 και γλώσσες ανωτέρου επιπέδου §1.4.1).



ΣΧΗΜΑ 4.1.3: Στάδια μεταγλώττισης πηγαίου κώδικα C++ για την παραγωγή εκτελέσιμου προγράμματος

Αντίθετα από άλλες γλώσσες προγραμματισμού, το πρόγραμμα C++ πρέπει να δρομολογηθεί μέσω ενός **προεπεξεργαστή** (preprocessor) πριν μεταγλωττιστεί (βλπ Σχήμα 4.1.3). Ο προεπεξεργαστής διαβάζει τις "οδηγίες προεπεξεργαστή" που εισάγονται στο πρόγραμμα για έλεγχο της μεταγλώττισης του προγράμματος. Ο μεταγλωττιστής C++ εκτελεί αυτόματα την προεπεξεργασία, οπότε δεν χρειάζεται η εκμάθηση άλλων εντολών.

Το πρόγραμμα σε C++πρέπει να περάσει και από ένα ακόμα στάδιο μετά την μεταγλώττιση και πριν την εκτέλεση. Αυτό καλείται **σύνδεση** (link) ή **φάση σύνδεσης επεξεργασίας** (βλπ Σχήμα 4.1.3). Όταν συνδέεται το πρόγραμμα C++, τότε ένα πρόγραμμα με το όνομα **συνδέτης** (linker) παρέχει τις απαιτούμενες πληροφορίες χρόνου εκτέλεσης στο μεταγλωττισμένο πρόγραμμα. Ο χρήστης μπορεί επίσης να συνδέσει αρκετά μεταγλωττισμένα προγράμματα σε ένα εκτελέσιμο, αν και τις περισσότερες φορές ο μεταγλωττιστής εκκινεί τη φάση σύνδεσης επεξεργασίας, το τελευταίο ισχύει κυρίως για μεταγλωττιστές σαν αυτούς της Borland και της Microsoft C/C++.

4.1.1 Ασκήσεις (Ερωτήσεις για Επανάληψη και Εμπέδωση)

1. Περιγράψατε τη γενική μορφή προγράμματος C++ με τη BNF και με τα Συντακτικά Διαγράμματα.

2. Ποιοί είναι οι κανόνες για τη δημιουργία ονομάτων ή καταλόγων στο DOS;

3. Τι γνωρίζετε για το όνομα του πηγαίου αρχείου (<ονομα_πηγαιου_αρχειου>) που είναι καταχωρημένο ένα πηγαίο πρόγραμμα της C++;

4. Τι προηγείται κάθε σχολίου σε ένα πρόγραμμα C++;

5. Πώς βελτιώνεται με το ευανάγνωστο ενός προγράμματος, τι επιτυγχάνεται γενικά με ευανάγνωστα προγράμματα;

6. Τι γνωρίζετε για την οδηγία του προεπεξεργαστή (<οδηγια_προεπεξεργαστη>);

7. Τι γνωρίζετε για τις αγγύλες { } και τη συνάρτηση **main( )** στα προγράμματα C++;

8. Τι γνωρίζετε για τις δηλώσεις γενικά στα προγράμματα και ειδικά στα προγράμματα της C++;

9. Ποιός είναι ο τελεστής που γράφει έξοδο στην οθόνη;

10. Το "\n" και το **endl** είναι ισοδύναμα; Τι σημαίνουν;

11. Τι γνωρίζετε για την πρόταση **return** των προγραμμάτων της C++;

12. Χρειάζονται τα σχόλια σε πρόγραμμα και γιατί;

13. Τι γνωρίζετε για την αντιμετώπιση των κεφαλαίων και πεζών γραμμάτων από την C++;

14. Ποιά είναι τα στάδια μεταγλώττισης πηγαίου κώδικα C++ για τη παραγωγή εκτελέσιμου προγράμματος C++;

15. Τι γνωρίζετε για τον πεξεργαστή και τι για τον προεπεξεργαστή;

16. Τι είναι η σύνδεση, η φάση σύνδεσης επεξεργασίας και τι είναι ο συνδέτης;

4.2 Το Λεξιλόγιο της C++

Η γλώσσα προγραμματισμού C++, όπως συμβαίνει με όλες τις τυπικές γλώσσες (βλπ κεφ1) έχει το αλφάβητό της, το λεξιλόγιό της, τους κανόνες που παράγει ή ελέγχει τους προσδιοριστές και τις προτάσεις της. Οι κανόνες που παράγουν αλλά και ελέγχουν τους προσδιοριστές και τις προτάσεις μιας τυπικής γλώσσας είναι οι κανόνες παραγωγής της, είναι οι κανόνες σύνταξης της και συγχρόνως και σημασιολογίας της, αφού η σύνταξη και η σημασιολογία στις προτάσεις των τυπικών γλωσσών ταυτίζεται (βλπ §1.1 κεφ1).

Το **αλφάβητο της C++** αποτελείται από τα γράμματα του αγγλικού αλφαβήτου (a-z), τα ψηφία του δεκαδικού συστήματος αρίθμησης (0-9) και ένα σύνολο ειδικών χαρακτήρων που χρησιμοποιούνται ως τελεστές, οριοθέτες κλπ. Το αλφάβητο χρησιμοποιείται για να εφραστούν ή να οριστούν τα διάφορα είδη λέξεων (αλυσίδες χαρακτήρων) που μπορούν να υπάρχουν σ'ένα πρόγραμμα C++. Τα είδη αυτά των λέξεων είναι:

1. Οι **δεσμευμένες λέξεις** (reserved Keywords). Είναι ένα σύνολο λέξεων που αποτελούν το λεξικό της γλώσσας. Οι λέξεις αυτές δεν μπορούν να χρησιμοποιηθούν για άλλο σκοπό εκτός από αυτόν που προβλέφθηκε από τους σχεδιαστές της γλώσσας.

Οι δεσμευμένες λέξεις της C++ χωρίζονται σε δύο κατηγορίες, σε αυτές που προέρχονται από την C (**auto, break, case, char, const, continue , default , do, double , else, entry, enum, extern, float, for, goto, if, int, long, register, return, short, signed, sizeof, static, struct, switch, typedef, union, unsigned , void, volatile, while**) και σε αυτές που εισάγει αυτή καθεαυτή η C++ (**asm, catch, class, delete , friend, inline , new, operator, overload, private, protected , public , template , this, virtual**).

2. Οι **προσδιοριστές** (identifiers). Είναι τα ονόματα των μεταβλητών (variables) που χρησιμοποιούνται σ'ένα πρόγραμμα.

3. Οι **σταθερές τιμές** (constands / const). Είναι οι τιμές (σταθερές) που αναφέρονται κυρίως στις αριθμητικές π.χ. 100, 5.3, κλπ. Οι μη αριθμητικές τιμές δηλαδή οι αλυσίδες χαρακτήρων συμπεριλαμβάνονται σε άλλη κατηγορία, επειδή η διαφορά στον τρόπο έκφρασης και αποθήκευσης είναι ουσιαστική.

4. Οι **αλυσίδες χαρακτήρων** (strings ή characters). Σύνολα χαρακτήρων (αλφαριθμητικών-ειδικών χαρακτήρων) ή λέξεων.

5. Οι **τελεστές** (operators). Είναι ειδικοί χαρακτήρες που δηλώνουν πράξεις. Η C++ διατηρεί όλο το πλούσιο σύνολο τελεστών που διαθέτει η C και επιπλέον επιτρέπει την επικάλυψη των τελεστών αυτών. Εκτός από τους τελεστές της C, η C++ διαθέτει και τους εξής: :: **this** & **new delete** → *. Οι τελεστές είναι: ( ) , [ ] , → , ::, . , ! , ~ , + ,

++ , -- , & , * , **sizeof** , **new** , **delete** , .* , ->* , * , / , % , + , - , << , >> , < , <= , > , >= , == , != , & , ^ , | , && , || , ?: , = , *= , /= , %= , += , -= , &= , ^= , |= , <<= , >>= , , .

6. Τα **σημεία στίξης και οριοθέτησης** (punctuators & separators). Είναι χαρακτήρες που χρησιμοποιούνται για να διαχωριστούν οι προσδιοριστές, οι δεσμευμένες λέξεις και οι σταθερές. Κάποιοι από τους χαρακτήρες αυτούς δεν φαίνονται, αυτοί είναι οι **space**, **tab**, **carriage return** (CR) και **line feed** (LF). Αντίθετα οι χαρκτήρες **# ( ) [ ] { } , : ;** χρησιμοποιούνται ως σύμβολα στίξης. Σημειώνουμε ότι τα σύμβολα **( ) [ ] { }** χρησιμοποιούνται πάντοτε κατά ζεύγη.

4.2.1 Ασκήσεις (Ερωτήσεις για Επανάληψη και Εμπέδωση)

1. Αλφάβητο, λεξιλόγιο και κανόνες σε τι χρησιμεύουν σε μια γλώσσα προγραμματισμού;
2. Το αλφάβητο της C++ από τι αποτελείται;
3. Ποιά είναι τα είδη των λέξεων που αποτελούν το λεξιλόγιο της C++;

4.3 Προσδιοριστές - Στοιχειώδεις Τύποι Δεδομένων - Δηλώσεις

Η κατανόηση της επεξεργασίας δεδομένων στην C++, όπως και σε κάθε γλώσσα προγραμματισμού, προυποθέτει τη γνώση και την εξοικείωση με τον τρόπο δημιουργίας, αποθήκευσης και χειρισμού των δεδομένων της γλώσσας. Αν δεν δηλωθούν ή δεν χρησιμοποιηθούν σωστά οι **μεταβλητές**[24] ή οι **σταθερές** ή οι **περιγραφικές σταθερές**[25] (literal), τα δεδομένα θα είναι ανακριβή και κατά συνέπεια τα αποτελέσματα θα είναι ανακριβή.

4.3.1 Μεταβλητές

Οι μεταβλητές έχουν χαρακτηριστικά. Τα χαρακτηριστικά αυτά αναφέρονται στο όνομα

[24] Οι μεταβλητές (variables), οι σταθερές (constants), και οι περιγραφικές σταθερές (literals) αποτελούν τα είδη των δεδομένων.

[25] Οι περιγραφικές σταθερές (literals) είναι συμβάσεται κατι διαφορετικό από τις σταθερές (constants), για περισσότερες λεπτομέρειες βλπ §4.3.2 και §4.3.3.

που προσδιορίζει την μεταβλητή, τον **προσδιοριστή** (identifier), στον **τύπο δεδομένων** (data type) που ανήκει και τέλος στην **τιμή** (value) που έχει δηλαδή που τις έχει ανατεθεί. Η κάθε μεταβλητή η οποία θα χρησιμοποιηθεί στο πρόγραμμα δηλώνεται με κάποια από τα χαρακτηριστικά της (προσδιοριστή και τύπο) στις **δηλώσεις** (declarations) στο πρόγραμμα. Οι δηλώσεις μεταβλητών στην C++ μπορεί να γίνουν σε οποιοδήποτε σημείο του προγράμματος, αρκεί οι μεταβλητές να μη χρησιμοποιηθούν πριν δηλωθούν. Είναι δυνατόν μια μεταβλητή συγχρόνως με τη δήλωσή της να της ανατεθεί αρχική τιμή.

4.3.1.1 Προσδιοριστές

Σε κάθε μεταβλητή σε ένα πρόγραμμα αντιστοιχεί ένας και μόνο προσδιοριστής. Για τη σύνθεση των προσδιοριστών ισχύουν οι παρακάτω κανόνες:

i) Οι προσδιοριστές συνθέτονται από αλφαριθμητικούς χαρακτήρες και τον ειδικό χαρακτήρα (_ υπογράμμισης - undercore, αντί για το κενό που δεν επιτρέπεται), πάντοτε ο πρώτος χαρακτήρας είναι αλφαβητικός.

Παραδείγματα: (α) **αποδεκτών προσδιοριστών**, meg, megelah, athr1, athr_1, diadio.
(β) **μη αποδεκτών προσδιοριστών**, 1athr, 1_athr.

Σημείωση: Οι προσδιοριστές θα πρέπει να αντιστοιχούν στην έννοια που αντιπροσωπεύουν, π.χ. MISTHOS για τον προσδιοριστή μεταβλητής που αντιστοιχεί σε μισθό κλπ.

ii) Το μήκος των προσδιοριστών, δηλαδή το πλήθος των χαρακτήρων που τους συνθέτουν, προσδιορίζεται από τον εκάστοτε μεταγλωττιστή, συνήθως είναι μέχρι 127 χαρακτήρες.

iii) Τα πεζά και τα κεφαλαία γράμματα δεν είναι ισοδύναμα. Έτσι οι μεταβλητές megelah, MegElah και MEGELAH είναι διαφορετικές.

iv) Οι περισσότεροι μεταγλωττιστές χρησιμοποιούν εσωτερικά ονόματα, που αρχίζουν ή τελειώνουν με τον ειδικό χαρακτήρα " _ " ή περιέχουν διπλό αυτόν τον ειδικό χαρακτήρα " _ " , γιαυτό, καλό θα είναι να αποφεύγεται η χρησιμοποίησή τους στους προσδιοριστές που ορίζονται παρόμοια. Για παράδειγμα τα ονόματα που έπονται θα πρέπει να αποφεύγονται, _card, card_ , amount_card. Με άλλα λόγια επινοείτε προσδιοριστές που δεν

χρησιμοποιούνται από το σύστημα σαν εντολές ή για να ορίσουν ενσωματωμένες συναρτήσεις.

4.3.1.2 Τύποι Μεταβλητών

Οι μεταβλητές μπορούν να περιέχουν διάφορους **τύπους δεδομένων**. Παραθέτουμε ορισμένους τύπους μεταβλητών της C++, προτάσσοντας το **όνομα δήλωσης του τύπου** και στην συνέχεια αναφέροντας **τον τύπο** που συμβολίζει το όνομα δήλωσής του:

char	χαρακτήρας
signed char	χαρακτήρας με πρόσημο (ταυτίζεται με τον **char**)
unsigned char	χαρακτήρας χωρίς πρόσημο
int	ακέραιος
signed int	ακέραιος με πρόσημο (ταυτίζεται με τον **int**)
unsigned int	ακέραιος χωρίς πρόσημο
short int	μικρός ακέραιος
signed short int	μικρός ακέραιος με πρόσημο (ταυτίζεται με τον **short int**)
unsigned short int	μικρός ακέραιος χωρίς πρόσημο
long	μεγάλος ακέραιος
long int	μεγάλος ακέραιος (ταυτίζεται με τον **long**)
signed long int	μεγάλος ακέραιος με πρόσημο (ταυτίζεται με τον **long int**)
unsigned long int	μεγάλος ακέραιος χωρίς πρόσημο
float	πραγματικός κινητού δεκαδικού σημείου (υποδιαστολής)
double	διπλής ακρίβειας πραγματικός κινητού δεκαδικού σημείου
long double	διπλής ακρίβειας μεγάλος πραγματικός κινητού δεκαδικού σημείου

Για την ώρα οι σημαντικότεροι τύποι που χειριζόμαστε είναι οι **char**, **int** και **float**. Με το πρόθεμα **long** μια ματαβλητή αποθηκεύει μεγαλύτερες τιμές απ' ότι συνήθως. Το πρόθεμα **unsigned** επιτρέπει σε μια μεταβλητή να αποθηκεύει μόνο θετικούς αριθμούς. Στην επόμενη παράγραφο περιγράφονται αναλυτικά οι τύποι αυτοί. Επικεντρώνουμε την προσοχή μας στη δήλωσή τους πριν τη χρήση τους.

4.3.1.3 Δηλώσεις

Τα δύο σημεία όπου δηλώνεται μια μεταβλητή είναι:

1. Πριν τον κώδικα που χρησιμοποιεί τη μεταβλητή.
2. Πριν το όνομα μιας συνάρτησης, για παράδειγμα πριν την **main( )** στο πρόγραμμα.

Το πρώτο σημείο είναι το συνηθέστερο, αυτό χρησιμοποιούμε συνήθως, συγκεντρώνοντας όλες τις δηλώσεις στην αρχή του κώδικά μας. Αν μια μεταβλητή δηλωθεί πριν το όνομα (προσδιοριστή) μιας συνάρτησης, τότε αυτή καλείται **καθολική μεταβλητή** (global variable - βλπ §4.8).

Για να δηλωθεί μια μεταβλητή πρέπει να δοθεί ο τύπος της και το όνομά της. Στο παράδειγμα 4.1.1 της §4.1, δηλώθηκαν τέσσερις μεταβλητές μετά την εκκίνηση της main( ), **εσωτερικές μεταβλητές** (internal variables - βλπ §4.8):

```
main( )
{
  int i, j;  // δήλωση των μεταβλητών i και j ως ακεραίων
  char c; // δήλωση της μεταβλητής c ως χαρακτήρα
  float x; // δήλωση της μεταβλητής x ως κινητού δεκαδικού σημείου
  // υπόλοιπο πρόγραμμα
}
```

Με τη δήλωση των δύο ακεραίων μεταβλητών με προσδιοριστές i και j, δεν έπεται ότι γνωστοποιούμε τους αριθμούς (τιμές) που υπάρχουν αποθηκευμένοι μέσα στις μεταβλητές, ούτε φυσικά μπορούμε να υποθέσουμε ότι είναι δεδομένοι, αυτό γίνεται μόνο όταν ανατεθούν τιμές στους προσδιοριστές μέσα στο ή από το πρόγραμμα.

Παρατήρηση: Συνήθως οι μεταβλητές του ιδίου τύπου δηλώνονται στην ίδια γραμμή, π.χ. **int** i, j; , δεν βελτιώνουμε το ευανάγνωστο του προγράμματος αν τις δηλώσουμε με δύο διαφορετικές δηλώσεις σε δύο διαφορετικές γραμμές, π.χ.:

```
int i;
int j; .
```

4.3.1.4 Ανασκόπηση των Τύπων Δεδομένων

Η C++ έχει τους περισσότερους τύπους δεδομένων από όλες τις άλλες γλώσσες προγραμματισμού. Ο τύπος μεταβλητή είναι εύχρηστος, αλλά και η επιλογή του ανάμεσα στους διαφόρους τύπος δεν είναι δύσκολη όσο φαίνεται από την πρώτη ματιά.

Μια ακόμα ιδιαιτερότητα της γλώσσας εντοπίζεται στον χειρισμό χαρακτήρων και αλυσίδων χαρακτήρων. Σε μια μεταβλητή χαρακτήρα δεν μπορεί να αποθηκευθούν περισσότεροι του ενός χαρακτήρες, δηλαδή στη μεταβλητή χαρακτήρα μπορεί να περιέχεται μόνο ένας χαρακτήρας. Δεν υφίστανται μεταβλητές που να μπορούν να

αποθηκεύσουν αλυσίδες χαρακτήρων, δηλαδή δεν υπάρχουν μεταβλητές του τύπου **αλυσίδας χαρακτήρων** (strings). Στην περίπτωση χειρισμού αλυσίδων χαρακτήρων πρέπει να χρησιμοποιηθεί ένας σύνθετος τύπος μεταβλητής, ο οποίος συνδυάζει άλλους θεμελιώδεις τύπους, όπως για παράδειγμα μια **παράταξη** (array) (βλπ § 4.5.1).

Τύπος	Περιοχή *
char	-128 ως 127
unsigned char	0 ως 255
signed char	-128 ως 127
int	-32768 ως 32767
unsigned int	0 ως 65535
signed int	-32768 ως 32767
short int	-32768 ως 32767
unsigned short int	0 ως 65535
signed short int	-32768 ως 32767
long int	-2147483648 ως 21474833647
signed long int	-2147483648 ως 21474833647
float	-3.4E-38 ως 3.4E+38
double	-1.7E-38 ως 3.4E+38
long double	-3.4E-38 ως 3.4E+38

ΠΙΝΑΚΑΣ 4.3.1: Τυπικές περιοχές τιμών για μεταβλητές της C++.
* Χρησιμοποιείτε αυτόν τον πίνακα σαν οδηγό. Διαφορετικοί μεταγλωττιστές και διαφορετικοί Η/Υ μπορεί να έχουν διαφορετικές τιμές.

Στον χώρο της Ε.Η/Υ λέγοντας **ακέραιο** εννοούμε κάθε αριθμό που δεν περιέχει δεκαδικά ψηφία, π.χ. -243, 43, κλπ. (σε αυτούς περιλαμβάνονται οι **φυσικοί** και οι **ακέραιοι αριθμοί** όπως ορίζονται στα Μαθηματικά). Αντίθετα εάν ο αριθμός περιέχει δεκαδικά ψηφία τότε λέγεται **κινητού δεκαδικού σημείου** (κ.δ.σ.) ή **κινητής υποδιαστολής** (σε αυτούς περιλαμβάνονται οι **πραγματικοί αριθμοί** - **ρητοί** και **άρρητοι** - όπως ορίζονται στα Μαθηματικά). Κάθε φορά που πρέπει να αποθηκευθεί ένας μισθός, μια θερμοκρασία ή οποιοσδήποτε αριθμός με δεκαδικό μέρος, πρέπει να αποθηκευθεί σαν μεταβλητή κινητού δεκαδικού ψηφίου, π.χ. 0.00, 36.45, -234.678, .0136,

κλπ. Άλλοτε χρειάζεται να αποθηκευθούν μεγάλοι αριθμοί και άλλοτε πάλι μικροί, ο πίνακας 4.3.1 δείχνει έναν κατάλογο περιοχών που μπορεί να αποθηκεύει κάθε τύπος μεταβλητής.

Παρατηρήσεις:

1. Υπογραμμίζουμε ότι οι προγραμματιστές της AT&T C++ δεν εμπιστεύονται τις ακριβείς τιμές του πίνακα 4.3.1σε κάθε Η/Υ που χρησιμοποιεί την C++. Οι περιοχές αυτές είναι κοινές για τα PC, μπορεί όμως να είναι διαφορετικές σε άλλους Η/Υ. Γι' αυτό χρησιμοποιείται τον πίνακα αυτόν μόνο σαν οδηγό.

2. Οι περιοχές κινητού δεκαδικού σημείου (float, double και long double) του πίνακα 4.3.1 εμφανίζονται σε εκθετική μορφή (Ε σημαίνει εκθέτης - exponent). Για να καθοριστούν οι ακριβείς περιοχές πρέπει να υπολογιστούν οι ποσότητες σε εκθετική μορφή, για παράδειγμα: -3.4E-38 = $-3.4 \cdot 10^{-38}$.

Οι μεγάλες ακέραιες (long int) και μεγάλες διπλής ακρίβειας (long double) μεταβλητές αποθηκεύουν μεγαλύτερους αριθμούς, με αποτέλεσμα να έχουν μεγαλύτερη ακρίβεια, από τις κανονικές ακέραιες (int) και τις κανονικές διπλής ακρίβειας (double) κινητού δεκαδικού σημείου μεταβλητές. Αυτό οφείλεται στο ότι για αυτούς τους τύπους χρησιμοποιείται από πολλούς μεταγλωττιστές της C++ μεγάλος αριθμός θέσεων μνήμης.

Επισημαίνουμε ότι, στις περισσότερες περιπτώσεις, δεν χρειάζεται για όλα τα δεδομένα να χρησιμοποιούνται μεγάλες μεταβλητές. Αυτό σίγουρα φαίνεται στην πράξη. Για παράδειγμα, χρειάζονται μεγάλες μεταβλητές όταν τα δεδομένα είναι πολύ μεγάλοι ή πολύ μικροί δεκαδικοί αριθμοί, διότι τότε, μπορεί να υπερχειλίσουν από τις περιοχές των κανονικών τύπων δεδομένων, και πάλι οι περιοχές αυτές διαφέρουν ανάλογα τον Η/Υ. Επίσης είναι προφανές ότι, όσες περισσότερες θέσεις μνήμης χρησιμοποιούνται από τα δεδομένα, τόσο μεγαλύτερα μπορεί να είναι τα δεδομένα. Κάθε φορά όμως που ο Η/Υ πρέπει να προσπελάσει περισσότερο χώρο μνήμης για μια μεταβλητή, όπως συμβαίνει συνήθως με τις μεγάλες μεταβλητές, η **κεντρική μονάδα επεξεργασίας**[37] (**ΚΜΕ** - central processing unit **CPU**) κάνει περισσότερο χρόνο για την προσπέλαση, υπολογισμό

[37] Είναι το μέρος του Η/Υ που αποτελείται από την αριθμητική - λογική μονάδα (arithmetic-logic unit) και τη μονάδα ελέγχου (control unit). Όλοι οι Η/Υ έχουν μια ΚΜΕ (CPU). Είναι ο χώρος που εισάγονται στον Η/Υ οι εντολές, αποκωδικοποιούνται και εκτελούνται, και από τον οποίο ελέγχεται κάθε ενέργεια του Η/Υ.

και αποθήκευση.

Γενικά, όλες οι αριθμητικές μεταβλητές πρέπει να είναι προσημασμένες (η εξ ορισμού κατάσταση), εκτός και αν είναι βέβαιο ότι τα δεδομένα περιέχουν μόνο θετικούς αριθμούς (ηλικία, απόσταση, κλπ.). Μετατρέποντας μια μεταβλητή σε μη προσημασμένη, κερδίζεται λίγος χώρος αποθήκευσης, γεγονός που προυποθέτει ότι η περιοχή αυτή είναι πάντα θετική.

Στον πίνακα 4.1.1 αναφέρεται ότι οι μεταβλητές του τύπου χαρακτήρα μπορούν να περιέχουν αριθμητικές τιμές. Στην C++, οι ακέραιες μεταβλητές και οι μεταβλητές χαρακτήρα μπορούν να χρησιμοποιηθούν εναλλακτικά. Κάθε χαρακτήρας στον πίνακα ASCII[29] έχει ένα μοναδικό αριθμό που αντιστοιχεί με τη θέση του στον πίνακα. Αν αποθηκευθεί ένας αριθμός σε μια μεταβλητή χαρακτήρα, η C++ χειρίζεται τα δεδομένα σαν να ήταν χαρακτήρας ASCII που αντιστοιχεί στον αριθμό αυτόν στον πίνακα. Αντίστοιχα, μπορεί να αποθηκευθούν χαρακτήρες σε μια ακεραία μεταβλητή. Η C++ βρίσκει τον αριθμό ASCII αυτού του χαρακτήρα και αποθηκεύει αυτόν τον αριθμό αντί του χαρακτήρα.

4.3.2 Περιγραφικές Σταθερές και οι Τύποι τους

Θα αναφερθούμε συνοπτικά στις περιγραφικές σταθερές για την πληρότητα της παρουσίασης των τύπων δεδομένων. Όπως και με τις μεταβλητές υπάρχουν διάφοροι τύποι περιγραφικών σταθερών της C++. Έχουμε τις περιγραφικές σταθερές ακεραίους, τις περιγραφικές σταθερές κινητού δεκαδικού σημείου, τις περιγραφικές σταθερές αλυσίδες χαρακτήρων και τέλος τις περιγροφικές σταθερές χαρακτήρες.

Η C++ επιτρέπει να ανατεθούν περιγραφικές σταθερές ακέραιοι σε μεταβλητές, να

---

[29] Ακρωνύμιο του American Standard Code For Information Interchange. Ενας τυποποιημένος κώδικας από διαδοχικών ψηφίων, ο οποίος κατά κανόνα χρησιμοποιείται με ένα ΨΗΦΙΟ ΕΞΙΟΧΩΣΗΣ (για να υπάρχει ένα σύνολο οκτώ ψηφίων ανά χαρακτήρα).

Ο κώδικας καθιερώθηκε από το Αμερικανικό Εθνικό Ινστιτούτο Τυποποιήσεων, για να εξασφαλίσθεί εναλλαξιμότητα μεταξύ των διαφόρων τύπων συσκευών που χρησιμοποιούνται για την επεξεργασία και διαβίβαση στοιχείων.

Ο ASCII είναι ο πιό διαδεδομένος κώδικας παράστασης χαρακτήρων, περιλαμβάνει όλους τους απαραίτητους γραφικούς χαρακτήρες, όπως τα λατινικά γράμματα, αριθμούς, σημεία στίξης και ειδικά σύμβολα που χρησιμοποιούνται στις γλώσσες προγραμματισμού και τις εντολές των λειτουργικών συστημάτων, τα οποία αποκτούν έτσι ένα κώδικα ταυτότητα. Στους γραφικούς αυτούς χαρακτήρες αντιστοιχεί και μια γραφική παράσταση. Ο ASCII είναι ο περισσότερο χρησιμοποιούμενος κώδικας για όλες τις συσκευές εκτός από αυτές της IBM.

χρησιμοποιηθούν περιγραφικές σταθερές ακέραιοι σε υπολογισμούς και να εκτυπωθούν περιγραφικές σταθερές ακέραιοι με τον τελεστή cout. Μια **περιγραφική σταθερά ακέραιος** μπορεί να διατυπωθεί στη δεκαδική μορφή, π.χ. 16, 65536, 25 ή αντίστοιχα στην οκταδική μορφή (με βάση οκταδικό σύστημα αρίθμησης) **0**20, **0**100000, **0**31 ή αντίστοιχα στην δεκαεξαδική μορφή (με βάση το δεκαεξαεδικό σύστημα αρίθμησης) **0x**10, **0x**10000, **0x**19. Η C++ ερμηνεύει το πρώτο **0** στη περιγραφική σταθερά ακέραιο σαν περιγραφική σταθερά σε οκταδική μορφή, ανάλογα το **0x** (ή **0X**) σαν περιγραφική σταθερά σε δεκαεξαεδική μορφή. Η C++ αναγνωρίζει ως **περιγραφική σταθερά κινητού δεκαδικού σημείου** τους αριθμούς με αρχικό μηδενικό με μορφή, π.χ., 0.16, 0.65536, 0.25, κλπ. Σημειώνουμε ότι μόνο οι ακέραιοι μπορεί να είναι οκταδικοί ή δεκαεξαεδικοί.

Ό τύπος της C++ που λέγεται περιγραφική σταθερά αλυσίδα χαρακτήρων δεν έχει αντίστοιχη μεταβλητή. Παραδείγματα αυτού του τύπου είναι: "Georgios Prasinos", "425 Dodonis str", "8816026", "x", " ", κλπ. Δηλαδή είναι **περιγραφική σταθερά αλυσίδα χαρακτήρων** ένα διάστημα, ένας χαρακτήρας, μια λέξη, ή μια ομάδα λέξεων σε διπλές αποστρόφους. Αντίθετα μια **περιγραφική σταθερά χαρκτήρας** είναι κάθε χαρακτήρας σε απλές αποστρόφους, π.χ. 'x', '8', κλπ.

4.3.3 Σταθερές

Ο όρος σταθερά χρησιμοποιείται για τις σταθερές τιμές των προγραμμάτων της C++. Μια σταθερά, σε ένα πρόγραμμα C++, έχει χαρακτηριστικά ανάλογα αυτών των μεταβλητών, δηλαδή όνομα που προσδιορίζει την σταθερά, τον **προσδιοριστή** (identifier), τον **τύπο δεδομένων** (data type) που ανήκει και τέλος την **τιμή** (value) που έχει δηλαδή που τις έχει ανατεθεί. Η κάθε σταθερά η οποία θα χρησιμοποιηθεί στο πρόγραμμα δηλώνεται με την δεσμευμένη λέξη **const** και με τα χαρακτηριστικά της (τύπο, προσδιοριστή και τιμή) στις **δηλώσεις** (declarations) στο πρόγραμμα. Οι δηλώσεις σταθερών στην C++ μπορεί να γίνουν σε οποιοδήποτε σημείο του προγράμματος, αρκεί αυτές να μη χρησιμοποιηθούν πριν δηλώθούν.

Ένας πρακτικός τρόπος για να διακρίνονται οι μεταβλητές από τις σταθερές στο πρόγραμμα είναι οι σταθερές να πληκτρολογούνται με κεφαλαία γράμματα, είναι η περίπτωση όπου τα κεφαλαία γράμματα μπορούν να χρησιμοποιηθούν ευρέως στην C++,

π.χ. : **const float** Pi=3.14159.

4.3.4 Ειδικοί Χαρακτήρες ή Αλυσίδες Διαφυγής

Κάθε χαρακτήρας (ή αλυσίδα) του οποίου προηγείται μια ανάστροφη κάθετος " \ " λέγεται χαρκτήρας διαφυγής (ή αλυσίδα διαφυγής). Ο πίνακας 4.3.1 δείχνει ορισμένους χαρακτήρες διαφυγής που είναι χρήσιμοι για εκτυπώσεις ειδικών χαρακτήρων.

ΧΑΡΑΚΤΗΡΕΣ Ή ΑΛΥΣΙΔΕΣ ΔΙΑΦΥΓΗΣ ΚΑΙ Η ΣΗΜΑΣΙΑ ΤΟΥΣ	
\a	Συναγερμός (το κουδούνι του τερματικού)
\b	Οπισθοδρόμηση
\f	Τροφοδότηση γραμμής (για τον εκτυπωτή)
\n	Νέα γραμμή (επαναφορά και τροφοδότηση)
\r	Επαναφορά
\t	Tab
\v	Κατακόρυφο tab
\\	Ανάστροφη κάθετος
\?	Αγγλικό ερωτιματικό
\'	Απλή απόστροφος
\"	Διπλές απόστροφοι
\000	Οκταδικός αριθμός
\xhh	Δεκαεξαδικός αριθμός
\0	Τερματικό μηδενικό (ή δυαδικό μηδενικό)

Πίνακας 4.3.4 : Ειδικοί χαρακτήρες ή αλυσίδες διαφυγής της C++

4.3.5 Ασκήσεις (Ερωτήσεις για Επανάληψη και Εμπέδωση)

1. Ποιά από τα παρακάτω ονόματα μεταβλητών είναι έγκυρα;

(α) onoma_mathiti, (β) 127_poleis. poleis_127, a-misthos.

2. Ποιές από τις παρακάτω περιγραφικές σταθερές είναι ακέραιοι, κινητής υποδιαστολής χαρκτήρες ή αλυσίδες χαρακτήρων;

0 -12.0 "2.0" 'x' 'y' 125.4 -808 '0'.

3. Πόσες μεταβλητές δηλώνουν οι παρακάτω δηλώσεις και ποιοί είναι οι τύποι τους;

(α) **int** i, j, k; (β) **char** c, d, e; (γ) **float** x = 653.56;

4. Με τι τελειώνουν όλες οι περιγραφικές σταθερές τύπου αλυσίδας χαρακτήρων;

5. Είναι αληθές ή ψευδές : "Μια προσημασμένη μεταβλητή μπορεί να περιέχει μεγαλύτερη τιμή από μια προσημασμένη."

6. Πόσους χαρακτήρες αποθήκευσης απαιτεί η περιγραφική σταθερά : '\x41' .

7. Για να τυπώστε δύο ονόματα σε δύο διαφορετικές γραμμές ποιά από τις παρακάτω προτάσεις είναι σωστές;

(α) **cout** << " Pappas\nKappas "; (β) **cout** << " Pappas " << "\n"<< " Kappas ";

8. Τι είναι λάθος με το παρακάτω πρόγραμμα;

```
main( )
{
 const int ilikia = 35;
 cout << " ilikia << "\n";
 ilikia = 52;
 cout << " ilikia << "\n";
 return(0);
}
```

9. Γράψτε κώδικα σε C++ για να αποθηκεύσετε τρείς μεταβλητές, το βάρος σας, το ύψος σας και τον αριθμό παπουτσιών που φοράτε. Δηλώστε τις μεταβλητές και μετά αναθέστε τους τιμές στο σώμα του προγράμματος.

10. Ξαναγράψτε το πρόγραμμα της προηγούμενης άσκησης, προσθέτοντας προτάσεις **cout** για να τυπώσετε τις τιμές στην οθόνη. Χρησιμοποιείστε κατάλληλα μηνύματα (εκτυπώνοντας"περιγραφικές σταθερές τύπου αλυσίδας χαρακτήρων") για να περιγράψτε τους αριθμούς που εκτυπώνονται.

11. Γράψτε ένα πρόγραμμα που αποθηκεύει μια τιμή και εκτυπώνει κάθε τύπο μεταβλητής που μάθατε σε αυτό το κεφάλαιο.

12. Γράψτε ένα πρόγραμμα που αποθηκεύει μια τιμή σε κάθε τύπο μεταβλητής που επιτρέπει η C++. Πρέπει να δηλώσετε κάθε μεταβλητή στην αρχή του προγράμματος. Δώστε τιμές στις μεταβλητές και εκτυπώστε αυτές,

13. Γράψτε πρόγραμμα το οποίο θα αναθέτει την αλυσίδα διαφυγής "\ a" σε μια μεταβλητή η οποία καθώς θα εκτυπώνεται θα κτυπά το κουδούνι του Η/Υ.

4.4 Τελεστές της C++

4.4.1 Τελεστές για Ακεραίου Τύπου Δεδομένα

4.4.1.1 Αριθμητικοί Τελεστές [39] :

- \+ πρόσθεση[40], πρόσημο[41]
- \- αφαίρεση, πρόσημο
- * πολλαπλασιασμός
- / διαίρεση[42]
- % υπόλοιπο ακέραιας διαίρεσης[43]
- x = υπολογισμός και καταχώρηση, όπου x μπορεί να είναι +, -, *, /, %[44]
- ++ αύξηση κατά 1(προσαύξηση) (βλπ Πίνακα 4.4.2)
- -- ελάττωση κατά 1(μείωση)(βλπ Πίνακα 4.4.2

4.4.1.2 Λογικοί και Σχεσιακοί Τελεστές [45] :

&& AND (σύζευξη)

---

[39] **Αριθμητικοί τελεστές** είναι τα ειδικά σύμβολα που χρησιμοποιούνται για τις αριθμητικές πράξεις, π.χ. πρόσθεση, αφαίρεση κλπ.

[40] **Δυαδικός τελεστής**, όταν ένας τελεστής χρησιμοποιείται μεταξύ δυο τελεστέων, μεταβλητών κλπ., δηλαδή λειτουργεί με τη χρήση δύο τιμών.

[41] **Εναδικός τελεστής** είναι τμήμα κωδικού που αποτελεί ή λειτουργεί σαν μια τιμή, π.χ. η ανάθεση σε μια μεταβλητή ενός θετικού ή ενός αρνητικού αριθμού με τη χρήση ενός **εναδικού + ή -** .

[42] Παράγει έναν ακέραιο, αν υπάρχει υπόλοιπο η C++ το αγνοεί.

[43] Παράγει ένα υπόλοιπο μιας διαίρεσης ακεραίων. Προσοχή! για να λειτουργήσει απαιτεί να υπάρχουν ακέραιοι και στις δύο πλευρές του συμβόλου.

[44] Δηλαδή έχουμε τους τελεστές +=, - =, *= και /= οι οποίοι λέγονται **σύνθετοι τελεστές**, π.χ. η έκφραση sum+=300 είναι ισοδύναμη με την έκφραση sum=sum+300.

Οι σύνθετοι τελεστές βρίσκονται χαμηλά στον πίνακα προτεραιότητας (βλπ 4.4.3). **Οι σύνθετοι τελεστές κανονικά υπολογίζονται σχεδόν τελευταίοι.**

Με τους σύνθετους τελεστές πραγματοποιούμε **σύνθετες αναθέσεις τιμών.** Η έκφραση daynum%=7, είναι μια σύνθετη ανάθεση.

[45] **Σχεσιακοί τελεστές** είναι αυτοί που χρησιμοποιούντα για σύγκριση δεδομένων. **Ενώ Λογικοί τελεστές** είναι αυτοί που επιτρέπουν να συνδυαστούν οι σχεσιακοί τελεστές σε δυναμικότερες προτάσεις ελέγχου.

|| OR (διάζευξη)[46]
! NOT (άρνηση)
== σύγκριση για ισότητα
!= σύγκριση για ανισότητα
> σύγκριση για μεγαλύτερο
>= σύγκριση για μεγαλύτερο ή ίσο
< σύγκριση για μικρότερο
<= σύγκριση για μικρότερο ή ίσο

4.4.1.2.1 Πίνακες Αληθείας Λογικών Τελεστών [47]

Αληθές	AND	Αληθές = Αληθές
Αληθές	AND	Ψευδές = Ψευδές
Ψευδές	AND	Αληθές = Ψευδές
Ψευδές	AND	Ψευδές = Αληθές

ΠΙΝΑΚΑΣ 4.4.1α: Ο πίνακας αληθείας του AND (&&)
(Και οι δύο πλευρές πρέπει να είναι αληθείς)

Αληθές	OR	Αληθές = Αληθές
Αληθές	OR	Ψευδές = Αληθές
Ψευδές	OR	Αληθές = Αληθές
Ψευδές	OR	Ψευδές = Ψευδές

ΠΙΝΑΚΑΣ 4.4.1β: Ο πίνακας αληθείας του OR (||)
(Τουλάχιστον μια πλευρά πρέπει να είναι αληθής)

NOT	Αληθές = Ψευδές
NOT	Ψευδές = Αληθές

ΠΙΝΑΚΑΣ 4.4.1γ: Ο πίνακας αληθείας του NOT (!)
(Έχει σαν αποτέλεσμα το αντίστροφο)

---

[46] Οι τελεστές **&&** και || δεν εμφανίζονται ποτέ μόνοι τους. Συνήθως εμφανίζονται ανάμεσα σε δύο ή περισσότερους ελέγχους.

[47] Οι πίνακες 4.1.1α,β,γ δείχνουν πως λειτουργεί κάθε λογικός τελεστής. Οι πίνακες αυτοί λέγονται **πίνακες αληθείας**, επειδή δείχνουν πως συνάγονται "**Αληθή** (True) **Συμπεράσματα**" από μια πρόταση **if**, που χρησιμοποιεί αυτούς του τελεστές. Επαναλαμβάνουμε ότι οι λογικοί τελεστές επιτρέπουν στον χρήστη να υπολογίσει σύνθετους σχεσιακούς ελέγχους.

4.4.1.3 Ψηφιακοί Τελεστές ή Τελεστές Δυαδικών[46] :

&	AND
|	OR (inclusive OR)
^	XOR (exclusive OR)
~	συμπλήρωμα 1
>>	μετατόπιση (ολίσθηση) προς τα δεξιά
<<	μετατόπιση (ολίσθηση) προς τα αριστερά
x =	υπολογισμός και καταχώρηση, όπου x μπορεί να είναι &, |, ^, >>, <<

4.4.2 Τελεστές για Τύπου Δεκαδικής Μορφής Δεδομένα

4.4.2.1 Αριθμητικοί Τελεστές[47] :

---

[46] Χειρίζονται τις εσωτερικές αναπαραστάσεις των δεδομένων, απαιτούν γνώση του συστήματος δυαδικής αρίθμησης και της μνήμης του Η/Υ, αφού χρησιμοποιούνται για τεχνικές προχωρημένου προγραμματισμού, που δεν καλύπτει το παρόν.

[47] **Είναι δυνατόν να αναμειχθούν τύποι δεδομένων** στην C++. Για παράδειγμα η πρόσθεση ενός ακεραίου αριθμού (int) και ενός αριθμού κινητού δεκαδικού σημείου (float) είναι μια ανάμειξη τύπων δεδομένων. Η C++ γενικά μετατρέπει τον μικρότερο από τους δύο τύπους στον άλλο. Ακόμα ένα παράδειγμα, εάν προστεθεί ένας διπλής ακρίβειας αριθμός (double) με έναν ακέραιο αριθμό (int), η C++ θα μετατρέψει πρώτα τον ακέραιο σε διπλής ακρίβειας αριθμό και στη συνέχεια θα εκτελέσει τον υοπολογισμό. Αυτή η μέθοδος παράγει το ακριβέστερο αποτέλεσμα. Η αυτόματη μετατροπή των τύπων των δεδομένων είναι προσωρινή. Η τιμή που μετατράπηκε θα βρεθεί στον αρχικό τύπο με τη λήξη της διαδικασίας. Επισημαίνουμε ότι σε αντίθετη περίπτωση, της μετατροπής δηλαδή του μεγαλυτέρου τύπου στον άλλο θα είχαμε μείωση της ακρίβειας.

Η **προσωρινή αλλαγή τύπου δεδομένων μπορεί να πραγματοποιηθεί από τον ίδιο τον προγραμματιστεί αν χρειαστεί**. Η διαδικασία αυτή προβλέπει δύο τύπους προσωρινής αλλαγής, οι οποίοι είναι:

1. **(<τυπος_δεδομενων>)<εκφραση>** και 2. **<τυπος_δεδομενων>(<εκφραση>)**

όπου: <τυπος_δεδομενων> ::= **int** | **float** | . . . και

<εκφραση> ::= <εκφραση><αριθμητικος_τελεστης><εκφραση> |
<μεταβλητη> | <σταθερη> | <περιγραφικη_σταθερη>.

Παράδειγμα: Έστω ότι είναι: **float age; double factor ;**

1. **age_factor = (double) age * factor ;** // προσωρινή αλλαγή της age σε factor, 1ος τύπος
2. **age_factor = double(age) * factor ;** // προσωρινή αλλαγή της age σε factor, 2ος τύπος

+ πρόσθεση, πρόσημο
- αφαίρεση, πρόσημο
* πολλαπλασιασμός
/ διαίρεση
x = υπολογισμός και καταχώρηση, όπου x μπορεί να είναι +, -, *, /
++ αύξηση κατά 1(προσαύξηση) (βλπ πιν 4.4.2)
-- ελάττωση κατά 1(μείωση) (βλπ πιν 4.4.2)

Τελεστής	Παράδειγμα	Περιγραφή	Ισοδύναμες Προτάσεις	
++	i++	επίθεμα	i = i + 1;	i += 1;
++	++i	πρόθεμα	i = i + 1;	i += 1;
--	i--	επίθεμα	i = i - 1;	i -= 1;
--	--i	πρόθεμα	i = i - 1;	i -= 1;

ΠΙΝΑΚΑΣ 4.4.2: Οι Τελεστές Προσαύξησης (++) και Μείωσης (--) [50]

4.4.2.2 Λογικοί και Σχεσιακοί Τελεστές:

&& AND
|| OR
! NOT
== σύγκριση για ισότητα
!= σύγκριση για ανισότητα

[50] Οι δύο αυτοί τελεστές μεταγλωττίζονται άμεσα στα ισοδύναμά τους σε συμβολική γλώσσα (assembly). Οι αντίστοιχες όμως εκφράσεις, όπως i = i + 1 και i = i - 1 δεν μεταγλωττίζονται στις αντίστοιχες εντολές της γλώσσας μηχανής.

Το πρόθεμα ή το επίθεμα δεν παίζει ρόλο αν προσαυξάνονται ή μειώνονται μεμονωμένες μεταβλητές σε γραμμές μόνες τους. Αν όμως οι τελεστές αυτοί συνδυάζονται με άλλους σε μια έκφραση τότε:

1. Αν μια μεταβλητή προσαυξάνεται ή μειώνεται με ένα τελεστή πρόθεμα, τότε η προσαύξηση ή η μείωση γίνεται πριν χρησιμοποιηθεί η τιμή της μεταβλητής στην υπόλοιπη έκφραση, π.χ. αν a = 6; b = ++a - 1; τότε έχουμε : b = (a + 1) - 1 δηλαδή b = 7 - 1 = 6.

2. Αν μια μεταβλητή προσαυξάνεται ή μειώνεται με ένα τελεστή επίθεμα, τότε η προσαύξηση ή η μείωση γίνεται μετά τη χρησιμοποίηση της τιμής της μεταβλητής στην υπόλοιπη έκφραση, π.χ. αν a = 6; b = a++ - 1; τότε έχουμε b = (a - 1) δηλαδή b = 5 και a++.

Προσοχή: 1.Οι τελεστές αυτοί εφαρμόζονται μόνο σε μεταβλητές, ποτέ σε εκφράσεις, π.χ. sales = ++(rate * hours); // δεν επιτρέπεται, είναι λάθος. 2.Πρέπει να αποφεύγονται οι τελεστές προσαύξησης ή μείωσης σε κλήσεις συναρτήσεων διότι, στην περίπτωση αυτή, δεν μπορεί να προβλεφθεί η σειρά με την οποία θα εκτελεστούν.

> σύγκριση για μεγαλύτερο
>= σύγκριση για μεγαλύτερο ή ίσο
< σύγκριση για μικρότερο
<= σύγκριση για μικρότερο ή ίσο

4.4.3 Σειρά Προτεραιότητας Τελεστών στις Διάφορες Παραστάσεις

επίπεδο προτεραιότητας 1 (υψηλότερο)
ο συσχετισμός πραγματοποιείται από **αριστερά προς τα δεξιά**

( ) κλήση συνάρτησης
[ ] δεικτοδότηση πίνακα
→ επιλογή πεδίου δομής δεδομένων με ενδείκτη
:: προσπέλαση μορφής/ανάλυσης
. επιλογή πεδίου δομής δεδομένων

επίπεδο προτεραιότητας 2 (εναδικό)
ο συσχετισμός πραγματοποιείται από **δεξιά προς τα αριστερά**

! λογικό NOT (λογική άρνηση)
~ συμπλήρωμα δυαδικών(1) (κατά bit)
+ πρόσημο συν
- πρόσημο μείον
++ αύξηση κατά 1(προσαύξηση)
-- μείωση κατά 1
& η διεύθυνση ενός μεγέθους
* αναφορά στη διεύθυνση του μεγέθους (indirection)
sizeof επιστρέφει το μέγεθος μιας μεταβλητής ή ενός τύπου σε bytes[91]

---

[91] Μοιάζει με συνάρτηση αλλά είναι τελεστής. Η μορφή του είναι: **sizeof** <δεδομενα> ή **sizeof**(<δεδομενα>) και **sizeof**(<τυπος_δεδομενων>), λειτουργεί με μια μόνο τιμή και παράγει ένα αποτέλεσμα που παριστά το μέγεθος σε bytes των δεδομένων ή του τύπου δεδομένων. Οι περισσότεροι τύποι δεδομένων και μεταβλητές χρειάζονται διαφορετικά ποσά εσωτερικής αποθήκευσης, ανάλογα με τον Η/Υ, ο τελεστής sizeof επιτρέπει σε προγράμματα να έχουν συνέπεια για διαφόρους τύπους Η/Υ.
Παράδειγμα: Να βρεθεί το μέγεθος, σε bytes, των μεταβλητών κινητού δεκαδικού σημείου στον Η/Υ σας.

```
. . . main( )
    { cout<< " Το μέγεθος κινητού σημείου \n";
      cout<< " στον Η/Υ . . . είναι: " << sizeof(float) << "\n";
      return(0); }
```

Στους περισσότερους Η/Υ θα παραχθεί η έξοδος: Το μέγεθος κινητού σημείου στον Η/Υ . . . είναι: 4

Ο τελεστής αυτός μερικές φορές αναφέρεται και σαν **τελεστής χρόνου μεταγλώττισης**. Χρησιμοποιείται δε και για τεχνικές προχωρημένες προγραμματισμού.

new δυναμικός μερισμός αποθήκευσης της C++
delete δυναμικός υπομερισμός αποθήκευσης της C++

επίπεδο προτεραιότητας 3 (προσπέλαση)
ο συσχετισμός πραγματοποιείται από **αριστερά προς τα δεξιά**

.* τιμή που δείχνει ο δείκτης της C++
->* τιμή που δείχνει ο δείκτης της C++

επίπεδο προτεραιότητας 4 (πολλαπλασιαστικό)
ο συσχετισμός πραγματοποιείται από **αριστερά προς τα δεξιά**

***** πολλαπλασιασμός
/ διαίρεση
% υπόλοιπο : τρείς πράξεις με ίση προτεραιότητα[62]

επίπεδο προτεραιότητας 5 (προσθετικό)
ο συσχετισμός πραγματοποιείται από **αριστερά προς τα δεξιά**

+ πρόσθεση
- αφαίρεση

επίπεδο προτεραιότητας 6 (μετατόπιση)
ο συσχετισμός πραγματοποιείται από **αριστερά προς τα δεξιά**

<< μετατόπιση (ολίσθηση) αριστερά
>> μετατόπιση (ολίσθηση) δεξιά

[62] Η C++ εκτελεί τον πολλαπλασιασμό, τη διαίρεση και το υπόλοιπο πριν από την πρόσθεση και την αφαίρεση. **Οι παρενθέσεις προσπερνούν (αλλοιώνουν) την προτεραιότητα των πράξεων.** Για παράδειγμα, **2+3*2**, ισούται με **8**, επειδή ο πολλαπλασιασμός εκτελέστηκε πριν από την πρόσθεση. Η προσθήκη παρενθέσεων στην πρόσθεση, **(2+3)*2**, αλλοίωσε το αποτέλεσμα, που στην περίπτωση αυτή ισούται με **10**.

Προσοχή! Χρησιμοποιήστε πολλές παρενθέσεις στα προγράματά σας της C++ για να υποδείξετε τη σειρά των τελεστών, ακόμη και αν δεν χρειάζεται να προσπεράσετε την εξ ορισμού προτεραιότητα. Η χρήση παρενθέσεων κάνει τους υπολογισμούς ευκολότερους στην κατανόηση, όταν πρέπει να τροποποιήσετε το πρόγραμμα. Είναι σημαντικότερη η συγγραφή προγραμμάτων που είναι εύκολα στην κατανόηση από τα προγράμματα που είναι σύντομα ή περιλαμβάνουν πολύπλοκους υπολογισμούς.

επίπεδο προτεραιότητας 7 (σχεσιακός)

ο συσχετισμός πραγματοποιείται από **αριστερά προς τα δεξιά**

< μικρότερος από
<= μικρότερος ίσος από
> μεγαλύτερος
>= μεγαλύτερος : τέσσερεις συγκρίσεις με ισότιμη προτεραιότητα

επίπεδο προτεραιότητας 8 (ισοδυναμίας)

ο συσχετισμός πραγματοποιείται από **αριστερά προς τα δεξιά**

== σύγκριση για ισότητα (ίσος με)
!= σύγκριση για ανισότητα (όχι ίσος με)

επίπεδο προτεραιότητας 9

ο συσχετισμός πραγματοποιείται από **αριστερά προς τα δεξιά**

& δυαδικό AND (AND κατά bit)

επίπεδο προτεραιότητας 10

ο συσχετισμός πραγματοποιείται από **αριστερά προς τα δεξιά**

^ δυαδικό XOR (exclusive OR κατά bit)

επίπεδο προτεραιότητας 11

ο συσχετισμός πραγματοποιείται από **αριστερά προς τα δεξιά**

| δυαδικό OR (OR κατά bit)

επίπεδο προτεραιότητας 12[53]

---

[53] **Παραδείγματα:**

1.Έστω η πρόταση: **if (sales < min_sal * 2 && yrs_emp > 10 * sub)...**, με τη βοήθεια των προτεραιοτήτων των τελεστών καθορίζουμε πως θα εκτελεστεί. Η πρόταση εκτελείται ως εξής, πρώτα οι δύο πολλαπλασιασμοί, ακολουθούν οι σχέσεις < και >, ενώ το && εκτελείται τελευταίο. Δηλαδή, ταυτίζεται με: **if ((sales < min_sal * 2) && (yrs_emp > 10 * sub))...**

2. Έστωσαν : **if (first_initial == 'A') && (last_initial == 'G') || (id == 321)...** και

if (((first_initial == 'A') && (last_initial == 'G')) || (id == 321))...

οι οποίες ταυτίζονται, λόγω όμως των παρενθέσεων η δεύτερη είναι σαφέστερη.

Παρατηρήσεις:

1.Για καλλίτερη κατανόηση χρησιμοποιούμε παρενθέσεις, ακόμα και αν η προτεραιότητα που εισάγουν ταυτίζεται με την εξ ορισμού.

2.Πρέπει να αποφεύγονται πολλές εκφράσεις μέσα σε ένα **if**.

ο συσχετισμός πραγματοποιείται από **αριστερά προς τα δεξιά**

&& λογικό AND

επίπεδο προτεραιότητας 13

ο συσχετισμός πραγματοποιείται από **αριστερά προς τα δεξιά**

|| λογικό OR

επίπεδο προτεραιότητας 14 (υπό συνθήκη)

ο συσχετισμός πραγματοποιείται από **δεξιά προς τα αριστερά**

?: τελεστής συνθήκης[54]

επίπεδο προτεραιότητας 15 (ανάθεσης)

ο συσχετισμός πραγματοποιείται από **δεξιά προς τα αριστερά**

= τελεστής απλής ανάθεσης (καταχώρησης)[55]

x= **τελεστές υπολογισμού και ανάθεσης** (καταχώρησης)[56],
όπου x = * , / , % , + , - , & , ^ , | , << , >> δηλαδή:

---

[54] Ο μόνος τελεστής τριών τελεστέων: <συνθηκη> ? (<αποδοση_1>) : (<αποδοση_2>) ή (<συνθηκη>) ? (<αποδοση_1>) : (<αποδοση_2>), χρησιμοποιείται σε μερικές περιπτώσεις αντί της **if-else**.

Παράδειγμα: Η πρόταση **if** (a > b)
{ meg = a }
else
{ meg = b ; } ισοδυναμεί με:

a > b ? (meg = a):(meg = b); ή (a > b) ? (meg = a):(meg = b); ή meg =(a > b) ? (a):(b)

Υπενθυμίζουμε ότι η <συνθηκη> μπορεί να περιλαμβάνει οποιουσδήποτε σχεσιακούς και λογικούς τελεστές και όλους τους δυνατούς συνδυασμούς τους.

[55] Το = εκτελεί μια **απλή ανάθεση**, π.χ. a=100.

Το = εκτελεί **πολλαπλές αναθέσεις**, π.χ. a=b=c=d=e=100. Επειδή η ανάθεση συσχετίζεται από δεξιά η έκφραση στο παράδειγμα αναθέτει 100 στη μεταβλητή e. Αυτή η ανάθεση παράγει μια τιμή, 100, για την έκφραση, η οποία διαδοχικά ανατίθεται στην a. Έτσι οι τιμές των μεταβλητών μετά από το τέλος της πρότασης είναι πλεόν 100. Η πολλαπλή ανάθεση χρησιμοποιείται και για τον ορισμό (ανάθεση) αρχικής τιμής ίσο με μηδέν σε κάποιες μεταβλητές, στην αρχή κάποιας διαδικασίας, αν χρειάζεται.

[56] Οι σύνθετοι αυτοί τελεστές μπορούν να εφαρμόζονται οποτεδήποτε εμφανίζεται η ίδια ματαβλητή και στα δύο μέρη του σημείου "=". Οι τελεστές αυτοί υπολογίζονται σχεδόν τελευταίοι αφού βρίσκονται χαμηλά στον πίνακα προτεραίοτητας.

*=	υπολογισμός και ανάθεση γινομένου
/=	υπολογισμός και ανάθεση πηλίκου
%=	υπολογισμός και ανάθεση υπολοίπου
+=	υπολογισμός και ανάθεση αθροίσματος
-=	υπολογισμός και ανάθεση διαφοράς
&=	υπολογισμός και ανάθεση δυαδικού AND
^=	υπολογισμός και ανάθεση δυαδικού XOR
|=	υπολογισμός και ανάθεση δυαδικού OR
<<=	υπολογισμός και ανάθεση αριστερής μετατόπισης
>>=	υπολογισμός και ανάθεση δεξιάς μετατόπισης

επίπεδο προτεραιότητας 16

, κόμμα[27] ο συσχετισμός πραγματοποιείται από **αριστερά προς τα δεξιά**

4.4.4 Ασκήσεις (Ερωτήσεις για Επανάληψη και Εμπέδωση)

Α. Αριθμητικοί Τελεστές

1. Ποιό είναι το αποτέλεσμα για κάθε μια από τις ακόλουθες εκφράσεις;
   (α) 1 + 2 * 4 / 2 (β) (1 + 2) * 4 / 2 (γ) 1 + 2 * (4 / 2)

2. Ποιό είναι το αποτέλεσμα για κάθε μια από τις ακόλουθες εκφράσεις;
   (α) 9 % 2 + 1 (β) (1 + (10 - (2 + 2)))

3. Μετατρέψτε τους παρακάτω "τύπους" στις ισοδύναμες "προτάσεις ανάθεσης" της C++:

(α) $a = \frac{3 + 3}{4 + 4}$

(β) $x = (a - b) * (a - c)^2$

(γ) $i = \frac{a^2}{b^3}$

(δ) $d = \frac{(8 - x^2)}{(x - 9)} * \frac{(4\ 2 - 1)}{x^3}$

4. Γράψτε ένα σύντομο πρόγραμμα που εκτυπώνει το εμβαδό κύκλου όταν η ακτίνα του (r)

---

[27] Το κόμμα δεν χρησιμοποιείται μόνο (σαν οριοθέτης) για να ξεχωρίσει μεταξύ τους προσδιοριστές σε μια δήλωση, π.χ. int a, b, c, d. Χρησιμοποιείται και σαν τελεστής για να ξεχωρίσει μεταξύ τους παραστάσεις, από τις οποίες μόνο η τελευταία είναι ουσιαστική, π.χ. η πρόταση x = (a++, b++, c++); αυξάνει τις τιμές των a, b και c κατά 1, αλλά στη x καταχωρεί την τιμή της c. Ο τελεστής αυτός δεν ενεργεί απευθείας στα δεδομένα.

είναι 4 απ. (Υπόδειξη: $E_κ = πr^2$, όπου π = 3.14159).

5. Γράψτε μια ανάθεση και εκτυπώστε το υπόλοιπο του 100/3.

Β. Σχεσιακοί Τελεστές

6. Ποιοί είναι οι σχεσιακοί τελεστές;

7. Ποιός τελεστής ελέγχει την ισότητα;

8. Ποιές από τις παρακάτω σχέσεις ισχύουν και ποιές δεν ισχύουν;

(α) 4 >= 5 (β) 4 == 4 (γ) 165 >= 165 (δ) 0 != 67

Γ. Λογικοί Τελεστές

9. Ποιοί είναι οι τρείς λογικοί τελεστές;

10. Οι παρακάτω έλεγχοι παράγουν αποτελέσματα που είναι "αληθή" ή "ψευδή". Βρείτε ποιοί από τους ελέγχους είναι "αληθείς" ή "ψευδείς".

(α) ! (True || False)

(β) (True && False) && (true || false)

(γ) || (True && False)

(δ) True || (False && False) || False

11. Έστω η πρόταση: **int** i = 12, j = 10, k = 5; Είναι "αληθείς" ή "ψευδείς" οι παρακάτω προτάσεις; (Υπόδειξη: η C++ ερμηνεύει τις μη μηδενικές προτάσεις σαν "αληθείς").

(α) i && j (β) 12 - i || k (γ) j != k && i != k .

12. Χρησιμοποιώντας τον πίνακα προτεραιότητας, καθορίστε αν οι παρακάτω προτάσεις παράγουν "αληθή" ή "ψευδή" αποτελέσματα. Μετά από αυτό θα εκτιμήσετε την αξία των παρενθέσεων!

(α) 5 == 4 + 1 || 7 * 2 != 12 - 1 && 5 == 8 / 2

(β) 8 + 9 != 6 - 1 || 10 % 2 != 5 + 0

(γ) 17 - 1 > 15+ 1 && 0 + 2 != 1 == 1 || 4 != 1

(δ) 409 * 0 != 1 * 409 + 0 || 1 + 8 * 2 >= 17

Δ. Τελεστές Συνθήκης, Προσαύξησης, Μείωσης, sizeof και Ψηφιακοί ή Δυαδικών

13. Ποιές προτάσεις αντικαθιστά ο τελεστής συνθήκης;

14. Γιατί ο τελεστής συνθήκης καλείται "τριπλός" τελεστής;

15. Είναι αληθές ή ψευδές ότι οι προτάσεις i++; και i = i + 1; παράγουν το ίδιο αποτέλεσμα;

16. Γιατί η χρήση των τελεστών προσαύξησης και μείωσης είναι αποδοτικότερη από τη χρήση των τελεστών πρόσθεσης και αφαίρεσης;

17. Τι είναι ένα σημείο ακολουθίας;

18. Μπορεί να καθοριστεί η έξοδος από το παρακάτω τμήμα κώδικα;

```
likia = 20;
cout << "Φέτος είσαι " << likia << "χρονών και του χρόνου θα είσαι " << likia++
        << "χρονών . ";
```

19. Ποιά είναι η έξοδος από το παρακάτω τμήμα κώδικα;

```
char onoma[20] = "Telis";
cout << " Το μήκος του ονόματος είναι : " << sizeof(onoma) << "\n";
```

20. Ποιό είναι το αποτέλεσμα των παρακάτω εκφράσεων;

(α) 1 ^ 0 & 1& 1| 0
(β) 1 & 1 & 1 & 1
(γ) 1 ^ 1 ^ 1 ^ 1
(δ) ~ (1 ^ 0)

4.5 Δεικτοφόρες Μεταβλητές και Παραπάξεις

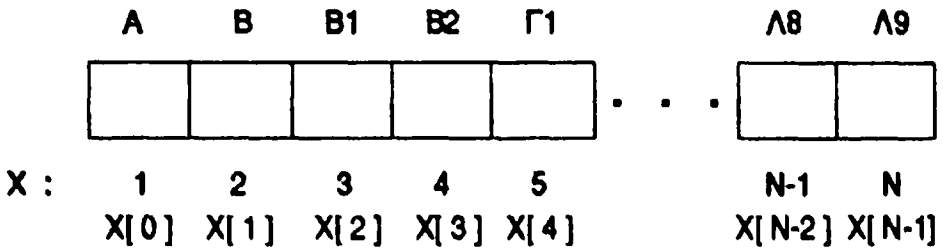
Στο κεφάλαιο 2 εξετάσαμε ορισμένα προβλήματα τα οποία απαιτούσαν τον χειρισμό πολλών δεδομένων, όπως π.χ., καταμέτρησης, αθροίσματος και υπολογισμού του μέσου όρου πολλών αριθμών, προσδιορισμού του αριθμού που έχει τη μέγιστη τιμή, κλπ. Για τη λύση αυτών των προβλημάτων δεν ήταν απαραίτητη η παρουσία όλων των δεδομένων στη μνήμη του Η/Υ, δηλαδή των δεδομένων που αντιστοιχούσαν στους δοσμένους για επεξεργασία (του ιδίου τύπου και λογικά συνδεδεμένους) αριθμούς. Υπάρχουν όμως προβλήματα, τα οποία είτε ζητούν την αποθήκευση και εκτύπωση (αυτών των) πολλών δεδομένων, είτε χρειάζονται την παρουσία όλων (αυτών) των δεδομένων για τη λύση.

Είναι γνωστό ότι πληροφορίες (δεδομένα) μπορούν να είναι προσιτές στον προγραμματιστή ανά πάσα στιγμή, μόνο εάν είναι εναποθηκευμένες στη μνήμη του Η/Υ, δηλαδή μόνο εάν κάθε λέξη μνήμης είναι συσχετισμένη με ένα συμβολικό όνομα, που δεν είναι άλλο από μια μεταβλητή της γλώσσας προγραμματισμού.

Η χρήση, όμως, πολλών διαφορετικών ονομάτων (προσδιοριστών), για του ιδίου τύπου (λογικά συνδεδεμένα) πολλά δεδομένα, δεν είναι καθόλου πρακτική, διότι ένα τμήμα

προγράμματος πρέπει να επαναλαμβάνεται πολλές φορές, ώστε οι ίδιες πράξεις να μπορούν να γίνουν με όλα τα δεδομένα.

Η παραπάνω δυσκολία παρακάμπτεται με την εισαγωγή των **δεικτοφόρων μεταβλητών** (indexed variables). Αντί να συσχετίσουμε πολλά ονόματα με τις επιμέρους λέξεις μιας περιοχής μνήμης, συσχετίζουμε ένα όνομα με ολόκληρη την περιοχή και τις επιμέρους λέξεις τις διακρίνουμε με **δείκτες** (index) (βλπ Σχήμα 4.5.1).



ΣΧΗΜΑ 4.5.1: Ένα όνομα X (όνομα παράταξης) το συσχετίζομε με ολόκληρη την περιοχή μνήμης και τις επιμέρους λέξεις (στοιχεία της παράταξης) τις διακρίνουμε με δείκτες, έτσι συμβολίζουμε τα N στοιχεία της παράταξης με δεικτοφόρες μεταβλητές, X[0], X[1], . . . , X[N-1] · αντί να συσχετίζουμε τις επιμέρους λέξεις μιας περιοχής με πολλά ονόματα A, B, B1, . . . , Λ9.

Έτσι έχουμε μια **παράταξη** (array), μια ακολουθία δηλαδή N (λογικά συνδεδεμένων) δεδομένων τα οποία προσδιορίζονται με ένα όνομα π.χ. ARITHM, ενώ καταχωρούνται (αποθηκεύονται) στις δεικτοφόρες μεταβλητές ARITHM[0], ARITHM[1], ARITHM[2], . . . , ARITHM[N-1]. Η παράταξη αυτή είναι μονοδιάστατη. Οι **μονοδιάστατες παρατάξεις** στην C++ δηλώνονται ως εξής:

<τυπος_παραταξης><προσδιοριστής> [<μηκος_παραταξης>];

όπου:

<τυπος_παραταξης> ::= **int** | **float** | **char**[59] κλπ. (βλπ Πίνακα 4.3.1),

<μηκος_παραταξης> ::= <πληθος_στοιχειων_παραταξης>

<πληθος_στοιχειων_παραταξης> ::= N, όπου N = {1. 2, 3, 4, 5, 6, . . . }

π.χ. **int** ARITHM[9]; // ακέραια παράταξη μήκους 10 ή πλήθους 10 στοιχείων

Υπάρχει μια δυνατότητα που μπορεί να εμφανιστεί συχγρόνως με τη δήλωση της

[59] Τις μεταβλητές τύπου **char** (παρατάξεις χαρακτήρων) θα τις εξετάσουμε ξεχωριστά και διεξοδικά στην επόμενη παράγραφο 4.5.1, στην παράγραφο αυτή ασχολούμαστε αποκλειστικά με αριθμητικού τύπου δεδομένα.

παράταξης, οποιουδήποτε τύπου. Είναι λοιπόν δυνατό να **ορισθούν στη δήλωση της παράταξης και οι αρχικές τιμές της**, π.χ.:

1. Να οριστούν οι αρχικές τιμές μιας ακεραίας παράταξης που περιέχει τις ηλικίες 5 παιδιών,

```
int child_age [5] = {2, 5, 6, 8, 12}; // δήλωση και ορισμός αρχικών τιμών παράταξης
```

2. Να οριστούν οι αρχικές τιμές μιας διπλής ακρίβειας παράταξης που περιέχει τα δισλότυπα (ίχνη) των πωλήσεων των τριών τελευταίων ετών,

```
double sales[ ] = {454323.43, 122355.32, 333424.96};
```

Προσοχή! πάντοτε στις δηλώσεις παραστάξεων δηλώνουμε τον αριθμό των δεικτών και μάλιστα τον μέγιστο, άλλως οι δηλώσεις μας είναι λάθος (`double sales[ ]; // λάθος`). Εξαίρεση γίνεται και δεν δηλώνουμε τον αριθμό των δεικτών, μόνο όταν γίνεται ταυτόχρονα ορισμός αρχικών τιμών, διότι η C++ αποφασίζει από τον κατάλογο των τιμών πόσες θέσεις θα δεσμεύσει, στην περίπτωση του παραδείγματός μας δεσμεύει τρείς θέσεις.

Παρατήρηση: Η C++ δεν ορίζει αυτόματα αρχικές τιμές στα στοιχεία μιας παράταξης. Είναι όμως δυνατόν εάν με τη δήλωση της παράταξης οριστούν αρχικές τιμές για κάποια, όχι για όλα τα στοιχεία της, τότε αυτόματα στα υπόλοιπα ανατίθεται το 0 (μηδέν) ως αρχική τιμή. Από την παρατήρηση αυτή συνάγεται ότι: για τον **ορισμό αρχικής τιμής 0 (μηδέν) στα στοιχεία μιας μεγάλης παράταξης**, αρκεί με τη δήλωση της παράταξης να ανατεθεί το 0 (μηδέν) στο πρώτο μόνο στοιχείο της παράταξης, στα υπόλοιπα στοιχεία θα ανατεθεί αυτόματα το 0(μηδέν) ως αρχική τιμή.

Μια παράταξη με περισσότερους από ένα δείκτες λέγεται **πολυδιάστατη παράταξη**. Ανάλογα λοιπόν έχουμε τις διοδιάστατες, τρισδιάστατες, τετρασδιάστατες κλπ. παραστάξεις.

Σημειώνουμε ότι:

1. Σε μια διοδιάστατη παράταξη ο πρώτος δείκτης δείχνει τον αριθμό (πλήθος) των γραμμών και ο δεύτερος τον αριθμό (πλήθος) των στηλών, όπως ακριβώς συμβαίνει με τις μήτρες (πίνακες - matrix, matrices) στα Μαθηματικά.
2. Σε μια τρισδιάστατη παράταξη ο πρώτος δείκτης δηλώνει το βάθος, ο δεύτερος το πλήθος των γραμμών και ο τρίτος το πλήθος των στηλών.
3. Σε μια πολυδιάστατη γενικά παράταξη ο τελευταίος δείκτης δηλώνει το πλήθος των στηλών, ενώ ο πρότελευταιος το πλήθος των γραμμών κλπ.

4. Σπάνια τα πραγματικά δεδομένα απαιτούν περισσότερες από δύο ή τρείς διαστάσεις.

Παραδείγματα 4.5: Παραθέτουμε παραδείγματα με δηλώσεις δισδιάστατων, τρισδιάστατων και τετρασδιάστατων αντίστοιχα παρατάξεων:

1. Δήλωση μιας δισδιάστατης παράταξης ακεραίων που ονομάζεται teams με 15 γραμμές και 10 στήλες,

```
int teams[15] [10]; // δεσμεύει μια δισδιάστατη παράταξη 150 ακεραίων στοιχείων
```

Ας υποθέσουμε ότι η παραπάνω παράταξη αποθηκεύει τους πόντους των παικτών μιας ομάδας (teams) υδατοσφαίρησης σε κάθε αγώνα. Η ομάδα έπαιξε 10 παιχνίδια και έχει 15 παίχτες.

```
int teams[4] [15] [10]; // δεσμεύει μια τρισδιάστατη παράταξη 600 ακεραίων στοιχείων
```

Η παραπάνω παράταξη αποθηκεύει τους πόντους των παικτών τεσσάρων (4) ομάδων (teams) υδατοσφαίρησης σε κάθε αγώνα. Η ομάδα έπαιξε 10 παιχνίδια και έχει 15 παίχτες. Κάθε ομάδα, το βάθος της παράταξης, έχει γραμμές και στήλες με δεδομένα τους πόντους, δηλαδή έχει 15 γραμμές τους 15 παίχτες και 10 στήλες τα 10 παιχνίδια για κάθε μια από τις 4 ομάδες, 4 το βάθος της παράταξης. Αν υπάχουν περισσότερες από μια κατηγορίες στο πρωτάθλημα, τότε είναι άλλη μια διάσταση, άλλο ένα σύνολο δεδομένων.

2. Δήλωση μιας τρισδιάστατης παράταξης αριθμών κινητού δεκαδικού σημείου (κ.δ.σ.) που ονομάζεται utilities με βάθος 5, 12 γραμμές και 4 στήλες,

```
float utilities [5] [12] [4]; // δεσμεύει μια τρισδιάστατη παράταξη 240 στοιχείων κ.δ.σ.
```

Ας υποθέσουμε ότι η παραπάνω τρισδιάστατη παράταξη αποθηκεύει τα ίχνη των λογαριασμών, 4 χρήσεων, των 5 τελευταίων ετών. Η πρώτη από αριστερά διάσταση, το βάθος, είναι των 5 χρόνων, η δεύτερη, οι γραμμές, των 12 μηνών (12 μήνες το έτος) και η τρίτη, οι στήλες, των 4 ανεξάρτητων χρήσεων. Τα στοιχεία που δεσμεύει είναι 240 στοιχείωνκ.δ.σ.

3. Δήλωση μιας τετρασδιάστατης παράταξης αριθμών κινητού δεκαδικού σημείου (κ.δ.σ.) που ονομάζεται sales με 2 στήλες, 7 γραμμές, κλπ.,

```
float sales[3] [4] [7] [2]; // δεσμεύει μια τετρασδιάστατη παράταξη 168 στοιχείων κ.δ.σ.
```

4. Ο ορισμός των αρχικών τιμών σε πολυδιάστατες παρατάξεις, συγχρόνως με τις δηλώσεις τους, γίνεται ως εξής:

```
int multidim1 [2] [4] = {4, 3, 2, 1,1, 2, 3, 4}; ή
int multidim1 [2] [4] = {{4, 3, 2, 1}, {1, 2, 3, 4}};

int multidim2 [2] [3] [4]={4,3,2,1,1,2,3,4,5,6,7,8,8,7,6,5,3,4,2,1,2,4,1,3};
int multidim2 [2] [3] [4]={ { {4,3,2,1},{1,2,3,4},{5,6,7,8} },{ {8,7,6,5},{3,4,2,1},{2,4,1,3} } };
```

Δεν υπάρχει διαφορά στον ορισμό αρχικών τιμών με ή χωρίς τις ένθετες αγγύλες. Οι ένθετες αγγύλες δείχνουν καλλίτερα τις διαστάσεις και τον τρόπο που ανατίθενται οι τιμές από τη γλώσσα. Προτρέπουμε να χρησιμοποιούνται.

5. Για να ανατεθεί το 0 (μηδέν) σαν αρχική τιμή σε μια πολυδιάστατη παράταξη, π.χ. στην τετρασδιάστατη πραγματική παράταξη **sales[3] [4] [7] [2]** , θα πρέπει αυτή να δηλωθεί ως εξής:

```
float sales[3] [4] [7] [2] = {0}; // αναθέτει το μηδέν σε όλα τα στοιχεία
```

Παρατηρήσεις 4.5:

1. Όλοι οι δείκτες των παρατάξεων αρχίζουν με 0.

2. Όταν ορίζεται μια παράταξη σε ένα πρόγραμμα, πάντοτε ορίζεται με τον μέγιστο αριθμό στοιχείων.

3. Κάθε στοιχείο σε μια παράταξη καταλαμβάνει την ίδια αποθηκευτική ποσότητα με μια μεταβλητή εκτός της παράταξης του ιδίου τύπου δεδομένων. Για παράδειγμα κάθε στοιχείο μιας παράταξης χαρακτήρων καταλαμβάνει ένα byte. Ενώ κάθε στοιχείο μιας παράταξης ακεραίων καταλαμβάνει δύο ή περισσότερα bytes, ανάλογα με την εσωτερική αρχιτεκτονική του Η/Υ.

4. Τα στοιχεία της παράταξης ακολουθούν το ένα το άλλο στη μνήμη, χωρίς να υπάρχει κενό ή οτιδήποτε ανάμεσά τους. Αυτό συμβαίνει στις παρατάξεις χαρακτήρων, ακεραίων, κ.δ.σ. και σε οποιονδήποτε άλλο τύπο παράταξης. Αν για παράδειγμα μια τιμή κ.δ.σ. καταλαμβάνει 4 bytes μνήμης στον Η/Υ, το επόμενο στοιχείο της παράταξης θα αρχίζει 4 bytes ακριβώς μετά από το προηγούμενο στοιχείο. Ανάλογα η C++ αποθηκεύει πολυδιάστατες παρατάξεις σε ακολουθία γραμμών.

5. Το μέγεθος της παράταξης, δηλαδή το πλήθος των bytes που δεσμεύτηκαν για την

παράταξη υπολογίζεται από τη συνάρτηση **sizeof(**<προσδιοριστης_παραταξης>**)**.

4.5.1 Παρατάξεις Χαρακτήρων και Αλυσίδες Χαρακτήρων

Όπως αναφέραμε και στην § 4.3.1.4 δεν υφίστανται μεταβλητές που να μπορούν να αποθηκεύσουν αλυσίδες χαρακτήρων, δηλαδή δεν υπάρχουν μεταβλητές του τύπου **αλυσίδας χαρακτήρων** (strings). Στην περίπτωση όμως χειρισμού αλυσίδων χαρακτήρων πρέπει να χρησιμοποιούμε παρατάξεις χαρακτήρων.

Οι παρατάξεις χαρακτήρων δηλώνονται ανάλογα με τις παρατάξεις που αναφέρονται σε αριθμητικά δεδομένα:

char <προσδιοριστης_παραταξης> **[**<μηκος_παραταξης>**];**

Μια αλυσίδα χαρακτήρων μπορεί να θεωρηθεί σαν μια ακολουθία ενός ή περισσοτέρων χαρακτήρων, έτσι είναι δυνατόν να ανατεθεί ως τιμή μιας παράταξης χαρακτήρων. Για παράδειγμα, η ηλικία, ο μισθός και το ονοματεπώνυμο κάποιου, ως δεδομένα σε κάποιο πρόγραμμα, θα μπορούσαν να παρασταθούνως εξής:

```
int age;
float salary;
char name[15];
```

Με την τελευταία δήλωση δημιουργήθηκε μια παράταξη χαρακτήρων, δηλαδή μια ή περισσότερες μεταβλητές χαρακτήρων σε μια γραμμή της μνήμης. Η δήλωση μιας παράταξης χαρακτήρων περιέχει, όπως και στις αριθμητικές παρατάξεις, τις αγγύλες που δηλώνουν τον χώρο της παράταξης. Η παράταξη αυτή με όνομα name έχει μέγεθος 15 χαρακτήρων.

Είναι δυνατόν να αναθέσουμε την τιμή στην παράταξη, όπως και στις αριθμητικές παρατάξεις, συγχρόνως με τη δήλωση της παράταξης, δηλαδή:

```
char name[15] = "Georgios Prasinos";
```

Σε κάθε στοιχείο της παράταξης ανατίθεται και ένας χαρακτήρας της δοθείσας αλυσίδας χαρακτήρων.

Τα χαρακτηριστικά της καταχώρησης αλυσίδας χαρακτήρων σε παράταξη χαρακτήρων είναι: **(α)** η αλυσίδα χαρακτήρων ανατίθεται εντός ζεύγους διπλών εισαγωγικών και **(β)** πάντοτε ως τελευταίος χαρακτήρας στην παράταξη ανατίθεται εξ ορισμού το στοιχείο **\0**, το οποίο λειτουργεί ως χαρακτήρας τερματισμού της αλυσίδας

χαρακτήρων.

Στο παράδειγμά μας, στην παράταξή μας καταχωρήθηκε το ονοματεπώνυμο στα 13 πρώτα στοιχεία, name[1] έως name[13] , όσοι δηλαδή είναι οι αλφαβητικοί χαρακτήρες αυτού και το κενό αναμεσα στο όνομα και το επώνυμο. Στο στοιχείο name[14] ανατέθηκε εξ ορισμού το \0, ενώ το στοιχείο A[15] παρέμεινε άδειο.

Προσοχή! Αν αναθέτετε μια τιμή σε μια παράταξη χαρακτήρων συγχρόνως με τη δήλωση της παράταξης, η C++ μετρά το μήκος της αλυσίδας χαρακτήρων, προσθέτει σε αυτό τη θέση του \0 και δεσμεύει τον χώρο για την παράταξη.

Μια άλλη δυνατότητα που παρέχεται από τις παρατάξεις στον χειρισμό αλυσίδων χαρακτήρων είναι η προσπέλαση σε μεμονωμένα στοιχεία της παράταξης ή και σε όλη την παράταξη σαν μια οντότητα. Η προσπέλαση σε μεμονωμένα στοιχεία υλοποιείται εκλέγοντας τον δείκτη του στοιχείου που θέλουμε να προσπελάσουμε, π.χ.:

```
name[3] = 'w';
```

Έτσι αντικαθίσταται ο χαρακτήρας ο από τον χαρακτήρα w στο Georgios, που γίνεται Gewrgios.

Η εκτύπωση της αλυσίδας χαρκτήρων ή καλλίτερα της παράταξης name[15] πραγματοποιείται με τον cout ως εξής:

```
cout << name;
```

χωρίς αγγύλες.

Όπως και στις αριθμητικές παρατάξεις, έτσι και στις παρατάξεις χαρακτήρων ισχύει:

```
char name[14] = "Georgios Prasinos";
char name[ ]  = "Georgios Prasinos";
```

Οι αλυσίδες χαρακτήρων πρέπει να αποθηκεύονται σε παρατάξεις αλυσίδων, αλλά όλες οι παρατάξεις αλυσίδων δεν περιέχουν αλυσίδες χαρακτήρων.

Παραδείγματα: Γίνεται σαφής διάκριση μεταξύ των δύο παρατάξεων χαρακτήρων, parataxi1[10] και parataxi2[10], που έπονται:

1. Η πρώτη parataxi1[10] δεν περιέχει αλυσίδα χαρακτήρων, απλώς είναι μια ακολουθία

χαρακτήρων τα στοιχεία της, η δήλωση του parataxi1[10] γίνεται με 10 ανεξάρτητους χαρακτήρες, δηλαδή:

```
char parataxi1[10] = {'a', 'b', 'c', 'd', 'e','f', 'g', 'h', 'i', 'j'}; ή

char parataxi1[10];
parataxi1[0] = 'a';
parataxi1[1] = 'b';
parataxi1[2] = 'c';
parataxi1[3] = 'd';
parataxi1[4] = 'e';
parataxi1[5] = 'f';
parataxi1[6] = 'g';
parataxi1[7] = 'h';
parataxi1[8] = 'i';
parataxi1[9] = 'j'; // τελευταίο στοιχείο με δείκτη 9
```

Η παράταξη χαρακτήρων parataxi1[10] δεν περιέχει το τερματικό \0, άρα δεν περιέχει αλυσίδα χαρκτήρων. Η παράταξη αυτή περιέχει χαρακτήρες που μπορούν σε ένα πρόγραμμα να να χρησιμοποιηθούν ανεξάρτητα, δεν μπορούν όμως να χρησιμοποιηθούν σαν αλυσίδα χαρακτήρων.

2. Η δήλωση της δεύτερης παράταξης χαρακτήρων parataxi2[10] γίνεται με την αλυσίδα χαρακτήρων "excellent", δηλαδή:

```
char parataxi1[10] = "excellent ";
```

Δεν μπορεί να ανατεθούν τιμές αλυσίδας χαρακτήρων σε παράταξη χαρκτήρων με μια κανονική πρόταση ανάθεσης, εκτός και αν δηλωθούν πρώτα σαν παρατάξεις χαρακτήρων.

Παραδείγματα:

1.
```
. . .
#include< iostream.h>
main( )
{
char name[20];     // δήλωση για παράταξη χαρακτήρων
name = "Alfabita"; // λάθος ανάθεση
cout << name;  // πρόγραμμα δεν φθάνει ποτέ στο σημείο αυτό λόγω του λάθους
return(0);
}
```

2.
```
. . .
#include< iostream.h>
main( )
```

```
{
char name[20];      // δήλωση για παράταξη χαρακτήρων
name[0] = 'A';      // ανάθεση τιμών στοιχείο προς στοιχείο
name[1] = 'l';
name[2] = 'l';
name[3] = 'a';
name[4] = 'b';
name[5] = 'i';
name[6] = 'l';
name[7] = 'a';
name[8] = '\0';     // χρειάζεται για να είναι αλυσίδα χαρακτήρων
cout << name;       // το όνομα εκτυπώνεται σωστά
return(0);
}
```

Η παράταξη χαρακτήρων περιέχει μια αλυσίδα χαρακτήρων, επειδή ο τελευταίος χαρακτήρας είανια το \0. Το μήκος της αλυσίδας χαρακτήρων είναι 8, διότι ποτέ δεν περιλαμβάνεται το \0 στο μήκος της.

Υπάρχουν και άλλοι πολλοί τρόποι ανάθεσης τιμής σε αλυσίδα χαρακτήρων. για παράδειγμα με τη χρησιμοποίηση της συνάρτησης:

strcpy(<προσδιοριστης_παραταξης>, "<περιγραφική_αλυσιδα_χαρακτηρων>")

Αυτή είναι μια ενσωματωμένη συνάρτηση που επιτρέπει την αντιγραφή μιας περιγραφικής αλυσίδας χαρακτήρων σε μια αλυσίδα χαρακτήρων.

Για να αντιγραφεί η περιγραφική αλυσίδα χαρακτήρων "Allabila" στην παράταξη χαρακτήρων name, πρέπει να πληκτρολογηθεί:

```
strcpy(name, "Allabila");   // αντιγράφει το Allabila στην παράταξη
```

Προσοχή στο μέγεθος της παράταξης χαρακτήρων, να έχει τόσο μέγεθος ώστε να μπορεί να αντιγραφεί η αλυσίδα χαρακτήρων σε αυτή.

Σε προγράμματα που αναφέρεται η συνάρτηση:

strcpy(<προσδιοριστης_παραταξης>, "<περιγραφική_αλυσιδα_χαρακτηρων>")

ή οποιαδήποτε άλλη συνάρτηση που χειρίζεται αλυσίδες χαρακτήρων, θα πρέπει να περιέχεται στο πρόγραμμα η οδηγία:

```
#include <string.h>
```

πριν τη συναρτηση main( ) αλλά μετά την οδηγία #include< iostream.h>, ο λόγος είναι ότι ο μεταγλωττιστής παρέχει το αρχείο string.h για να διευκολύνει την συνάρτηση

strcpy( ) να λειτουργήσει κανονικά. Έτσι έχουμε:

```
. . .
#include< iostream.h>
#include <string.h >
main( )
{
 . . .
 strcpy( );
 . . .
 return(0);
}
```

4.5.2 Ασκήσεις (Ερωτήσεις για Επανάληψη και Εμπέδωση)

1. Είναι αληθές ή ψευδές ότι, μια παράταξη μπορεί να περιέχει τιμές διαφορετικών τύπων δεδομένων;

2. Σε ένα πρόγραμμα C++, πώς ένα στοιχείο παράταξης διακρίνεται από ένα άλλο στοιχείο που έχει ίδιο όνομα;

3. Γιατί θα πρέπει να ορίζουμε αρχικές τιμές στα στοιχεία μιας παράταξης πριν την χρησιμοποίησή της;

4. Με δεδομένο τον ορισμό: **int** varos[10] = { 5, 2, 4 }; της παράταξης varos[10], ποιά είναι η τιμή του στοιχείου varos[5] ;

5. Πώς θα δηλώνατε μια παράταξη χαρακτήρων που προσδιορίζεται με τον προσδιοριστή "onoma", στην οποία καταχωρείται η περιγραφική αλυσίδα χαρακτήρων: " Αυτή είναι η C++";

6. Ποιό είναι το μήκος της αλυσίδας χαρακτήρων της προηγούμενης άσκησης;

7. Με τι τελειώνουν όλες οι περιγρφικές αλυσίδες χαρακτήρων;

8. Πόσες μεταβλητές δηλώνουν οι παρακάτω προτάσεις και ποιοί είναι οι τύποι τους;
   (α) **char** onoma[25]; (β) **char** dieidinsi [25];

9. Είναι αληθές ή ψευδές ότι, η παρακάτω πρόταση αναθέτει μια περιγραφική αλυσίδα χαρακτήρων σε μια παράταξη χαρακτήρων;
   onomateponymo[ ] = "Polihronis Polihronos";

10. Είναι αληθές ή ψευδές ότι, η παρακάτω δήλωση αναθέτει μια αλυσίδα χαρακτήρων στην παράταξη χαρακτήρων με όνομα vrahonisides;

```
char vrahonisides[ ] = { 'Γ', 'Μ', 'Γ', 'Α', '\0' };
```

11. Είναι αληθές ή ψευδές ότι, η παρακάτω δήλωση αναθέτει μια αλυσίδα χαρακτήρων στην παράταξη χαρακτήρων με όνομα vrahonisides;

```
char vrahonisides[ ] = { 'Γ', 'Μ', 'Γ', 'Α' };
```

12. Είναι αληθές ή ψευδές ότι, θα πρέπει να ανφερόμαστε σε μια παράταξη με την ίδια σειρά με την οποία ορίσαμε αρχικές τιμές;

13. Πως χρησιμοποιεί η C++ το όνομα ενός πίνακα;

14. Δίδεται η δήλωση παράταξης:

```
char omades[ ] = { 'Έ', 'α', 'γ', 'ι', 'ο', 's', '\0', 'Ή', 'α', 'μ', 's', '\0' };
```

Τι θα τυπωθεί σε καθεμία από τις παρακάτω προτάσεις; Απαντήστε άκυρο εάν ο cout είναι λάθος.

(α) `cout << omades;` (β) `cout << omades + 7;` (γ) `cout << ( omades + 3);`

(δ) `cout << omades[0];` (ε) `cout << (omades + 0)[0];` (ς) `cout << ( omades + 5);`

15. Πώς δηλώνεται μια δισδιάστατη ακεραία παράταξη που ονομάζεται "epidoseis" και έχει πέντε γραμμές και έξι στήλες;

16. Πώς δηλώνεται μια τρισδιάστατη παράταξη τεσσάρων παρατάξεων αλυσίδων που ονομάζεται "arxika" και είναι δέκα γραμμών και είκοσι στηλών;

17. Στην παρακάτω δήλωση ποιός δείκτης αντιστοιχεί στις γραμμές και ποιός στις στήλες;

```
int varos[5][10];
```

18. Πόσα στοιχεία δεσμεύονται με την ακόλουθη δήλωση;

```
int dialexi[5][5];
```

19. Ποιά πρόταση ελέγχου είναι η κατάλληλη για χειρισμό πολυδιάστατων παρατάξεων;

20. Δίδεται το ακόλουθο τμήμα προγράμματος:

```
int vathmoi[3][5] = { 80, 90, 96, 73, 65, 67, 90, 68, 92, 84, 70, 55, 95, 78, 100};
```

Ποιές είναι οι τιμές των ακολούθων;

(α) `vathmoi[2][3]` (β) `vathmoi[2][4]` (γ) `vathmoi[0][1]`

21. Η ακόλουθη δισδιάστατη παράταξη ονομάζεται "dialexi" :

4	1	3	5	9
10	2	12	1	6
25	42	2	91	8

Ποιές τιμές περιέχουν τα ακόλουθα στοιχεία;

(α) dialexi[2][2] (β) dialexi[0][1] (γ) dialexi[2][3] (δ) dialexi[2][4]

4.6 Εκτελέσιμες και μη Προτάσεις

4.6.1 Προτάσεις Ελέγχου Ροής Υπολογισμών

4.6.1.1 Η Πρόταση if

Η πρόταση αυτή ελέγχει μια ή περισσότερες σχέσεις, με τους σχεσιακούς τελεστές, και αποφασίζει ή εκλέγει ποιά πρόταση θα εκτελεστεί στη συνέχεια. Για τον λόγο αυτόν η πρόταση **if** λέγεται **πρόταση απόφασης** ή **εκλογής** ή **υποθετική** (conditional statement).

Η σύνταξη της **if** ακολουθεί τους εξής κανόνες οι οποίοι διατυπώνονται με τη βοήθεια της BNF:

1. <προταση_if> ::= if(<υποθεση>)
   {<αποδοση>} ,

2. <υποθεση> ::= <λογικη_εκφραση>,

3. <λογικη_εκφραση> ::= <απλη_εκφραση><σχεσιακος_τελεστης><απλη_εκφραση> |
   <λογικη_εκφραση><λογικος_τελεστης><λογικη_εκφραση> ,

4. <απλη_εκφραση> ::= <προσημο><οροs> |
   <προσημο><οροs><απλη_εκφραση> |
   <προσημο><οροs> || <απλη_εκφραση> ,

5. <προσημο> ::= + | - ,

6. <οροs> ::= <παραγοντας> |
   <παραγοντας><αριθμητικος_τελεστης><οροs> |

<παραγοντας> && <ορος>,

7. <παραγοντας> ::= <ακεραιος> | <πραγματικος> | <χαρακτηρας>
<προσδιοριστης> | ... | <προσδιοριστης_συναρτησης> | ...,

8. <αριθμητικος_τελεστης> ::= + | - | * | / | % ,

9. <σχεσιακος_τελεστης> ::= == | != | > | >= | < | <= ,

10. <λογικος_τελεστης> ::= && | || | ! ,

11. <αποδοση> ::= <συνθετη_προταση> ,

12. <συνθετη_προταση> ::= <προταση>; |
<προταση>;<συνθετη_προταση>,

13. <προταση> ::= <αναθεση> |
<αναγνωση _εισοδου> | <γραφη_εξοδου> |
<ανακυκλωση> |
<υποθετικη> |
<εκτροπη> |
<πολλαπλη_επιλογη> |
<κληση_υποπρογραμματος> (δηλαδή <κληση_συναρτησης>) .

Σημείωση: Αν η <υποθεση> είναι **αληθής** (true) τότε η <αποδοση> εκτελείται. Αν η <υποθεση> είναι **ψευδής** (false), η C++ αγνοεί την <αποδοση> και εκτελεί την επόμενη πρόταση στο πρόγραμμα μετά την **if**.

Προσοχή! Το ερωτηματικό δεν εμφανίζεται ποτέ μετά τις παρενθέσεις του σχεσιακού ελέγχου (<υποθεση>, βλπ κανόνες 1,2,3). Τα ερωτηματικά εμφανίζονται μετά από κάθε πρόταση του <σωμα> - <αποδοση> του **if** (βλπ κανόνα 12).

Τέλος, η C++ ερμηνεύει κάθε μη μηδενική τιμή σαν **αληθή** και το μηδέν σαν **ψευδή**. Αυτό μας επιτρέπει να χειριζόμαστε τις λογικές μεταβλητές στη λογική **if**.

Με τα παραδείγματα που ακολουθούν, γίνεται μια προσπάθεια να δοθεί μια πληρέστερη και όσο το δυνατόν σφαιρικότερη εικόνα για τις περιπτώσεις χρήσης και εφαρμογής της πρότασης **if** στον προγραμματισμό. Φυσικά οι επιμέρους εικόνες - των επί μέρους προτάσεων - ολοκληρώνονται όσο ολοκληρώνεται η εικόνα - παρουσίαση όλων των

προτάσεων της γλώσσας.

Παραδείγματα:

```
1. if ( poliseis > 5000 )
   { epihorigima = 500;}
```

Εάν η παραπάνω πρόταση είναι μέρος ενός προγράμματος της C++, η τιμή της μεταβλητής poliseis (πωλήσεις) επηρεάζει τη λήψη απόφασης, δηλαδή για το τι θα ακολουθήσει μετά. Έτσι, **εαν** poliseis > 5000 **τοτε** epihorigima = 500 (στο επιχορήγημα - epihorigima - ανατίθεται η (αρχική) τιμή 500), δηλαδή εκτελείται το <σωμα> της **if**. Στην αντίθετη περίπτωση όμως που poliseis < = 5000 τότε το <σωμα> της **if** δεν θα εκτελεστεί, το πρόγραμμα, όμως, θα προχωρήσει στην εκτέλεση της επόμενης πρότασης, δηλαδή αυτής που ακολουθεί την πρόταση **if**.

```
2. if ( forologoumeno_poso . pososto_forou = = elahisto )
   { hamilos_misthos = 207,603;}
```

Η παραπάνω πρόταση υποτίθεται ότι είναι μέρος ενός προγράμματος της C++. Στην <υποθεση> της **if** όταν χρησιμοποιούνται εκφράσεις, π.χ. της μορφής:

```
forologoumeno_poso . pososto_forou = = elahisto,
```

για να γίνονται πιό ευανάγνωστες χρειάζονται παρενθέσεις, π.χ.:

```
if ( ( forologoumeno_poso . pososto_forou ) = = elahisto )
  { hamilos_misthos = 207,603;}
```

ακόμη, και εάν η C++ δεν το απαιτεί λόγω της ισχύουσας προτεραιότητας στον αυτόματο χειρισμό των τελεστών της. Κατά τα άλλα, στο παράδειγμα_2 ισχύουν ό,τι και στο παράδειγμα_1.

3. Παρατίθεται ένα δείγμα προγράμματος που υπολογίζει την πληρωμή ενός πωλητή. Ο πωλητής πληρώνεται με 4,500 δρχ την ώρα. Επιπλέον, εάν οι πωλήσεις ξεπερνούν τις 90,000 δρχ, στον πωλητή δίνεται επίδομα 8,000 δρχ. Το πρόγραμμα αυτό είναι ένα παράδειγμα της υπό συνθήκη λογικής εξαγωγής συμπεράσματος. Η <υποθεση> (συνθήκη) εξαρτάται από τη σχέση μεταξύ των μεταβλητών poliseis (πωλήσεις) και misthos (μισθός).

```
// EG3PLIROMIS.CPP
```

```
// Υπολογίζει την πληρωμή ενός πωλητή βάσει των πωλήσεών του
// Πρόωρη Έξοδος λόγω της exit( )
#include< iostream.h>
main( )
{
 char onoma_politi[20];
 int ores;
 float synolo_poliseon, epihorigima, misthos;
 cout << "\n\n";     // Εκτυπώνει δύο κενές γραμμές
 cout << " Υπολογισμός πληρωμής πωλητή\n";
 cout << ". . . . . . . . . . . . . . . . . . . . . . . .\n";
 //        Εισαγωγή των δεδομένων από τον χρήστη
 cout << " Εισάγετε το επώνυμο του πωλητή : ";
 cin >> onoma_politi;
 cout << " Εισάγετε τις ώρες απασχόλησης του πωλητή : ";
 cin >> ores;
 cout << " Εισάγετε το σύνολο των πωλήσεων που έγιναν : ";
 cin >> synolo_poliseon;
 epihorigima = 0.00;  // Αρχικά δεν δίδεται το επιχορήγημα
 //        Υπολογισμός του βασικού μισθού του πωλητή
 misthos = 4,500 * (float)ores; // Αλλαγή του τύπου της μεταβλητής ores (ώρες)
 //        Προσθέτει το επιχορήγημα όταν οι πωλήσεις είναι υψηλές
if ( poliseis > 90,000 )
   { epihorigima = 8,000;}
 cout << " Ο πωλητής : " << onoma_politi << "\n";
 cout << " εισέπραξε μισθό :" << misthos << "δρχ\n";
 cout << " και επιχορήγημα : " << epihorigima << "δρχ\n";
 return(0);
}
```

Η έξοδος που ακολουθεί δείχνει το αποτέλεσμα από τη διπλή εκτέλεση αυτού του προγράμματος κάθε φορά με διαφορετικά δεδομένα.

```
Υπολογιστής πληρωμής πωλητή
. . . . . .
Εισάγετε το επώνυμο του πωλητή : Pappas
Εισάγετε τις ώρες απασχόλησης του πωλητή : 40
Εισάγετε το σύνολο των πωλήσεων που έγιναν : 85,000
Ο πωλητής : Pappas
εισέπραξε μισθό : 180,000 δρχ
και επιχορήγημα : 0.00 δρχ

Υπολογιστής πληρωμής πωλητή
. . . . . .
Εισάγετε το επώνυμο του πωλητή : Kappas
Εισάγετε τις ώρες απασχόλησης του πωλητή : 40
Εισάγετε το σύνολο των πωλήσεων που έγιναν : 100,000
Ο πωλητής : Kappas
εισέπραξε μισθό : 180,000 δρχ
και επιχορήγημα : 8,000 δρχ
```

4. Στον προγραμματισμό καλό είναι, όταν είναι εφικτό και όσο είναι δυνατόν, να προγραμματίζεται και ο έλεγχος της εγκυρότητας των δεδομένων - τιμών την ώρα που εισάγονται - πληκτρολογούνται από τους χρήστες.

Έτσι εάν εισάγουν λάθος τιμή, για παράδειγμα μια αρνητική τιμή όταν τα δεδομένα δεν μπορεί να είναι αρνητικά, είναι δυνατόν να έχει προγραμματιστεί για την περίπτωση αυτή ένας έλεγχος, που θα τους πληροφορεί ότι υπάρχει πρόβλημα και ότι πρέπει να επαναεισάγουν τα δεδομένα.

Εξυπακούεται ότι έλεγχος μπορεί να γίνει για τιμές που βρίσκονται έξω από κάποιο αποδεκτό διάστημα τιμών, όχι όμως και για τιμές του διαστήματος. Για παράδειγμα η ηλικία των σπουδαστών δεν μπορεί να είναι 135 ή -4, όμως εάν αντί για 20, πληκτολογηθεί 19, αυτό γενικά δεν μπορεί να ελεχθεί, ιδίως εάν και οι δύο τιμές ανήκουν στο αποδεκτό διάστημα ηλικιών των σπουδαστών.

4.6.1.1.1 Ερωτήσεις για Επανάληψη και Εμπέδωση

1. Δουλέψτε με μολύβι και χαρτί τους αναδρομικούς κανόνες-ορισμούς της σύνταξης της πρότασης **if**. Συμπληρώσατέ τους αν χρειαστεί.

2. Είναι αληθές ή ψευδές ότι, τυπώνεται στην οθόνη το μήνυμα: "η C++ είναι υψηλή χαμηλού επιπέδου γλώσσα" , όταν εκτελείται η παρακάτω πρόταση.

```
if ( 54 <= 54 )
  { cout << " η C++ είναι υψηλή χαμηλού επιπέδου γλώσσα \n"; }
```

3. Εκτελείται ο παρακάτω τελεστής **cout**;

```
if ( 3 != 4 != 1 )
  { cout << " η C++ είναι υψηλή χαμηλού επιπέδου γλώσσα \n"; }
```

4. Εκτελείται ο παρακάτω τελεστής **cout**;

```
if ( !0 )
  { cout << " η C++ είναι υψηλή χαμηλού επιπέδου γλώσσα \n"; }
```

5. Ποιά είναι η τιμή που εκτυπώνεται στο παρακάτω πρόγραμμα; (Υπόδειξη: Στη συνθήκη του **if**, σε κάθε πλευρά του || οι τελεστές είναι ανάθεσης.)

```
// Όνομα αρχείου: EKTYPOSI.CPP
// Έλεγχος Λογικών Τελεστών
#include< iostream.h>
main( )
{
  int a, b;
  a = 5;
  b = 13;
  if ( a = 25 ) || ( b = 35 )
     { cout << " η a είναι :" << a << " ενώ η b θα γίνει :" << b;}
  return(0);
}
```

4.6.1.2 Η Πρόταση else ή οι Προτάσεις if-else

Η πρόταση **else** ποτέ δεν εμφανίζεται σε ένα πρόγραμμα χωρίς μια πρόταση **if**. Έτσι έχουμε πάντοτε συνδυασμό των προτάσεων **if-else**, ο οποίος θα μπορούσε να θεωρηθεί σαν επέκταση της απλής πρότασης **if** και λειτουργεί στην πράξη σαν **πρόταση απόφασης**

ή **εκλογής** ή **υποθετική** (conditional statement).

Η σύνταξη των προτάσεων **if-else** ακολουθεί τους εξής κανόνες οι οποίοι διατυπώνονται με τη βοήθεια της BNF:

1. <πρόταση_if> ::= **if**(<υπόθεση>)
   {<απόδοση_1>]
   else
   {<απόδοση_2>} ,κ.λπ.,

δηλαδή, όπου: <υπόθεση>, <απόδοση_1>, <απόδοση_2> ορίζονται όπως και προηγουμένως από τους κανόνες 2 έως και 13. Εξυπακούεται ότι η <απόδοση_1> και η <απόδοση_2> ορίζονται με τη βοήθεια των ίδιων κανόνων 11, 12 και 13.

Στην πρόταση **if-else** το πρώτο μέρος της είναι ακριβώς ίδιο με αυτό της πρότασης **if**. Δηλαδή, αν η <υπόθεση> είναι **αληθής** (true) τότε εκτελείται η <απόδοση_1>, το σώμα που ακολουθεί την **if**. Αν όμως η <υπόθεση> είναι **ψευδής** (false) τότε εκτελείται η <απόδοση_2>, το σώμα που ακολουθεί την **else**.

Θα μπορούσαμε να συνοψίσουμε τις ομοιότητες και τη διαφορά αυτών των δύο προτάσεων απόφασης ή εκλογής ή υποθετικών προτάσεων ως εξής:

" Η πρόταση **if** αποφασίζει τι θα συμβεί μόνο αν η <υπόθεση> είναι **αληθής** (true), ενώ η πρόταση **if-else** αποφασίζει και στην περίπτωση που η <υπόθεση> είναι **ψευδής** (false)."

Ανεξάρτητα από την διαδρομή που ακολουθείται σε μια **if-else** πρόταση, μετά την έξοδο εκτελείται η πρόταση που ακολουθεί την **if-else** πρόταση.

Σημείωση: Στις **if-else** προτάσεις μπορεί να συγκρίνονται και χαρακτήρες εκτός από αριθμούς. Στην περίπτωση αυτή η C++ χρησιμοποιεί τον πίνακα ASCII για να αποφασίσει ποιός χαρακτήρας είναι "μικρότερος" από τον άλλον, ανάλογα με το ποιός χαρακτήρας είναι χαμηλότερα στον πίνακα αυτόν. Δεν μπορούν όμως να συγκρίνονται απευθείας μεταξύ τους, με σχεσιακούς τελεστές, αλυσίδες χαρακτήρων ή παρατάξεις αλυσίδων χαρακτήρων.

Παρατηρήσεις: Στον σχεδιασμό κάποιου προγράμματος μπορεί να προκύπτουν περισσότερες των τριών ή τεσσάρων κιβωτισμένων **if** ή **if-else**. Το σχήμα αυτό μπορεί να προκαλέσει σύγχυση στον προγραμματιστή, το πρόβλημα αυτό, των πολλαπλών επιλογών ή κιβωτισμένων **if** ή **if-else**, αντιμετωπίζεται αποτελεσματικά με την πρόταση **switch** (βλ

π.χ. 4 και §4.6.1.5).

Παραδείγματα:

1. Να γραφεί πρόγραμμα που ζητάει από το χρήστη έναν αριθμό, μετά τυπώνει τον αριθμό αναλόγως αν είναι θετικός ή αρνητικός με την χρήση της πρότασης **if-else**.

```
// EG1IFELSE1.CPP
// Παρουσίαση της if-else
// Εκτυπώνει εάν η τιμή εισόδου είναι μεγαλύτερη του μηδενός ή όχι
#include< iostream.h>
main( )
{
 int arithm;
 cout << " Δώστε τον αριθμό :";
 cin >> arithm;  // Εισαγωγή του αριθμού από τον χρήστη
 if ( arithm > 0 )
   { cout << " Ο αριθμός είναι μεγαλύτερος του μεδενός \n ";}
 else
   { cout << " Ο αριθμός είναι μικρότερος ή ίσος του μεδενός \n ";}

 //   Ανεξάρτητα από τον αριθμό εκτυπώνεται το παρακάτω
 cout << " \n\n Ευχαριστούμε για το χρόνο που διαθέσατε ! \n";
 return(0);
}
```

Παρατήρηση: Εφόσον χρησιμοποιείται η **else** δεν χρειάζεται να γίνει έλεγχος εάν ο αριθμός είναι μικρότερος ή ίσος του μηδενός, διότι η **if** ελέγχει εάν ο αριθμός είναι μεγαλύτερος του μηδενός και η **else** αναλαμβάνει αυτόματα τις υπόλοιπες πιθανότητες.

2. Να γραφεί πρόγραμμα που ζητάει από το χρήστη το επώνυμό του, το οποίο και αποθηκεύει σε ένα πίνακα χαρακτήρων. Το πρόγραμμα στη συνέχεια ελέγχει τον πρώτο χαρακτήρα του πίνακα για να βρεί εάν βρίσκεται στο πρώτο μισό του αλφαβήτου. Εάν βρίσκεται τότε εμφανίζεται ένα κατάλληλο μήνυμα.

```
// EG2IFELSE2.CPP
// Ελέγχει το αρχικό γράμμα του επωνύμου του χρήστη και
// εκτυπώνει κατάλληλο μήνυμα
#include< iostream.h>
```

```
main( )
{
 char eponymo[20];    // Προσδιοριστής του επωνύμου
 cout << " Δώστε το επώνυμό σας : ";
 cin >> eponymo;

 // Έλεγχος του πρώτου γράμματος
 if ( eponymo <= 'P' )
   { cout << " Το επώνυμό σας είναι από τα πρώτα στον κατάλογο \n "; }
 else
   { cout << " Περιμένετε για λίγο έως ότου ανακληθεί το επώνυμό σας \n "; }
 return(0);
}
```

Παρατήρηση : Επειδή το πρόγραμμα συγκρίνει στοιχείο πίνακα με έναν χαρακτήρα, πρέπει ο χαρακτήρας να βρίσκεται μέσα σε εισαγωγικά. Ο τύπος δεδομένων σε κάθε πλευρά του σχεσιακού τελεστή πρέπει να ταιριάζει.

3. Το πρόγραμμα που ακολουθεί είναι μια ολοκληρωμένη διαδικασία πληρωμών σε σχέση με αυτή του προηγούμενου προγράμματος. Παραθέτουμε τη λογική του προγράμματος το οποίο χρησιμοποιεί την πρόταση **if** για να υπολογίσει την πληρωμή των υπερωριών:

" - Εάν οι υπάλληλοι εργάζονται 40 ώρες ή λιγότερο τότε πληρώνονται κανονικά.

- Εάν εργάζονται 40 με 50 ώρες τότε πληρώνονται κανονικά για τις 40 πρώτες ώρες και επιπλέον πληρώνονται για τις υπόλοιπες ώρες μιάμιση φορά επί του κανονικού ωρομισθίου.

- Εάν οι ώρες είναι άνω των 50 τότε πληρώνονται διπλά σε σχέση με τον κανονικό μισθό."

```
// EG3IFELSE3.CPP
// Υπολογισμός όλων των πιθανών πλήρων πληρωμών υπερωρίας
#include< iostream.h>
#include< stdio.h>
main( )
{
 int ores;
 float diplyperoria, miamisyperoria, kanonikorario, pososto, pliromi;

 cout << " Δώστε τις ώρες που έχουν δουλευτεί : ";
 cin >> ores;

 cout << " \n Δώστε την κανονική ωριαία αποζημίωση : ";
```

```
cin >> pososto;
// Υπολογισμός πληρωμής ανάλογα με τις ώρες απασχόλησης
// Πιθανότητα διπλού μισθού
if ( ores > 50 )
  { diplyperoria = 2.0 * pososto * (float)(ores - 50);
    miamisyperoria = 1.5 * pososto * 10.0; }
else
  { diplyperoria = 0.0; }
// Πιθανότητα ενάμισυ μισθού
if ( ores > 40 )
  { miamisyperoria = 1.5 * pososto * (float)(ores - 40); }
// Πιθανότητα κανονικού μισθού
if ( ores >= 40 )
  { kanonikorario = 40 * pososto; }
else
  { kanonikorario = (float) ores * pososto; }
 // Πρόσθεση των τριών συστατικών της πληρωμής
pliromi = diplyperoria + miamisyperoria + kanonikorario;
cout << "\n  Ο μισθός είναι :" << pliromi; }
return(0);
}
```

4. Το πρόγραμμα που ακολουθεί χρησιμοποιείται για τον εντοπισμό των υπαλλήλων μιας εταιρίας προκειμένου να βραβευθούν για την πολυετή προσφορά τους στην εταιρία. Έτσι για το παράδειγμα δίνεται ένα χρυσό ρολόι σε αυτούς που δουλεύουν περισσότερα από 20 χρόνια, ένα πρές-παπιέ σε αυτούς που δουλεύουν περισσότερα από 10 χρόνια και τίποτα, για την ώρα, σε όλους τους υπόλοιπους.

```
// EG4IFELSE4.CPP
// ΥΕκτυπώνει ένα μήνυμα ανάλογα με τα χρόνια υπηρεσίας
#include< iostream.h>
main( )
{
 int xronia;
 cout << " Δώστε τα χρόνια υπηρεσίας :";
```

```
    cin >> xronia;

    if ( xronia > 20 )
      { cout << " Δικαιούται χρυσό ρολόι \n";}
    else
      { if ( xronia > 10 )
          { cout << " Δικαιούται πρές-παπά \n";}
        else
          { cout << " Ευχαριστούμε για την προσφορά σου \n";}
      }
    return(0);
}
```

Παρατήρηση: Η απόδοση (<αποδοση>) της **if** μπορεί να είναι οποιαδήποτε αποδεκτή πρόταση της C++, ακόμα και μια άλλη πρόταση **if**. Αυτό μερικές φορές είναι χρήσιμο, καλό όμως είναι να αποφεύγεται τον κιβωτισμό πολλών **if** για τον χειρισμό πολλών συνθηκών, διότι περισσότερες από τρείς ή τέσσερεις συνθήκες μπορεί να προκαλέσουν σύγχυση.

4.6.1.2.1 Ερωτήσεις για Επανάληψη και Εμπέδωση

1. Δουλέψτε με χαρτί και μολύβι τους αναδρομικούς κανόνες-ορισμούς της σύνταξης της πρότασης **if-else**, συμπληρώσατέ τους αν χρειαστεί.

2. Ποιά είναι η διαφορά μεταξύ των προτάσεων **if** και **if-else**;

3. Ξαναγράψτε τον παρακάτω τελεστή συνθήκης σαν πρόταση **if-else**:

```
apandisi = ( a == b )? c + 2 : c + 3;
```

4. Ποιά από τις παρακάτω προτάσεις που περιέχουν τον τελεστή **cout** εκτελείται;

```
if ( 3 != 4 != 1 )
  { cout << " η C++ είναι υψηλή χαμηλού επιπέδου γλώσσα \n";}
else
  { cout << " η FORTRAN και η Pascal είναι ανωτέρου επιπέδου γλώσσες \n";}
```

4.6.1.3 Η Πρόταση - Βρόγχος while

Η πρόταση **while** είναι μια από τις προτάσεις που δομούν **συγκροτήματα από προτάσεις** (blocks of statements) και οι οποίες θεωρούνται σαν μια πρόταση. Κάθε **πρόταση που δομεί συγκροτήματα** αποτελείται απο μια πρόταση ή από μια ακολουθία

προτάσεων γλώσσας προγραμματισμού. Η πρόταση αυτή ελέγχει ή οι προτάσεις αυτές ελέγχουν αντίστοιχα, επαναληπτικές εκτελέσεις (ανακυκλώσεις) της ίδιας ενότητας προγράμματος (διαδικασίας) που αποτελείται και πάλι από μια πρόταση ή μια ακολουθία προτάσεων. Η επαναληπτική εκτέλεση, φυσικά υπό όρους, μιας ενότητας προγράμματος λέγεται **βρόγχος** (loop).

Η σύνταξη της **while** ακολουθεί τους εξής κανόνες οι οποίοι διατυπώνονται με τη βοήθεια της BNF:

1. <προταση_while> ::= **while**(<συνθηκη>)
   {<σωμα>} ,

2. <συνθηκη> ::= <λογικη_εκφραση>, (πρβλ κανόνες §4.6.1.1),

3. <σωμα> ::= <συνθετη_προταση> , (πρβλ κανόνες §4.6.1.1).

Οι παρενθέσεις στην <συνθηκη> επιβάλλονται. Όσο η <συνθηκη> είναι αληθής (μη μηδενική), το <σωμα> που αποτελείται από μια ή περισσότερες προτάσεις εκτελείται επαναληπτικά μέχρι η <συνθηκη> να γίνει ψευδής (υπολογίζει μηδέν).

Τα άγκιστρα απαιτούνται πριν και μετά το <σωμα>. Αν το <σωμα> περιλαμβάνει μια μόνο πρόταση τα άγκιστρα δεν απαιτούνται, όμως, είναι μια καλή συνήθεια να βάζουμε πάντοτε άγγιστρα για να μη τα ξεχνάμε όταν χρειάζονται.

Επίσης δεν πρέπει να παραλείπεται στο τέλος κάθε πρότασης του <σωμα> ο οριοθέτης " ; ".

Η <συνθηκη> περιέχει συνήθως σχεσιακούς και λογικούς τελεστές, οι οποίοι εφαρμοζόμενοι δίνουν τιμή αληθή ή ψευδή στη <συνθηκη>, ανάλογα τότε ο βρόγχος εκτελείται μια φορά, πολλές φορές ή δεν εκτελείται καθόλου.

Επειδή η <συνθηκη> ορίζει πότε τελειώνει ο βρόγχος, θα πρέπει να υπάρχει η δέουσα προσοχή ώστε η <συνθηκη> να ελέγχει σωστά τις τιμές των μεταβλητών που χειρίζεται και να έχουμε την κατάλληλη στιγμή τερματισμό. Σε περίπτωση λάθους, υπάρχει κίνδυνος ο βρόγχος να επαναλαμβάνεται αενάως, δηλαδή χωρίς τερματισμό, αυτό είναι γνωστό ως **ατελεύτητος βρόγχος** (endless loop) και πρέπει να αποφεύγεται[*].

---

[*] Αν έχει γραφτεί κατά λάθος ένας ατελεύτητος βρόγχος, πρέπει ο χρήστης του Η/Υ να σταματήσει το πρόγραμμα μόνος του. Έτσι:

1. Αν χρησιμοποιεί PC, πρέπει να πατήσει το Ctrl-Break .
2. Αν χρησιμοποιεί σύστημα βασισμένο σε UNIX, τότε ο διαχειριστής του συστήματος μπορεί και πρέπει να σταματήσει την εκτέλεση του προγράμματός του.

Παράδειγμα: Δίδεται ένα πρόγραμμα που ελέγχει ή επιβεβαιώνει την απάντηση που δίνει ή την τιμή που εισάγει ο χρήστης του. Το πρόγραμμα, προσαρμοζόμενο μπορεί να αποτελέσει μέρος οποιουδήποτε προγράμματος, το οποίο ζητά κάποιες συγκεκριμένες απαντήσεις από τον χρήστη του, π.χ., ένα "ναί" (ή ισοδύναμα "1") ή ένα "όχι" (ή ισοδύναμα "0"), ή ακόμα κάποιες συγκεκριμένες τιμές, π.χ., N < 100, προκειμένου το πρόγραμμα να προχωρήσει παρακάτω ή σε μια νέα ανακύκλωσή του, ή τέλος να εκτραπεί σε άλλο σημείο του κ.λπ..

```
. . .
#include <iostream.h>
main( )
{
 char apandisi;
 cout << "Θέλεις να συνεχίσεις (N/Y)?";
 cin >> apandisi;      // Απάντηση του χρήστη
 while ( ( apandisi != 'Y' ) && (apandisi != 'N' ) )
   {
     cout << "\n Πρέπει να πληκτρολογήσεις Y ή N \n"; // Προειδοποιεί
                                                      // και
     cout << "Θέλεις να συνεχίσεις (N/Y)?";           // Ξαναρωτά
     cin >> apandisi;      // Νέα απάντηση του χρήστη
   }
 return(0);
}
```

4.6.1.3.1 Η Πρόταση - Βρόγχος do-while

Η πρόταση **do-while** ελέγχει τον βρόγχο **do-while**. Οι δύο βρόγχοι **do-while** και **while** είναι όμοιοι, εκτός από το ότι ο έλεγχος "<συνθηκη>" στον βρόγχο **do-while** πραγματοποιείται στο τέλος και όχι στην αρχή όπως στον βρόγχο **while**. Αυτό σημαίνει ότι το <σωμα> στον βρόγχο **do-while** εκτελείται τουλάχιστο μια φορά.

Η σύνταξη της **do-while** ακολουθεί τους εξής κανόνες οι οποίοι διατυπώνονται με τη βοήθεια της BNF:

1. <προταση_do_while> ::= **do**
   {<σωμα>}

while(<συνθηκη>);,

2. <σωμα> ::= <συνθετη_προταση> , (πρβλ κανόνες §4.6.1.1),

3. <συνθηκη> ::= <λογικη_εκφραση>, (πρβλ κανόνες §4.6.1.1).

Στην πρόταση **do-while** ισχύουν ό,τι και στην πρόταση **while**, για τα άγγιστρα στο <σωμα>, τις παρενθέσεις και την ορθότητα της <συνθηκη> καθώς και τους ατελεύτητους βρόγχους.

Παραθέτουμε το παράδειγμα της προηγούμενης παραγράφου, στο οποίο, το πρόγραμμα διατυπώνεται με τη βοήθεια της πρότασης **do-while** αντί για την πρόταση **while**.

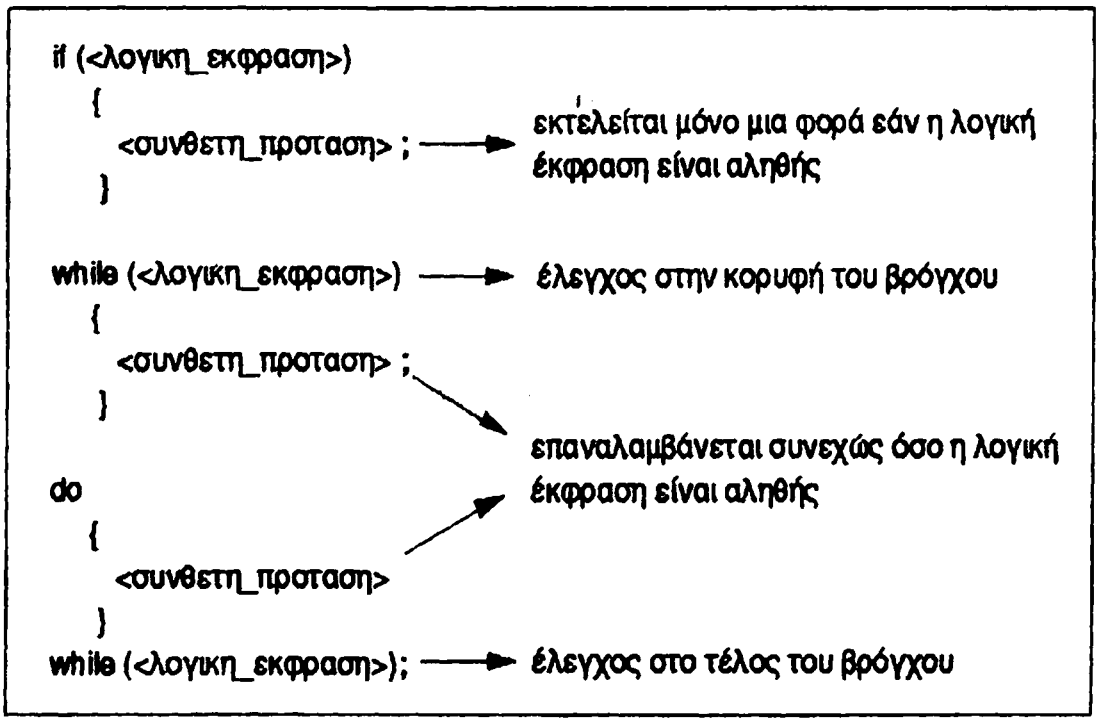
```
. . .
#include <iostream.h>
main( )
{
 char apandisi;
 do
   {
     cout << "\n Πρέπει να πληκτρολογήσεις Υ ή Ν \n"; // Προειδοποιεί
                                                      // και
     cout << "Θέλεις να συνεχίσεις (Ν/Υ)?";           // Ξαναρωτά
     cin >> apandisi;      // Απάντηση του χρήστη - Ο βρόγχος τελειώνει εδώ
   }
   while ( ( apandisi != 'Υ' ) && (apandisi != 'Ν' ) );
 return(0);
}
```

4.6.1.3.2 Διαφορές μεταξύ των Προτάσεων - Βρόγχων if και while

Πολλές φορές στους αρχάριους προγραμματιστές δημιουργείται σύγχυση μεταξύ των προτάσεων **if** από τη μια και **while**, **do-while** από την άλλη.

Οι βρόγχοι **while** και **do-while** επαναλαμβάνουν ένα τμήμα του κώδικα (<σωμα::= <συνθετη_προταση>) πολλές φορές, τόσες, όσες η <συνθηκη> (::=<λογικη_ εκφραση>) το επιτρέπει.

Η πρόταση **if** μπορεί να εκτελεί ή όχι ένα τμήμα του κώδικα (<αποδοση>::= <συνθετη_προταση>), εάν ανάλογα με την <υποθεση> (::=<λογικη_εκφραση>) εκτελείται, τότε αυτό γίνεται μόνο μια φορά.



ΠΙΝΑΚΑΣ 4.6.1.3.2 : Διαφορές μεταξύ της πρότασης if και των δύο βρόγχων while

4.6.1.3.3 Η Συνάρτηση exit( )

Η C++ μας παρέχει τη δυνατότητα με τη συνάρτηση **exit( )** για πρόωρη έξοδο από το πρόγραμμα, δηλαδή εγκατάλειψης του προγράμματος νωρίτερα, πριν από το κανονικό του τέλος. Η μορφή της **exit( )** είναι:

exit(<ακεραια_μεταβλητη>**)** ή **exit(**<περιγραφικη_σταθερα>**).**

Η πρόωρη έξοδος από ένα πρόγραμμα πραγματοποιείται με την άμεση επικοινωνία του χρήστη με το λειτουργικό σύστημα, με τη βοήθεια των κωδικών επιστροφής του λειτουργικού συστήματος. Έτσι, δίνοντας κάποιες εκ των προτέρων γνωστές τιμές στην <ακεραια_μεταβλητη> ή στην <περιγραφικη_σταθερα>, οι οποίες ορίζουν την <καταταση>, παρέχεται η δυνατότητα στον χρήστη να ελέγχει τα αποτελέσματα των

προγραμμάτων της C++ (βλπ παράδειγμα 2).

Για παράδειγμα στο DOS, αφού δοθεί η τιμή στην <κατασταση>, αυτή στέλνεται στη <μεταβλητη_περιβαλλοντος_επιπεδου_λαθους> του λειτουργικού συστήματος, που ελέγχεται από μαζικά αρχεία. Πολλές φορές, συμβαίνει κάτι σοβαρό σε κάποιο πρόγραμμα, π.χ. λάθος ενός οδηγού δίσκου, που απαιτεί το τέλος του προγράμματος. Τότε ο χρήστης του προγράμματος μπορεί να το διακόψει δίνοντας με τον cin, που εμφανίζεται, μια εδική τιμή εκ των προτέρων γνωστή[60], η οποία διαβιβάζεται μετά από τον έλεγχο της <συνθηκη> ως παράμετρος της **exit( )**, η οποία έχει τοποθετηθεί στο <σωμα> μιας **if**.

Παρατηρήσεις:

1. Η **exit( )** μπορεί να γραφεί σε μια γραμμή σε οποιοδήποτε σημείο του προγράμματος, όπως οποιαδήποτε πρόταση ή συνάρτηση της C++.
2. Η χρησιμοποίηση της συνάρτησης **exit( )** σε ένα πρόγραμμμα, προυποθέτει την αναφορά στο αρχείο **stdlib.h** διά της **#include**< stdlib.h >. Το αρχείο stdlib.h περιγράφει τη λειτουργία της **exit( )** στο πρόγραμμα που εμφανίζεται. Σημειώνουμε ότι, όποτε χρησιμοποιείται μια συνάρτηση σε ένα πρόγραμμμα, απαραίτητα πρέπει να αναφέρεται το αντιστοιχο αρχείο, που περιγράφει τη λειτουργία της στο πρόγραμμα, στην **#include**< >. Τα αρχεία αυτά περιέχονται συνήθως στο εγχειρίδιο οδηγιών του μεταγλωττιστή.

Παραδείγματα: Δίνονται δύο γενικά παραδείγματα - προγράμματα, ένα λειτουργίας και ένα χρήσης της συνάρτησης **exit( )**:

1. **Παράδειγμα Λειτουργίας** της συνάρτησης **exit( )**,

```
// EG1EXIT.CPP
// Παρουσίαση συνάρτησης exit( )
// Πρόωρη Έξοδος λόγω της exit( )
#include< iostream.h>
#include< stdlib.h >     // χρειάζεται λόγω της exit( )
main( )
{
 exit(0);     // υποχρεώνει το πρόγραμμα σε πρόωρη έξοδο σε αυτό το σημείο
```

[60] Η τιμή αυτή ορίζεται από τον προγραμματιστή, δηλαδή από αυτόν που σχεδίασε και έγραψε το πρόγραμμα.

```
cout << " Είναι ευχάριστος ο προγραμματισμός με την C++\n";
cout << " Είναι ενδιαφέρον να μαθαίνεις την C++ με παραδείγματα\n";
cout << " Η C++ είναι μια δυναμική γλώσσα που εξοικειώνεσαι εύκολα.";
return(0);
}
```

2. **Παράδειγμα Χρήσης** της συνάρτησης **exit( )**. Με το παράδειγμα αυτό παρουσιάζεται πως μπορεί να χρησιμοποιηθεί ένα πρόγραμμα για να ελεγχθούν δύο άλλα προγράμματα. Έτσι το πρόγραμμα του παραδείγματος είναι τελικά ένα πρόγραμμα επιλογής, που επιπλέον δείχνει πως η C++ μπορεί να περάσει και πληροφορίες στο DOS με την **exit( )**. Καταρχήν ο χρήστης αποφασίζει τι θέλει να κάνει ο Η/Υ σου, επιλέγοντας από τις διαθέσιμες επιλογές του προγράμματος. Στη συνέχεια το πρόγραμμα διαβιβάζει την επιλεγμένη από τον χρήστη τιμή στο λειτουργικό σύστημα, στο οποίο τέλος έγκειται να ελέγξει την τιμή της **exit( )** και να εκτελέσει το κατάλληλο πρόγραμμα.

```
// EG2EXIT.CPP
// Ζητά επιλογή από τον χρήστη και επιστρέφει την επιλογή
// στο λειτουργικό σύστημα με την exit( )
#include< iostream.h>
#include< stdlib.h >     // χρειάζεται λόγω της exit( )
main( )
{
 int apandisi;
 do
  {
   cout<<" Εισάγοντας την τιμή:\n\n";
   cout<<"1. Ενεργοποιείς τον Επεξεργαστή Κειμένου (word processor)\n";
   cout<<"2. Ενεργοποιείς το πρόγραμμα Βάσης Δεδομένων (dbase program)\n";
   cout<<" Ανάλογα λοιπόν πληκτρολόγησε μια από τις δύο τιμές\n";
   cin>> apandisi;
  }
 while (( apandisi != 1 ) && ( apandisi != 2 )); // Έλεγχος εισαγωγής 1ή 2 από
                                                 // τον χρήστη
 exit(apandisi);  // Επιστροφή τιμής στο λειτουργικό σύστημα
 return(0);       // Το return δεν εκτελείται λόγω της exit( )
```

```
}
```

4.6.1.3.4 Η Πρόταση break

Η πρόταση **break** χρησιμοποιείται για την έξοδο από τρέχοντα βρόγχο και όχι για την έξοδο από το πρόγραμμα όπως συμβαίνει με την **exit( )**.

Η μορφή της πρότασης είναι απλώς **break;** . Η πρόταση **break** μπορεί να γραφεί σε μια γραμμή σε οποιοδήποτε σημείο του προγράμματος, όπως οποιαδήποτε πρόταση ή συνάρτηση της C++. Στην πράξη όμως, η πρόταση **break** εμφανίζεται στο <σωμα> βρόγχων **while** ή **do_while** , όπου λειτουργεί για πρόωρη έξοδο από αυτούς.

Σημείωση: Στην περίπτωση κιβωτισμένων βρόγχων **while** ή **do_while** , η πρόταση **break** λειτουργεί για πρόωρη έξοδο από τον τρέχοντα μόνο βρόγχο, δηλαδή τον εσωτερικό ή τον εξωτερικό ανάλογα σε ποιού το <σωμα> ανήκει η **break**.

Παραδείγματα: Δίνονται δύο γενικά παραδείγματα προγράμματα, ένα λειτουργίας και ένα χρήσης της πρότασης **break**:

1. **Παράδειγμα Λειτουργίας** της πρότασης **break**,

```
// EG1BRK.CPP
// Παρουσίαση πρότασης break
#include< iostream.h>
main( )
{
 char apendisi_christi ;
 do
  {
   cout << " Είναι ευχάριστος ο προγραμματισμός με την C++\n";
   cout << " Είναι ενδιαφέρον να μαθαίνεις την C++ με παραδείγματα\n";
   cout << " Η C++ είναι μια δυναμική γλώσσα που εξοικειώνεσαι εύκολα.\n";
   break;
   cout<<" Θέλεις να ξαναπιάσουμε τα μηνύματα(N/Y)?\n";
   cin>> apendisi_christi ;
```

```
  }
 while ( apandisi_christi == 'Y' );
 cout << " Αυτά για την ώρα.";
 return(0);
}
```

Αυτό το πρόγραμμα δίνει πάντοτε την απάντηση:

```
Είναι ευχάριστος ο προγραμματισμός με την C++
Είναι ενδιαφέρον να μαθαίνεις την C++ με παραδείγματα
Η C++ είναι μια δυναμική γλώσσα που εξοικειώνεσαι εύκολα.
Αυτά για την ώρα.
```

2. **Παράδειγμα Χρήσης** της πρότασης **break**. Αντίθετα από το προηγούμενο πρόγραμμα, σε αυτό που έπεται η **break** εμφανίζεται, όπως συνήθως, στην <αποδοση> μιας **if**. Αυτό δημιουργεί μια διακοπή (**break**) υπό συνθήκη, η οποία λαμβάνει χώρα μόνο εάν η <υποθεση> της πρότασης **if** είναι αληθής.

```
// EG2BRK.CPP
// Εισάγονται στοιχεία από χρήστη που αντιστοιχούν σε αποθέματα και
// τυπώνει κατάλογο συνολικών στοιχείων που δίδονται ή υπολογίζονται
#include< iostream.h>
#include< iomanip.h>
main( )
{
 int kodarithm, posotita;
 float kostos;
 cout << "*** Μηχανική Απογραφή Εμπορευμάτων ***\n\n";      // Τίτλος
 // Λήψη πληροφοριών αποθεμάτων
 do
  {
   cout << " Δώστε τον επόμενο κωδικό αριθμό (kodarithm) - ΤΕΡΜΑΤ ← -999 ";
   cin >> kodarithm;
   if ( kodarithm == -999 )
     { break; }          // Έξοδος εάν δεν υπάρχουν άλλοι κωδικοί
   cout << " Ποσότητα που αγοράστηκε : ";
```

```
    cin >> posotita;
    cout << " Τιμή ανά μονάδα κάθε προϊόντος : ";
    cin >> kostos;
    cout << " \n" << posotita << " του # " << kodarithm << " κοστίζουν " <<
            setprecision( 2) << kostos * posotita;
    cout << " \n\n\n"; // Δύο κενές γραμμές
   }
  while ( kodarithm != -999 );  // Ο βρόγχος συνεχίζεται όσο
                                // ο κωδικός αριθμός δεν είναι -999
  cout << " Τέλος Μηχανικής Απογραφής Εμπορευμάτων \n";
  return(0);
 }
```

4.6.1.3.5 Μετρητές και Σύνολα - Παραγωγή Συνόλων

Στο Κεφάλαιο 2, "Περί Συστηματικού Προγραμματισμού" και ειδικά στα Παραδείγματα 2.2.4, 2.2.5 και 2.2.7, μελετήσαμε τη μεθοδολογία σχεδιασμού και γραφής προγραμμάτων σε C++, στα οποία, αντίστοιχα, καταμετρούνται αγνώστου πλήθους στοιχεία, υπολογίζεται το άθροισμα τους και τέλος εντοπίζεται το στοιχείο (αριθμός) με τη μέγιστη τιμή.

Στην παράγραφο αυτή παραθέτονται κάποια παραδείγματα-προγράμματα που στόχο έχουν την περαιτέρω διεύρυνση των προγραμματικών οριζόντων των μελετητών. Φυσικά, με υπόβαθρο τη μέχρι τώρα γνώση που αποκτήθηκε, γενικά αλλά και ειδικά, από τη μελέτη των προηγούμενων παραγράφων και κεφαλαίων, τις ερωτήσεις, την επίλυση των ασκήσεων που παρατίθενται ή προτείνονται και την υλοποίηση στον Η/Υ των αντίστοιχων προγραμμάτων τους.

Παραδείγματα:

1. Να γραφεί πρόγραμμα που τυπώνει 10 φορές το μήνυμα: "Ο προγραμματισμός είναι ευχάριστος και δημιουργικός". Τα εργαλεία που θα χρησιμοποιηθούν είναι ένας βρόγχος **while** και ένας μετρητής. Ο μετρητής θα καταμετρά τις εκτυπώσεις, ενώ ο βρόχος **while** θα ελέγχει το πλήθος των ανακυκλώσεων έτσι ώστε, μόλις φθάσουν και ξεπεράσουν τον προβλεπόμενο αριθμό Ν, π.χ. Ν = 10, να διακόπτει τη διαδικασία.

```
// EG1KATAMETR.CPP
// Εκτύπωση ενός μηνύματος Ν φορές
#include< iostream.h>
main( )
{
 int metritis = 0;   // Καταμετρητής εκτυπώσεων
 int plithos_minimat ;
 cout << "Καθορίστε το πλήθος των μήνυμάτων που θέλετε να εκτυπωθούν : ";
 cin >> plithos_minimat ;
 do
  {
   cout << " Ο προγραμματισμός είναι ευχάριστος και δημιουργικός ";
   metritis++ ;  // Πρόσθεσε 1 στον μετρητή μετά από κάθε εκτύπωση
  }
 while ( metritis < plithos_minimat ); // Ξαναεκτύπωσε, αν οι εκτυπώσεις είναι
                                       // λιγότερες από το πλήθος των μηνυμάτων
 return(0);
}
```

Έτσι το πρόγραμμα αυτό δεν προσθέτει μόνο μια σταθερή τιμή στον μετρητή, αλλά επίσης, εκτελεί τον βρόγχο συγκεκριμένες φορές. Το πρόγραμμα αυτό αποτελεί μια κοινή μέθοδο για την εκτέλεση υπό συνθήκη τμημάτων ενός προγράμματος για συγκεκριμένο αριθμών ανακυκλώσεων.

2. Παραθέτουμε ένα πρόγραμμα με "συνθηματικό μήνυμα" (password). Το πρόγραμμα αυτό είναι μια κοινή μέθοδος που χρησιμοποιοείται με τη βοήθεια των Η/Υ για τηλεφωνικές κλήσεις, προκειμένου ο χρήστης σχηματίζοντας το σωστό συνθηματικό (password) να πάρει κάποιο "κρυφό" μήνυμα.

Σύμφωνα με την μέθοδο αυτή, δίνεται η δυνατότητα στον χρήστη που καλεί, να μπορεί να προσπαθήσει ορισμένες φορές για να εισάγει το σωστό συνθηματικό, εάν ξεπεραστεί το όριο των περιορισμένων προσπαθειών το τηλέφωνο κλείνει. Με τον τρόπο αυτόν αποτρέπονται οι χρήστες που δοκιμάζουν με ένα τηλεφώνημα πολλά συνθηματικά, μέχρις ότου επιτύχουν το σωστό.

Το "συνθηματικό μήνυμα" έχει αποθηκευθεί σαν μια ακεραία μεταβλητή. Ο χρήστης

πρέπει να εισάγει σωστά το "συνθηματικό μήνυμα", έχοντας περιορισμένα περιθώρια δοκιμών, για τηλεφωνική κλήση.

```
// EG2PASSWORD.CPP
// Πρόγραμμα που ζητά συνθηματικό μήνυμα,
// με περιορισμένα περιθώρια δοκιμών, για τηλεφωνική κλήση
#include< iostream.h>
#include< stdlib.h >
main( )
{
 int apothikevm_pass  = 2117;   // Αποθηκευμένο συνθηματικό μήνυμα
 int orio_prospath = 3;         // Ανώτατο όριο προσπαθειών
 int plithos_prospath = 0;      // Μετρητής προσπαθειών
 int pass_christi ;
 while ( plithos_prospath < orio_prospath ) // Βρόγχος με όριο 3 ανακυκλώσεις
  {
   cout <<"Δώστε το συνθηματικό μήνυμά (password) σας (όριο 3 προσπάθειες)";
   cin << pass_christi ;
   plithos_prospath++ ;  // Πρόσθεσε 1 στον μετρητή
   if ( pass_christi = = apothikevm_pass )
    {
     cout << "Το συνθηματικό μήνυμά (password) σας έγινε αποδεκτό \n";
     cout << "Αναμείνατε \n";
     cout << " Το μυστικό μήνυμα βρίσκεται πίσω από την εικόνα\n";
     exit(0);
    }
   else
    {
     cout << "Το συνθηματικό μήνυμά (password) σας δεν έγινε αποδεκτό \n";
     cout << "Είναι λάθος \n";
     if ( plithos_prospath = = orio_prospath )
      { cout << "Δεν έχετε άλλα περιθώρια δοκιμών εισαγωγής password\n"; }
     else
      { cout << "Ξαναπροσπαθήστε, \n";
        cout << "έχετε ακόμα περιθώρια δοκιμών εισαγωγής password\n"; }
```

```
    }
  }
 exit(0);
 return(0);
}
```

Το πρόγραμμα δίνει στους χρήστες τρείς ευκαιρίες σε περίπτωση που κάνουν λάθη. Μετά από τρείς ανεπιτυχείς προσπάθειες, το πρόγραμμα διακόπτεται χωρίς να εμφανίσει το κρυμμένο μήνυμα.

3. Το πρόγραμμα που παρατίθεται είναι ένα παιχνίδι πρόβλεψης αριθμών ή γραμμάτων. Στο πρόγραμμα περιλαμβάνεται ένα μήνυμα που ενημερώνει τους χρήστες-παίχτες πόσες προσπάθειες έχουν κάνει πριν την πρόβλεψη του τυχερού αριθμού ή γράμματος. Ένας μετρητής καταμετρά το πλήθος αυτών των προσπαθειών.

```
// EG3PROVLEPSI.CPP
// Παιχνίδι πρόβλεψης αριθμού ή γράμματος
#include< iostream.h>
main( )
{
 int dokimes = 0;
 char apandisi_comp,  provlepsi_christi;

 // Αποθήκευση του "τυχερού" αριθμού ή γράμματος
 apandisi_comp = 'R'; // Μπορεί να αλλαχθεί εαν είναι επιθυμητό
 cout << "Σκεφθείτε για την πρόβλεψη του γράμματος . . . \n";
 do
  {
   cout << " Πληκτρολογήστε την πρόβλεψή σας : ";
   cin >> provlepsi_christi ;
   dokimes++; // Μετρητής προβλέψεων
   if ( provlepsi_christi  < apandisi_comp )
    {
     cout << "Το γράμμα που προβλέψατε είναι μετά από το τυχερό γράμμα\n";
     cout << "\n Ξαναπροσπαθήστε . . . ";
    }
   if ( provlepsi_christi  > apandisi_comp )
```

```
        {
         cout << "Το γράμμα που προβλέψατε είναι πριν από το τυχερό γράμμα\n";
         cout << "\n Ξαναπροσπαθήστε ... ";
        }
       }
    while ( proviepsi_christi  != apandisi_comp ); // Διακόπτει όταν βρεθεί

    // Βρέθηκε το ζητούμενο γράμμα
    cout << "***** Συγχαρητήρια κερδίσατε !!!! ";
    cout << " Συνολικά κάνατε " << dokimes << " προσπάθειες\n";
    return(0);
   }
```

Η έξοδος αυτού του προγράμματος είναι:

```
Σκεφθείτε για την πρόβλεψη του γράμματος ...
Πληκτρολογήστε την πρόβλεψή σας : T
Το γράμμα που προβλέψατε είναι μετά από το τυχερό γράμμα

Ξαναπροσπαθήστε ... Πληκτρολογήστε την πρόβλεψή σας : O
Το γράμμα που προβλέψατε είναι πριν από το τυχερό γράμμα

Ξαναπροσπαθήστε ... Πληκτρολογήστε την πρόβλεψή σας : R
***** Συγχαρητήρια κερδίσατε !!!!
Συνολικά κάνατε 3 προσπάθειες
```

4. Το πρόγραμμα που ακολουθεί προσθέτει τους βαθμούς κάποιου σπουδαστή σε ένα από τα μαθήματα που έχει πάρει. Υπάρχει η συμφωνία ότι κάποιος παίρνει άριστα άν έχει φθάσει τα 450 μόρια και πάνω.

Το πρόγραμμα τροφοδοτείται με τους βαθμούς μέχρι που στον TERMAT ανατεθεί η τιμή -1. Ο TERMAT, όπως έχουμε ξαναναφέρει, δίνει το σήμα διακοπής της διαδικασίας, στην προκειμένη περίπτωση εισαγωγής των βαθμών. Τέλος, το πρόγραμμα τυπώνει και συγχαρητήριο μήνυμα σε περίπτωση που ο σπουδαστής συμπληρώσει τα μόρια και πάρει άριστα.

```
// EG4VATHMOLOGIO.CPP
// Προσθέτει βαθμούς και καθαρίζει εάν η βαθμολογία είναι το ΑΡΙΣΤΑ
#include< iostream.h>
```

```
#include< iomanip.h>
main( )
{
 float synoliki_vathmologia  = 0.0;
 float vathmos; // Αποθηκεύει τους βαθμούς
 do
  {
    cout << " Πληκτρολογήστε τον βαθμό σας (Προσοχή TERMAT = -1) : ";
    cin >> vathmos;
    if ( vathmos > = 0.0 )
     { synoliki_vathmologia + = vathmos; } // Πρόσθεσε στο σύνολο
    }
 while ( vathmos > = 0.0 );   // Διακόπτει όταν TERMAT = -1

 // Ο έλεγχος για το άριστα αρχίζει όταν δεν υπάρχουν άλλοι βαθμοί
 cout << "\n\n Το σύνολο της βαθμολογίας σας είναι" << setprecision(1) <<
        synoliki_vathmologia  << "μονάδες\n";
 if ( vathmos > = 450.00 )
    { cout << "***** Συγχαρητήρια πήρατε ΑΡΙΣΤΑ ! ! ! ! "; }
 return(0);
}
```

Προσοχή ! Η τιμή του TERMAT = -1 δεν έχει προστεθεί στη συνολική βαθμολογία, διότι ο έλεγχος γίνεται πριν την πρόσθεση.

Η έξοδος αυτού του προγράμματος είναι:

```
Πληκτρολογήστε τον βαθμό σας (Προσοχή TERMAT = -1) : 92.3
Πληκτρολογήστε τον βαθμό σας (Προσοχή TERMAT = -1) : 87.7
Πληκτρολογήστε τον βαθμό σας (Προσοχή TERMAT = -1) : 96
Πληκτρολογήστε τον βαθμό σας (Προσοχή TERMAT = -1) : 85
Πληκτρολογήστε τον βαθμό σας (Προσοχή TERMAT = -1) : 86

Το σύνολο της βαθμολογίας σας είναι 477.0 μονάδες
***** Συγχαρητήρια πήρατε ΑΡΙΣΤΑ ! ! ! !
```

5. Διατυπώνουμε πρόγραμμα που υπολογίζει τον μέσο όρο των βαθμών κάποιου σπουδαστή σε ένα από τα μαθήματα που έχει πάρει.

Στην πραγματικότητα το πρόγραμμα αυτό είναι επέκταση του προηγούμενου, αφού για να υπολογιστεί ο μέσος όρος των βαθμών θα πρέπει να προστεθούν οι βαθμοί αλλά και να καταμετρηθούν · αφού ο μέσος όρος Ν αριθμών, ισούται με το πηλίκο του αθροίσματος των αριθμών διά του πλήθους των.

```
// EG5MESORVATHMOLOG.CPP
// Προσθέτει βαθμούς, υπολογίζει τον μέσο όρο και
// καθορίζει αν είναι ΑΡΙΣΤΑ
#include< iostream.h>
#include< iomanip.h>
main( )
{
 float synoliki_vathmologia  = 0.0;
 float mesor_vathmologias  = 0.0;
 float vathmos;
 int plithos_ vathm;

 do
  {
   cout << " Πληκτρολογήστε τον βαθμό σας (Προσοχή TERMAT = -1) : ";
   cin >> vathmos;
   if ( vathmos > = 0.0 )
    { synoliki_vathmologia + = vathmos;  // Πρόσθεσε στο Σύνολο
      plithos_ vathm ++; }               // Πρόσθεσε στον Μετρητή
   }
 while ( vathmos > = 0.0 );   // Διακόπτει όταν TERMAT = -1

 // Ο έλεγχος για το άριστα αρχίζει όταν δεν υπάρχουν άλλοι βαθμοί
 // Υπολογισμός Μέσου Όρου
 mesor_vathmologias = ( synoliki_vathmologia / plithos_ vathm );
 cout << "\n\n Το σύνολο της βαθμολογίας σας είναι" << setprecision(1) <<
       synoliki_vathmologia  << "μονάδες\n";
```

```
    cout << "Ο μέσος όρος της βαθμολογίας σας είναι : " << mesor_vathmologias <<
            "\n";
    if ( vathmos > = 450.00 )
      { cout << "***** Συγχαρητήρια πήρατε ΑΡΙΣΤΑ ! ! ! ! ";}
    return(0);
  }
```

Προσοχή ! Η τιμή του TERMAT = -1 δεν έχει προστεθεί στη συνολική βαθμολογία, διότι ο έλεγχος γίνεται πριν την πρόσθεση.

Η έξοδος αυτού του προγράμματος είναι:

```
Πληκτρολογήστε τον βαθμό σας (Προσοχή TERMAT = -1) : 92.3
Πληκτρολογήστε τον βαθμό σας (Προσοχή TERMAT = -1) : 87.7
Πληκτρολογήστε τον βαθμό σας (Προσοχή TERMAT = -1) : 96
Πληκτρολογήστε τον βαθμό σας (Προσοχή TERMAT = -1) : 85
Πληκτρολογήστε τον βαθμό σας (Προσοχή TERMAT = -1) : 86

Το σύνολο της βαθμολογίας σας είναι 477.0 μονάδες
Ο μέσος όρος της βαθμολογίας σας είναι : 95.40
***** Συγχαρητήρια πήρατε ΑΡΙΣΤΑ ! ! ! !
```

4.6.1.3.6 Ερωτήσεις για Επανάληψη και Εμπέδωση

1. Δουλέψτε με μολύβι και χαρτί τους αναδρομικούς κανόνες - ορισμούς της σύνταξης των προτάσεων **while** και **do_while** , συμπληρώσατέ τους αν χρειαστεί.

2. Ποιά είναι η διαφορά μεταξύ του βρόγχου **while** και **do_while** ;

3. Ποιά είναι η διαφορά μεταξύ μιας μεταβλητής Συνόλου και μιας μεταβλητής Μετρητή;

4. Ποιός τελεστής της C++ είναι πιό χρήσιμος για μέτρηση;

5. Είναι αληθές ή ψευδές ότι, τα άγγιστρα δεν απαιτούνται στο <σωμα> των βρόγχων **while** και **do_while** ;

6. Τι γίνεται με τον κώδικα που ακολουθεί;

```
while ( poliseis > 50 )
  cout << "Οι πωλήσεις σου πήγαν πολύ καλά αυτόν τον μήνα \n";
```

cout << "Θα πάρεις και δώρο για την πολύ καλή επίδοσή σου αυτόν τον μήνα \n";

7. Ποιά πρόταση πρέπει να συμπεριλάβετε στον κώδικά σας εάν χρησιμοποιήσετε την exit( );

8. Πόσες φορές τυπώνει η πρόταση που περιέχει τον τελεστή cout;

```
int a = 0;
do
  { cout << " Προσοχή ! \n";
    a++; }
while (a > 5);
```

9. Πώς μπορείτε να πληροφορήσετε το DOS για την κατάσταση εξόδου του προγράμματός σας.

10. Τι τυπώνεται στην οθόνη στο ακόλουθο τμήμα κώδικα;

```
a = 1;
while ( a < 4 )
 {
   cout << Αυτός είναι ο Εξωτερικός Βρόγχος\n";
   a++;
   while ( a < = 25 )
      { break;
        cout << "Αυτό τυπώνεται 25 φορές\n";
      }
}
```

4.6.1.4 Η Πρόταση - Βρόγχος for

Η πρόταση **for** είναι μια από τις προτάσεις που δομούν συγκροτήματα από προτάσεις και οι οποιες θεωρούνται σαν μια πρόταση. Η πρόταση **for** επιτρέπει να επαναλαμβάνονται τμήματα του προγράμματος (<σωμα>) για ένα συγκεκριμένο αριθμό ανανακυκλώσεων (βρόγχων). Σε αντίθεση με τις προτάσεις **while** και **do-while** οι οποίες δίνουν τη δυνατότητα στην ενότητα του προγράμματος (<σωμα>) να επαναλαμβάνεται αγνώστου πλήθους φορές, δηλαδή, ο βρόγχος να επαναλαμβάνεται μέχρις ότου η <συνθηκη> που τον ελέγχει πάψει να ισχύει.

Η σύνταξη της **for** ακολουθεί τους εξής κανόνες οι οποίοι διατυπώνονται με τη βοήθεια της BNF:

1.<προταση_for> ::= for (<αρχικη_εκφραση> ;<εκφραση_ελεγχου> ;<εκφραση_μετρητη>)

{<σωμα>} ,

2. <αρχικη_εκφραση> ::= <προσδιοριστης_μετρητη> = <αρχικη_τιμη> ,

3. <εκφραση_ελεγχου> ::= <προσδιοριστης_μετρητη>
<σχεσιακος_τελεστης><τελικη_τιμη> ,

4. <σχεσιακος_τελεστης> ::= > | >= | < | <= ,

5. <εκφραση_μετρητη> ::= <προσδιοριστης_μετρητη> =
<προσδιοριστης_μετρητη><αριθμητικος_τελεστης><βημα_αυξησης> ,

6. <αριθμητικος_τελεστης> ::= + | - ,

7. <σωμα> ::= <συνθετη_προταση> , (πρβλ κανόνες §4.6.1.1),

Για την καλλίτερη κατανόηση του ορισμού της **for** παραθέτουμε το ακόλουθο παράδειγμα.
(α) Δίδεται ένα πρόγραμμα που χρησιμοποιεί τον βρόγχο **for** για να:

1. προσθέτει 1(ένα) στον μετρητή,
2. τυπώνει την τιμή του και
3. επαναλάβει το βήμα 1, αν ο μετρητής είναι μικρότερος ή ίσος του 10.

```
. . .
// Εισαγωγή στον βρόγχο for
// Εύρεση συνόλου με βρόγχο for
#include <iostream.h>
main( )
{
 int metritis ;
 for ( metritis = 1; metritis <= 10; metritis++ )  // Αρχίζει με metritis = 1
                                                   // Προσαυξηση μέσα στον βρόγχο
   { cout << metritis << "\n"; }                   // <σωμα> του βρόγχου for
 return(0);
}
```

Το πρόγραμμα παράγει ως έξοδο τα εξής:

```
1
2
3
4
```

5
6
7
8
9
10

Υπόδειξη 1 Το σώμα (<σωμα>) της πρόταση **for** είναι γραμμένο πιό μέσα από το υπόλοιπο πρόγραμμα. Με τον τρόπο αυτόν, που είναι μια πολύ καλή συνήθεια, γίνεται ευκολοδιάκριτο το σώμα (<σωμα>) της **for**.

(β) Για καθαρά εκπαιδευτικούς λόγους, παραθέτουμε το ίδιο πρόγραμμα αλλά χρησιμοποιώντας την πρόταση **do-while** .

```
. . .
// Εύρεση συνόλου με βρόγχο do-while
#include <iostream.h>
main( )
{
 int metritis = 1;
 do
  { cout << metritis << "\n";   // <σωμα> του βρόγχου do-while
    metritis++ ; }
 while (metritis <= 10);
 return(0);
}
```

Από τα δύο παραδείγματα (α) και (β) που προηγήθηκαν παρατηρούμε ότι η πρόταση **for** είναι ο καλλίτερος τρόπος για τον έλεγχο της διεργασίας του βρόγχου γνωστού πλήθους ανακυκλώσεων. Η διαδικασία αυτή εκτελείται με λιγότερες προτάσεις από αυτές που απαιτεί ένας βρόγχος **while** .

Με τη **for** δηλαδή, δεν χρειάζονται οι επιπλέον κωδικοί για τον καθορισμό αρχικών τιμών στις μεταβλητές - μετρητές, καθώς και οι κωδικοί προσαύξησης ή μείωσης τους, αφού αυτοί είναι ενσωματωμένοι στη **for**, εγγράφονται, όπως είναι γνωστό, μέσα στην παρένθεση μαζί με τη συνθήκη τερματισμού του βρόγχου (βλπ (α) παράδειγμα) :

```
. . .
for ( metritis = 1; metritis <= 10; metritis++ )
```

. . .

Αντίθετα, με την **do-while** όπου οι αντίστοιχοι κωδικοί εμφανίζονται σε τρία διαφορετικά σημεία (βλπ (β) παράδειγμα) :

```
. . .
int metritis = 1;
do
 { . . .
   metritis++ ; }
while (metritis <= 10);
. . .
```

μόνο η συνθήκη τερματισμού του βρόγχου, όπως παρατηρούμε, είναι ενσωματωμένη στην πρόταση **do-while**.

Παρατηρήσεις:

1. Η C++ υλοποιεί την αρχική έκφραση (<αρχικη_εκφραση>) πριν αρχίσει ο βρόγχος, δηλαδή στην αρχή του βρόγχου και μόνο μια φορά. Η αρχική έκφραση (<αρχικη_εκφραση>) είναι συνήθως μια πρόταση ανάθεσης, μπορεί όμως να είναι και μια οποιαδήποτε έγκυρη πρόταση που θα οριστεί από τον προγραμματιστή και θα οδηγεί στον καθορισμό της τιμής της αρχικής έκφρασης (<αρχικη_εκφραση>).
2. Γενικά δεν υπάρχει οριοθέτης " ; " μετά την δεξιά παρένθεση της **for**. Σε αντίθετη περίπτωση, δηλαδή που πληκτρολογηθεί από λάθος, ερμηνεύεται σαν το σώμα (<σωμα>) του **for** να στερείται προτάσεων, δηλαδή κενή **for**, με αποτέλεσμα ο βρόγχος να συνεχίζεται μέχρις ότου η έκφραση ελέγχου (<εκφραση_ελεγχου>) γίνει ψευδής, χωρίς όμως να υπολογίζει τίποτα από τις προτάσεις του σώματος (<σωμα>), οτιδήποτε και αν το σώμα (<σωμα>) περιέχει. Από τον κανόνα εξαιρούνται οι βρόγχοι αναμονής οι οποίοι μπορεί να είναι και κενές προτάσεις **for**, δηλαδή προτάσεις **for** χωρίς σώμα (<σωμα>) (βλπ §4.6.1.4.1).
3. Κάθε φορά που επαναλαμβάνεται το σώμα (<σωμα>) του **for**, εκτελείται η έκφραση μετρητής (<εκφραση_μετρητη>), συνήθως αυξάνοντας ή μειώνοντας μια μεταβλητή που εκφράζεται από τον προσδιοριστή του μετρητή.
4. Η έκφραση ελέγχου (<εκφραση_ελεγχου>) ελέγχει κάθε φορά τη νέα τιμή του μετρητή στην νέα ανακύκλωση και μετά αποφασίζει (ναί ή όχι) αν το σώμα (<σωμα>) της ανακύκλωσης θα εκτελεστεί.

5. Αν το σώμα (<σωμα>) της **for** αποτελείται από μια μόνο πρόταση οι αγκύλες δεν απαιτούνται, αλλά ως συνήθως προτείνονται. Αν όμως προστεθούν και άλλες προτάσεις στο σώμα (<σωμα>) της **for**, τότε οι αγκύλες, που ήδη υπάρχουν, απαιτούνται.

Παραδείγματα:

1. Να γραφούν δύο προγράμματα που προσθέτουν αριθμούς από το 100 μέχρι το 200. Στο πρώτο πρόγραμμα να χρησιμοποιείται ένας βρόγχος **for**, ενώ στο δεύτερο ένας **do-while**.

(α) . . .

```
// Εύρεση συνόλου με βρόγχο for και εκτύπωση αποτελέσματος
#include <iostream.h>
main( )
{
 int athrisma, arithm;
 athrisma = 0;
 for ( arithm = 100; arithm <= 200; arithm++ )    // Ο arithm παίρνει τις τιμές 100,
                                                  // 101, 102, 103, ... , 200
   { athrisma += arithm;}                         // Προσθέτει την τιμή της arithm
                                                  // σε κάθε επανάληψη
  cout << " Το άθροισμα είναι :" << athrisma << "\n";
 return(0);
}
```

(β) . . .

```
// Εύρεση συνόλου με βρόγχο do-while και εκτύπωση αποτελέσματος
#include <iostream.h>
main( )
{
 int athrisma = 0;                  // Ορισμός αρχικής τιμής της athrisma
 int arithm = 100;                  // Ορισμός αρχικής τιμής της arithm
 do
  { athrisma += arithm;             // Πρόσθεση στην athrisma
    arithm++; }                     // Μετρητής προσαύξησης
 while (arithm <= 200);
 cout << " Το άθροισμα είναι : " << athrisma << "\n";
 return(0);
```

```
}
```

Όπως είναι φυσικό και τα δύο προγράμματα παράγουν την ίδια έξοδο:

Το άθροισμα είναι : 15150

Το κύριο μέρος του βρόγχου και στα δύο προγράμματα εκτελείται 101 φορές. Η αρχική τιμή της arithm είναι 100. Επίσης, παρατηρούμε ότι ο βρόγχος **for** είναι λιγότερο πολύπλοκος, όπως είδαμε και παραπάνω, από τον αντίστοιχο βρόγχο **do-while**.

2. Να γραφεί πρόγραμμα που διαβάζει πέντε ζεύγη τιμών δεδομένων: τα ονόματα των παιδιών και τις ηλικίες τους. Τυπώνει το όνομα του δασκάλου που έχει αναλάβει το κάθε παιδί, ανάλογα με την ηλικία του.

```
. . .
// Βρόγχος ανάθεσης και εκτύπωσης το όνομα ενός καθηγητή για κάθε μαθητή
#include <iostream.h>
main( )
{
 char paidi[25];  // Αποθηκεύει το όνομα κάθε παιδιού
 int ilikia;      // Αποθηκεύει την ηλικία κάθε παιδιού
 int metritis;    // Μετρητής βρόγχου for

 for ( metritis = 1; metritis <= 5; metritis++ )
  {
    cout << " Δώστε το όνομα του επόμενου μαθητή : ";
    cin >> paidi;
    cout << " Δώστε την ηλικία του μαθητή : ";
    cin >> ilikia;
    if ( ilikia <= 5 )
      { cout << "\n" << paidi << " , έχει την κα "
             << " Πολυαυστηρή για δασκάλα \n"; }
    if ( ilikia == 6 )
      { cout << "\n" << paidi << " , έχει τη διδα "
             << " Παραπολικαλή για δασκάλα \n"; }
    if ( ilikia >= 7 )
      { cout << "\n" << paidi << " , έχει τον κο "
             << " Φωνάρα για δάσκαλο \n"; }
  }   // Η διαδικασία διακόπτεται μετά από 5 ανακυκλώσεις
return(0);
```

```
}
```

Δίδεται αμέσως μετά η έξοδος αυτού του προγράμματος. Το πρόγραμμα μπορεί να βελτιωθεί ακόμα περισσότερο με τη χρήση της πρότασης **switch** που παρουσιάζεται παρακάτω στην § 4.6.1.5.1.

```
Δώστε το όνομα του επόμενου μαθητή : Amvrossios
Δώστε την ηλικία του μαθητή : 5

Amvrossios , έχει την κα Πολυαυστηρή για δασκάλα
Δώστε το όνομα του επόμενου μαθητή : Meletios
Δώστε την ηλικία του μαθητή : 6

Meletios , έχει τη διδα Παραπολυκαλή για δασκάλα
Δώστε το όνομα του επόμενου μαθητή : Ignatios
Δώστε την ηλικία του μαθητή : 9

Ignatios , έχει τον κο Φωνάρα για δάσκαλο
Δώστε το όνομα του επόμενου μαθητή : Pelagia
Δώστε την ηλικία του μαθητή : 6

Pelagia , έχει την διδα Παραπολυκαλή για δασκάλα
Δώστε το όνομα του επόμενου μαθητή : Glykeria
Δώστε την ηλικία του μαθητή : 5

Glykeria , έχει την κα Πολυαυστηρή για δασκάλα
```

3α. Να γραφεί ένα πρόγραμμα που χρησιμοποιεί τον βρόγχο **for** για να **τυπώσει** όλους τους άρτιους ακεραίους και μετά όλους τους περιττούς ακέραιους αριθμούς από το 1 έως το 20.

```
. . .
// Εκτυπώνει άρτιους από 1έως 20
// Εκτυπώνει περιττούς από 1έως 20
#include <iostream.h>
main( )
{
 int arithm;                                  // Μεταβλητή-μετρητής βρόγχου for
 cout << " Οι άρτιοι οι μικρότεροι του 21" << "\n"; // Τίτλος
 for ( arithm = 2; arithm <= 20; arithm+=2 )
  { cout << arithm << " "; }                  // Τυπώνει κάθε δεύτερο αριθμό
 cout << " Οι περιττοί οι μικρότεροι του 20" << "\n"; // Δεύτερος τίτλος
```

```
 for ( arithm = 1; arithm <= 20; arithm+=2 )
  { cout << arithm << " "; }                    // Τυπώνει κάθε δεύτερο αριθμό
 return(0);
}
```

3β. Να γραφεί ένα πρόγραμμα που χρησιμοποιεί τον βρόγχο **for** για να **υπολογίσει και να τυπώσει** όλους τους άρτιους ακεραίους και μετά όλους τους περιττούς ακέραιους αριθμούς από το 1 έως το 20.

```
. . .
// Υπολογίζει και εκτυπώνει τους άρτιους από 1έως 20
// Υπολογίζει και εκτυπώνει τους περιττούς από 1έως 20
#include <iostream.h>
main( )
{
 int arithm;                                    // Μεταβλητή-μετρητής βρόγχου for
 cout << " Οι άρτιοι οι μικρότεροι του 21 :" << "\n"; // Τίτλος
 for ( arithm = 1; arithm <= 100; arithm++ )
  {
   if ( 2 * (arithm / 2) = arithm )
     { cout << arithm << " "; }                 // Τυπώνει κάθε δεύτερο αριθμό
 cout << "\n Οι περιττοί οι μικρότεροι του 20 :" << "\n"; // Δεύτερος τίτλος
 for ( arithm = 1; arithm <= 100; arithm++ )
  {
   if ( 2 * (arithm / 2) != arithm )
     { cout << arithm << " "; }                 // Τυπώνει κάθε δεύτερο αριθμό
  }
 return(0);
}
```

Τα προγράμματα 3α και 3β παράγουν την ίδια έξοδο :

```
Οι άρτιοι οι μικρότεροι του 21 :
2 4 6 8 10 12 14 16 18 20

Οι περιττοί οι μικρότεροι του 20 :
1 3 5 7 9 11 13 15 17 19
```

4. Να γραφούν δύο προγράμματα που πραγματοποιούν την αντίστροφη μέτρηση

ξεκινώντας από το 10. Στο πρώτο πρόγραμμα να χρησιμοποιείται ένας βρόγχος **for**, ενώ στο δεύτερο ένας **do-while**.

Σημείωση: Στην πραγματικότητα το παραπάνω πρόβλημα παράγει το αντίθετο αποτέλεσμα από αυτό που παράγει το πρόβλημα της μέτρησης. Υπενθυμίζουμε ότι το πρόβλημα της μέτρησης παρουσιάστηκε στην αρχή της παραγράφου, ακριβώς μετά τον ορισμό της πρότασης **for**, ως παράδειγμα εφαρμογής της πρότασης αυτής.

(α) . . .

```
// Αντίστροφη μέτρηση με βρόγχο for
#include <iostream.h>
main( )
{
 int metritis;
 for ( metritis = 10; metritis != 0; metritis-- )
  { cout << metritis << "\n"; }  // Εκτυπώνει την metritis καθώς μειώνεται
 cout << " ΠΥΡΙΙΙ \n";
 return(0);
}
```

(β) . . .

```
// Αντίστροφη μέτρηση με βρόγχο do-while
#include <iostream.h>
main( )
{
 int metritis = 10;
 do
  { cout << metritis << "\n";   // <σωμα> του βρόγχου do-while
                                // Εκτυπώνει την metritis καθώς μειώνεται
    metritis-- ; }
 while (metritis != 0);
 cout << " ΠΥΡΙΙΙ \n";
 return(0);
}
```

Οι δύο μορφές (α) και (β) του ιδίου προγράμματος παράγουν ως έξοδο τα εξής:

```
9
8
7
6
5
4
3
2
1
ΠΥΡΙΙΙ
```

Στις δυο μορφές (α) και (β) του προγράμματος για τον υπολογισμό της αντίστροφης μέτρησης, παρατηρούμε ότι μπορούμε να μειώσουμε τη μεταβλητή-μετρητή του βρόγχου. Κατά τη διαδικασία μείωσης η σταθερή τιμή, στο παράδειγμά μας η σταθερή τιμή είναι 1, αφαιρείται από τη μεταβλητή-μετρητή του βρόγχου κάθε φορά μέσω του βρόγχου, π.χ. metritis--, το οποίο όπως είναι γνωστό ισοδυναμεί με metritis = metritis - 1. Είναι προφανές ότι στις περιπτώσεις μείωσης, της μεταβλητής-μετρητή του βρόγχου, η αρχική τιμή της θα πρέπει να είναι μεγαλύτερη από την τελική τιμή της που ελέγχεται, π.χ. metritis = 10 (αρχική τιμή) και metritis ! = 0 (τελική τιμή που ελέγχεται).

5. Στο πρόγραμμα που ακολουθεί συνδυάζονται και αξιοποιούνται πολλά από αυτά που έχουν μέχρι τώρα παρουσιαστεί. Έτσι το πρόγραμμα υπολογίζει τον μέσο όρο των εξαμηνιαίων βαθμών των μαθητών κάποιας τάξης. Δηλαδή το πρόγραμμα:

1. Διαβάζει το πλήθος των μαθητών και τον εξαμηνιαίο βαθμό για κάθε μαθητή,
2. Υπολογίζει τον μέσο όρο των δοθέντων βαθμών,
3. Τυπώνει τον μέσο όρο των δοθέντων βαθμών.

```
. . .
// Υπολογίζει μέσο όρο με βρόγχο for
#include <iostream.h>
#include <iomanip.h>
main( )
{
 float vathmos, mesoros;
 float athrisma = 0.0;
 int plithmathit ;          // Αρχικό πλήθος μαθητών-βαθμών
 int metritis ;             // Μετρητής για τον έλεγχο των βρόγχων
```

```
cout << " \n          """Υπολογισμός Βαθμολογίας """          \n\n"; // Τίτλος
cout << " Δώστε το πλήθος των μαθητών : ";
cin >> plithmathit ;              // Αρχικό πλήθος μαθητών-βαθμών
cout << " \n";
for ( metritis = 1; metritis <= plithmathit; metritis++ )
 {
  cout << " \n Δώστε τον εξαμηνιαίο βαθμό του επόμενου μαθητή : ";
  cin >> vathmos;
  athrisma += vathmos;
 }
mesoros = athrisma / plithmathit;
cout << "\n\n Ο μέσος όρος των εξαμηνιαίων βαθμών της τάξης είναι : " <<
      setprecision(1) << mesoros;
return(0);
}
```

6. Να γραφεί πρόγραμμα που τυπώνει τα κεφαλαία γράμματα του λατινικού αλφαβήτου.

```
. . .
// Εκτυπώνει το λατινικό αλφάβητο με βρόγχο for
#include <iostream.h>
main( )
{
 char gramma;
 cout << " Το λατινικό αλφάβητο είναι : " \n";
 for ( gramma = 'A'; gramma <= 'Z'; gramma++ ) // Βρόγχος από Α έως Ζ
  { cout << " " << gramma; }
 return(0);
}
```

Το πρόγραμμα παράγει την ακόλουθη έξοδο:

```
Το λατινικό αλφάβητο είναι :
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
```

7. Η ενότητα αυτή αποτελείται από επιμέρους παραδείγματα, στα οποία η for εκφράζεται χωρίς καμμία από τις τρείς εκφράσεις-συνθήκες: αρχική έκφραση (<αρχικη_εκφραση>),

έκφραση ελέγχου (<εκφραση_ελεγχου>) και έκφραση μετρητή (<εκφραση_μετρητη>) ή μόνο με την έκφραση ελέγχου (<εκφραση_ελεγχου>). Έτσι έχουμε:

7.1 Όταν η πρόταση **for** εκφράζεται χωρίς καμμία από τις τρείς εκφράσεις-συνθήκες: αρχική έκφραση (<αρχικη_εκφραση>), έκφραση ελέγχου (<εκφραση_ελεγχου>) και έκφραση μετρητή (<εκφραση_μετρητη>), λέμε ότι η **for** έχει κενή ή μηδενική έκφραση. Η πρόταση **for** με κενή ή μηδενική έκφραση, γενικά έχει τη μορφή:

```
for ( ; ; )
   { <σωμα>; }
```

Ο βρόγχος της **for** με κενή ή μηδενική έκφραση επαναλαμβάνεται συνέχεια. Τέτοιοι βρόγχοι πρέπει να αποφεύγονται εν γένει στον προγραμματισμό, εκτός και εάν το υπό κατασκευή πρόγραμμα υπαγορεύει την ύπαρξή τους.

Παραθέτουμε το μέρος προγράμματος που περιέχει την **for** με κενή ή μηδενική έκφραση:

```
. . .
for ( ; ; )
    { cout << " ΑΝΑΜΕΙΝΑΤΕ, . . . ΚΑΝΤΕ ΥΠΟΜΟΝΗ, . . . \n"; }
. . .
```

7.2 Το παρακάτω πρόγραμμα παραλείπει την αρχική έκφραση (<αρχικη_εκφραση>) και την έκφραση μετρητή (<εκφραση_μετρητη>), αφήνοντας μόνο την έκφραση ελέγχου (<εκφραση_ελεγχου>) του βρόγχου **for**, δηλαδή την αντίστοιχη της συνθήκης (<συνθηκη>) της πρότασης **while** ή της ισοδύναμης πρότασης **do-while**. Συνήθως στην πράξη μπορεί να παραλειφθεί μόνο μια από τις εκφράσεις αυτές.

Παρατήρηση: Στην πρίπτωση που χρησιμοποιηθεί ο βρόγχος **for** χωρίς δύο από τις εκφράσεις του, παρά μόνο με την έκφραση ελέγχου (<εκφραση_ελεγχου>), τότε μήπως θα ήταν ορθότερο (βάσει του ορισμού της κανονικής βαθμίδας ανακύκλωσης) να χρησιμοποιηθεί ο βρόγχος **while** ή **do-while** ;

Το πρόγραμμα που έπεται χρησιμοποιεί μόνο την κατάλληλη έκφραση ελέγχου (<εκφραση_ελεγχου>) στον βρόγχο **for** για μέτρηση σε πεντάδες των πρώτων 100 ακεραίων αριθμών.

```
. . .
// Χρησιμοποιεί την έκφραση ελέγχου(<εκφραση_ελεγχου>)
// στον βρόγχο for για μέτρηση σε πεντάδες
#include <iostream.h>
main( )
{
 int arithm = 5;       // Μεταβλητή-μετρητής βρόγχου for και <αρχικη_εκφραση>
 cout << " Μετράντας κατά πεντάδες : \n"; // Τίτλος
 for ( ; arithm <= 100; )    // Περιέχει μόνο την <εκφραση_ελεγχου>
  {
   cout << arithm << " ";
   arithm += 5;   // <εκφραση_μετρητη> έξω από τις εκφράσεις-συνθήκες του for
  }               // Τέλος σώματος (<σωμα>) βρόγχου for
 return(0);
}
```

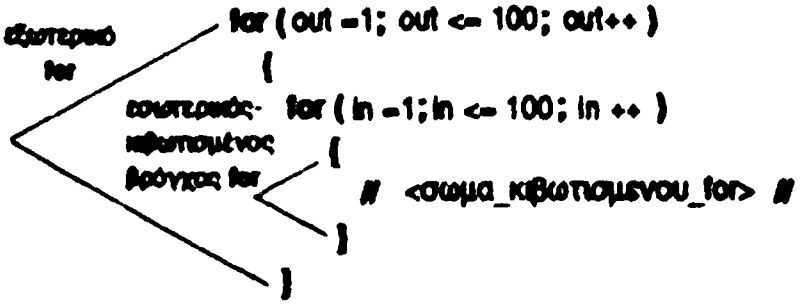
Η έξοδος του προγράμματος έχει ως ακολούθως:

```
Μετράντας κατά πεντάδες :
5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95 100
```

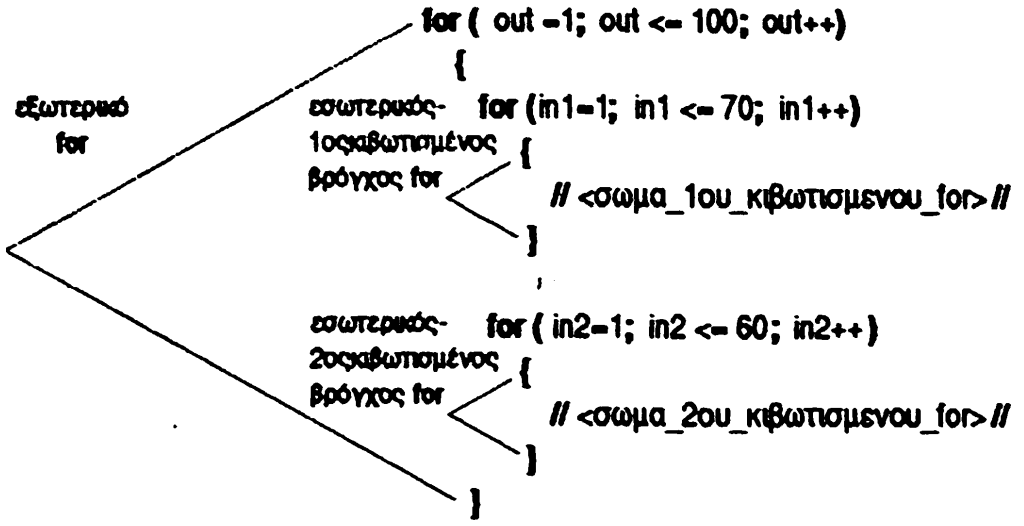
4.6.1.4.1 Κιβωτισμένοι βρόγχοι for

Το σώμα (<σωμα>) μιας πρότασης for ενδέχεται να περιέχει μια ή περισσότερες προτάσεις for, έτσι δημιουργείται ένας κιβωτισμός προτάσεων for. Παραθέτουμε τα παρακάτω προγραμματικά σχήματα:

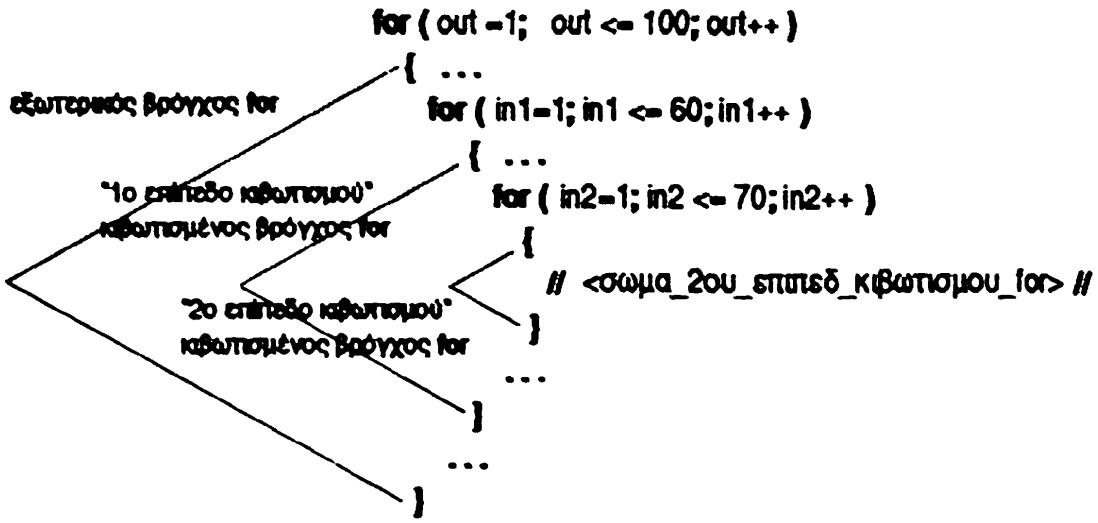
(α) Μια πρόταση for περιέχει κιβωτισμένη μια άλλη πρόταση for :



(β) Μια πρόταση **for** περιέχει κιβωτισμένες περισσότερες από μια προτάσεις **for**:



Είναι ακόμα δυνατόν, να έχουμε περισσότερα του ενός επίπεδα κιβωτισμένων προτάσεων **for**, δηλαδή στο σώμα (<σωμα>) της κιβωτισμένης **for** να περιέχεται μια πρόταση **for** της οποίας το σώμα (<σωμα>) περιέχει μια πρόταση **for** κ.ο.κ. Για παράδειγμα έχουμε:



Παρατήρηση: Υπενθυμίζεται ότι ένα δομημένο πρόγραμμα αποτελείται από ακολουθία απλών προτάσεων και κανονικών βαθμίδων επιλογής και ανακύκλωσης. Οι **κανονικές βαθμίδες μπορεί να είναι κιβωτισμένες ή μη, απαραιτήτως** όμως, **με συδιάκριτα τα επίπεδα κιβωτισμού**, για να είναι **εύκολο στον έλεγχο και εντοπισμό τυχών**

σφαλμάτων αλλά και ευκολοσυντηρούμενο (βλπ παρατήρηση 2, π.χ. 2.2.1, §2.2)

Σημείωση: Στους κιβωτισμένους βρόγχους ο εσωτερικός βρόγχος (ή βρόγχοι), όπως είναι λογικό, εκτελείται πλήρως πριν από την κάθε επόμενη επανάληψη του εξωτερικού βρόγχου. Αυτή είναι και η αιτία που εμφανίζεται, στην πράξη, ο εξωτερικός βρόγχος να "κινείται" πιό αργά από κάθε κιβωτισμένο του, γεγονός που αξιοποιείται στους βρόγχους αναμονής (βλπ §4.6.1.5.1).

Παραδείγματα:

1. Να γραφεί πρόγραμμα που μετράει και τυπώνει από το 1μέχρι το 5, τρείς φορές.

```
. . .
// Εκτυπώνει τους αριθμούς 1- 5 τρείς φορές
// με κιβωτισμένο βρόγχο for
#include <iostream.h>
main( )
{
 int metrita, arithm; // Μεταβλητές-μετρητές εξωτερικού και
                      // κιβωτισμένου βρόγχου for, αντίστοιχα
 for ( metrita = 1; metrita <= 3; metrita++ )
  {
    for ( arithm = 1; arithm <= 3; arithm++ )
      { cout << arithm << " " << " \n"; }  // Σώμα κιβωτισμένου βρόγχου
   cout << " \n";
  }                                        // Τέλος εξωτερικού βρόγχου
 return(0);
}
```

Το πρόγραμμα παράγει την ακόλουθη έξοδο:

```
1 2 3 4 5

1 2 3 4 5

1 2 3 4 5
```

2.Να γραφεί πρόγραμμα που μετράει και τυπώνει αντίστροφα (outer= 5; outer >=1; outer–)

μια μεταβλητή (inner) η οποία είναι μεταβλητή μέτρησης κιβωτισμένου βρόγχου **for**, του οποίου η έκφραση ελέγχου (<εκφραση_ελεγχου>) εξαρτάται από την μεταβλητή μέτρησης του εξωτερικού βρόγχου (inner=1; inner <= outer ; inner++).

```
. . .
// Κιβωτισμένος βρόγχος for που ελέγχεται από
// μεταβλητή ελέγχου εξωτερικού βρόγχου for
#include <iostream.h>
main( )
{
 int outer, inner; // Μεταβλητές-μετρητές εξωτερικού και
                   // κιβωτισμένου βρόγχου for, αντίστοιχα
 for ( outer = 5; outer >= 1; outer-- )
  {
    for ( inner = 1; inner <= outer; inner++ )
      { cout << inner << " " << " \n"; }  // Σώμα κιβωτισμένου βρόγχου
   cout << " \n";
  }                                       // Τέλος εξωτερικού βρόγχου
 return(0);
}
```

Το πρόγραμμα παράγει την ακόλουθη έξοδο:

```
1 2 3 4 5

1 2 3 4

1 2 3

1 2

1
```

Προσοχή ! Η πρόταση **for** είναι η πρόταση που υλοποιεί άμεσα αυτό που συμβολίζει το **"μαθηματικό σύμβολο του αθροίσματος Σ"**, δηλαδή το άθροισμα συγκεκριμένου πλήθους όρων. Για παράδειγμα η μαθηματική έκφραση:

$$\sum_{i=1}^{N} (1/3.2)$$

ισοδυναμεί με το ακόλουθο, κωδικοποιημένο σε C++, προγραμματικό σχήμα:

```
. . .
int i, n;
float athroisma;
. . .
for ( i = 1; i <= n; i++ )
  { athroisma += (i / 3 * 2); }
. . .
```

3. Να γραφεί πρόγραμμα που διαβάζει έναν αριθμό, ακέραιο θετικό ή μηδέν, υπολογίζει και τυπώνει το παραγοντικό του αριθμού αυτού. Υπενθυμίζουμε ότι, το **παραγοντικό αριθμού** $n \in Z_0^+$ ορίζεται από τους αναδρομικούς τύπους:

$$0! = 1, \quad 1! = 1 \quad \text{και} \quad n! = (n-1)! \cdot n = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot (n-1) \cdot n.$$

```
. . .
// Υπολογίζει το παραγοντικό ακεραίου θετικού ή μηδέν
#include <iostream.h>
main( )
{
 int metritis, arithm, parag, paragontiko;
 cout << " Δώστε τον αριθμό - προσοχή! ακέραιο θετικό ή μηδέν - πουλι";
 cout << " θα θέλατε να υπολογιστεί το παραγοντικό του; : ";
 cin >> arithm;
 for ( metritis = 1; metritis <= arithm; metritis++ )
  {
    paragontiko = 1; // Καθορισμός αρχικής τιμής για κάθε παραγοντικό
    for ( parag = 1; parag <= metritis; parag++ )
      { paragontiko *= parag; }  // Υπολογισμός παραγοντικού
  }
 cout << "\nΤο παραγοντικό του αριθμού " << arithm << "είναι :"<< paragontiko;
 return(0);
}
```

Το πρόγραμμα παράγει την ακόλουθη έξοδο:

```
Δώστε τον αριθμό - προσοχή! ακέραιο θετικό ή μηδέν - που
θα θέλατε να υπολογιστεί το παραγοντικό του; : 7

Το παραγοντικό του αριθμού 7 είναι : 5040
```

4.6.1.4.2 Ερωτήσεις για Επανάληψη και Εμπέδωση

1. Τι είναι συγκρότημα προτάσεων , ποιές προτάσεις σχηματίζουν συγκροτήματα;

2. Τι είναι βρόγχος;

3. Είναι αληθές ή ψευδές ότι, το σώμα (<σωμα>) της πρότασης **for** πρέπει να περιέχει το λιγότερο μια πρόταση;

4. Τι είναι ο κιβωτισμός βρόγχων ;

5. Πρέπει και γιάτι να αφήνετε μια ή περισσότερες από τις εκφρασεις (αρχική, ελέγχου και μετρητή) εκτός της παρένθεσης της πρότασης **for**;

6. Ποιός βρόγχος "κινείται" πιό γρήγορα ο εξωτερικός ή ο κιβωτισμένος του;

7. Ποιά είναι η έξοδος του παρακάτω προγράμματος;

```
for ( metritis = 10; metritis >= 1; metritis - = 3 )
   { cout << metritis << " \n"; }
```

8. Είναι αληθές ή ψευδές ότι, ο βρόγχος **for** είναι καλλίτερα να χρησιμοπιείται αντί του βρόγχου **while** όταν είναι γνωστό εκ των προτέρων το πλήθος N των ανακυκλώσεων που απαιτεί ο βρόγχος;

9. Τι συμβαίνει όταν η έκφραση ελέγχου (<εκφραση_ελεγχου>) είναι ψευδής σε έναν βρόγχο **for**;

10. Είναι αληθές ή ψευδές ότι, το ακόλουθο πρόγραμμα περιέχει έναν σωστό κιβωτισμένο βρόγχο;

```
for ( i = 1; i <= 10; i++ )
   {
     for ( j = 1; j <= 5; j++ )
        { cout << i << j; }
   }
```

11. Ποιά είναι η έξοδος του παρακάτω τμήματος κώδικα;

```
i = 1;
arxi = 1;
telos = 5;
vima =1;
for ( ; arxi >= telos; )
  { cout << i << "\n";
    arxi += vima;
    telos--; }
```

4.6.1.5 Άλλες Επιλογές Βρόγχων

Ο χειρισμός μεγάλων ποσοτήτων δεδομένων με δυναμικότερα προγράμματα απαιτεί την περαιτέρω αξιοποίηση των βρόγχων. Η αξιοποίησή τους συνίσταται κυρίως στη δυνατότητα ελέγχου ή ελέγχων τους, η οποία επιτυγχάνεται με την χρησιμοποίηση κάποιων τεχνικών ή και προτάσεων που σχετίζονται άμεσα με τον τρόπο που λειτουργούν.

Έτσι ως προς τις τεχνικές έχουμε τους **βρόγχους αναμονής**, οι οποίοι επιτρέπουν ελεγχόμενες καθυστερήσεις στην εκτέλεση προγραμμάτων. Οι βρόγχοι αναμονής είναι χρήσιμοι όταν θέλουμε να εμφανίσουμε μηνύματα για ορισμένες χρονικές περιόδους ή να παρακολουθήσουμε την πορεία εκτέλεσης προγραμμάτων ή να γράψουμε παιχνίδια για Η/Υ, οποιασδήποτε φύσεως, με μικρότερες ταχύτητες κ.λπ.

Ως προς τις προτάσεις έχουμε την πρόταση **break** και την πρόταση **continue**, οι οποίες συνδυαζόμενες με τους βρόγχους των προτάσεων **while** και **for** τους ελέγχουν.

4.6.1.5.1 Βρόγχοι Αναμονής

Αναμφίβολα οι σημερινοί Η/Υ είναι γρήγοροι, πολλές φορές τους θέλουμε ακόμα ταχύτερους. Όμως υπάρχουν και οι περιπτώσεις που θέλουμε να μειώσουμε την ταχύτητά τους, π.χ. για να προλαβαίνουμε να διαβάζουμε κάποια μηνύματα στην οθόνη, τότε χρησιμοποιούμε βρόγχο αναμονής.

Οι βρόγχοι αναμονής είναι συνήθως κβαντισμένοι βρόγχοι, οι οποίοι απλά εκτελούν την ανακύκλωση ενός βρόγχου **for** ή **while** πολλές φορές. Όσο μεγαλύτερη είναι η τιμή του βρόγχου **for**, τόσο μεγαλύτερος είναι ο χρόνος στον οποίο επαναλαμβάνεται ο βρόγχος. Οι βρόγχοι αναμονής είναι εύκολοι στην σχεδίασή τους, απλά

θέτουμε έναν κατάλληλο κενό βρόγχο **for** στην κατάλληλη θέση μέσα στο πρόγραμμα.

Ένας κιβωτισμένος βρόγχος αναμονής είναι επίσης κατάλληλος για μηνύματα λάθους. Εάν για παράδειγμα, ο χρήστης ζητήσει μια αναφορά, αλλά δεν εισάγει αρκετά δεδομένα στο πρόγραμμα ώστε αυτή να τυπωθεί, τότε μπορεί να εμφανιστεί στην οθόνη ένα προειδοποιητικό μήνυμα για μερικά δευτερόλεπτα (βρόγχος χρονομέτρησης). Το μήνυμα αυτό θα πληροφορεί ότι δεν μπορεί ακόμα ο χρήστης να ζητήσει την αναφορά, μετά αφού καθαριστεί η οθόνη από το μήνυμα, μπορεί να δοθεί άλλη μια ευκαιρία στον χρήστη.

Η αλήθεια είναι ότι δεν υπάρχει τρόπος για τον καθορισμό των επαναλήψεων ενός βρόγχου αναμονής σε ένα δευτερόλεπτο ή ένα λεπτό ή μια ώρα αναμονής, επειδή οι Η/Υ έχουν διαφορετικές ταχύτητες εκτέλεσης. Έτσι θα πρέπει να προσαρμοστεί η τελική τιμή των βρόγχων αναμονής ώστε να θέτει ανάλογα την καθυστέρηση.

Παραδείγματα:

1. Το πρόγραμμα που ακολουθεί είναι μια επανέκδοση του προγράμματος της αντίστροφης μέτρησης, που είδαμε στο π.χ.2 της §4.6.1.4, με εισαγωγή καθυστέρησης δηλαδή βρόγχου αναμονής. Έτσι κάθε αριθμός της αντίστροφης μέτρησης καθυστερείται ώστε να φαίνεται ότι η μέτρηση δεν γίνεται αμέσως. Η καθυστέρηση επιτυγχάνεται με μια κιβωτισμένη κενή **for**:

```
for ( kathisterisi = 1; kathisterisi <= 30000; kathisterisi++ );
```

Εάν το πρόγραμμα εκτελείται πολύ αργά ή πολύ γρήγορα στον Η/Υ σας τότε προσαρμόστε την τιμή καθυστέρησης (kathisterisi).

```
. . .
// Αντίστροφη μέτρηση με βρόγχο for και καθυστέρηση
#include <iostream.h>
main( )
{
 int metritis, kathisterisi;
 for ( metritis = 10; metritis != 0; metritis-- )
   {
     for ( kathisterisi = 1; kathisterisi <= 30000; kathisterisi++ ); // Καθυστέρηση
     cout << metritis << "\n";          // Εκτυπώνει την metritis καθώς μειώνεται
```

```
    }
 cout << " ΤΥΡΙΙΙ \n";
 return(0);
}
```

2. Το πρόγραμμα που ακολουθεί ζητά από τους χρήστες τις ηλικίες τους με το μήνυμα: "Δώστε την ηλικία σας : ".

Εάν ο χρήστης εισάγει μια ηλικία μικρότερη ή ίση του 0, το πρόγραμμα προειδοποιεί και μετά εμφανίζει ένα μήνυμα (... Η ηλικία σας δεν μπορεί να είναι τόσο μικρή ...) λάθους για μερικά δευτερόλεπτα με τη χρήση ενός κεβωτισμένου βρόγχου αναμονής - χρονομέτρησης.

Στη συνέχεια ζητά και πάλι από τους χρήστες τις ηλικίες τους με το μήνυμα: "Δώστε εκ νέου την ηλικία σας : ", δίνοντάς τους την ευχέρεια να κάνουν συνολικά 2 φορές λάθος.

Εάν για τρίτη φορά η ηλικία δοθεί μικρότερη ή ίση του 0, τότε το πρόγραμμα διακόπτεται εμφανίζοντας στην οθόνη το μήνυμα: " Ευχαριστούμε, δεν νομίζουμε ότι θέλετε πράγματι να μας πείτε την ηλικία σας" .

Εάν η ηλικία είναι μεγαλύτερη του 0 και μικρότερη του 18, τυπώνεται το μήνυμα: "Λυπούμαστε, δεν έχετε ακόμα ενηλικιωθεί και δεν μπορείτε να πάρετε μέρος στις εκλογές".

Εάν η ηλικία είναι μεγαλύτερη ή ίση του 18, τυπώνεται το μήνυμα " Έχετε ενηλικιωθεί και μπορείτε να πάρετε μέρος στις εκλογές".

Ανάλογα με την ταχύτητα του Η/Υ σας, προσαρμόστε το πλήθος των βρόγχων για να κανονίσετε τον χρόνο παραμονής του μηνύματος στην οθόνη.

```
. . .
// Εμφάνιση μηνύματος λάθους για λίγα δευτερόλεπτα
#include <iostream.h>
main( )
{
 int i = 1;
 int metritis, kathisterisi, ilikia;
 cout << "Δώστε την ηλικία σας : ";
 cin >> ilikia;
```

```
while ( ( ilikia <= 0 ) && ( i <= 2 ) )
  {
    cout << "... Η ηλικία σας δεν μπορεί να είναι τόσο μικρή ...";
    // Βρόγχος χρονομέτρησης
    for ( metritis = 1; metritis <= 30000; metritis++ )
      { for ( kathisterisi = 1; kathisterisi <= 500; kathisterisi++ ); }
    // Διαγραφή μηνύματος
    cout << "\n\n\n";
    cout << "Δώστε εκ νέου την ηλικία σας : ";
    cin >> ilikia;
    i = i + 1;
  }
if ( ilikia <= 0 )
  {
    cout << "\n\n Ευχαριστούμε, δεν νομίζουμε ότι θέλετε πράγματι";
    cout << "να μας πείτε την ηλικία σας";
  }
 else
  { if ( ilikia < 18 )
      { cout << "\n\n Λυπούμαστε, δεν έχετε ακόμα ενηλικιωθεί";
        cout << "και δεν μπορείτε να πάρετε μέρος στις εκλογές"; }
     else
      { cout << "\n\n Έχετε ενηλικιωθεί";
        cout << "και μπορείτε να πάρετε μέρος στις εκλογές"; }
  }
 return(0);
}
```

4.6.1.5.2 Οι Προτάσεις break και for

Ο βρόγχος **for** έχει σχεδιαστεί για να εκτελεί γνωστού πλήθους ανακυκλώσεις. Πολύ σπάνια μπορεί κάποιος να θελήσει να σταματήσει τον βρόγχο **for** πριν η μεταβλητή μέτρησής (<μεταβλητη_μετρησης>) του φθάσει στη τελική της τιμή. Στην περίπτωση αυτή, όπως και με τους βρόγχους **while**, χρησιμοποιείται η πρόταση **break**.

Η πρόταση **break** γράφεται σαν απόδοση μιας πρότασης **if** που είναι κιβωτισμένη

στο **for**. Δηλαδή η **if** που περιέχει την **break** είναι πρόταση του σώματος (<σωμα>) της **for**. Η πρόταση **break** που είναι απόδοση μιας πρότασης **if** λέγεται **πρόταση break υπό συνθήκη**.

Εάν η **break** ήταν πρόταση του σώματος (<σωμα>) της **for** χωρίς την **if**, τότε η πρόταση **break** θα κατέστρεφε κυριολεκτικά τον βρόγχο **for** (βλπ π.χ. 1.α - πρβλ π.χ. 1.β).

Εάν ένας βρόγχος κιβωτισθεί σε έναν άλλο βρόγχο, στο σώμα (<σωμα>) των οποίων έχει πληκτρολογηθεί η πρόταση **break**, τότε διακόπτεται ο "πιό ενεργός βρόγχος" για τη **break**, δηλαδή ο βρόγχος στου οποίου το σώμα (<σωμα>) ανήκει άμεσα. Αναλυτικότερα:

- Εάν η **break** ανήκει άμεσα στο σώμα (<σωμα>) του κιβωτισμένου βρόγχου, τότε διακόπτεται ο κιβωτισμένος βρόγχος.
- Εάν η **break** ανήκει άμεσα στο σώμα (<σωμα>) του εξωτερικού βρόγχου, τότε διακόπτεται ο εξωτερικός βρόγχος.

Η **break** υπό συνθήκη χρησιμοποιείται για περαιτώσεις ελέγχου του πλήθους δεδομένων, κατά τον χειρισμό ή αρχείων δεδομένων ή μεγάλου πλήθους δεδομένων που εισάγονται από το πληκτρολόγιο. Δηλαδή με την πρόταση αυτή δίνεται η δυνατότητα να διακόπτουμε την διαδικασία, μόλις αυτή πραγματοποιηθεί, για ένα υποσύνολο των δεδομένων που επιλέξαμε με κάποια κριτήρια, ως προς το πλήθος, από το αρχείο των δεδομένων ή από τα δεδομένα που εισάγονται από το πληκτρολόγιο. Σημειώνουμε ότι χωρίς την **break** υπό συνθήκη η διαδικασία θα συνεχιζόταν για όλο το πλήθος των δεδομένων (βλπ π.χ.1.β και π.χ.2).

Παραδείγματα:

1. Να γραφεί πρόγραμμα που μετρά 20 αριθμούς και χρησιμοποιεί μέσα στο σώμα της **for** (α) την πρόταση **break** και (β) την πρόταση **break** υπό συνθήκη, δηλαδή σαν απόδοση μιας **if** κιβωτισμένης στη **for**.

(α) . . .

```
// Ακύρωση βρόγχου for από πρόταση break
#include <iostream.h>
main( )
```

```
{
 int arithm;                                    // Μεταβλητή-μετρητής βρόγχου for
 cout << " Οι ακέραιοι αριθμοί από το 1 έως το 20 είναι : " << "\n"; // Τίτλος
 for ( arithm = 1; arithm <= 20; arithm++ )    // Τυπώνει κάθε δεύτερο αριθμό
  {
    cout << arithm;
    break;                                      // Άμεση έξοδος από τον βρόγχο
  }
 cout << "\n\n ... Τέλος του Προγράμματος ... ";
 return(0);
}
```

Το πρόγραμμα παράγει την ακόλουθη έξοδο:

```
Οι ακέραιοι αριθμοί από το 1 έως το 20 είναι :
1


... Τέλος του Προγράμματος ...
```

(β) . . .

```
// Βρόγχος for που λειτουργεί με αίτηση του χρήστη
// χρησιμοποιώντας την πρόταση break υπό συνθήκη if
#include <iostream.h>
main( )
{
 int arithm;                                    // Μεταβλητή-μετρητής βρόγχου for
 char apandisi;
 cout << " Οι ακέραιοι αριθμοί από το 1 έως το 20 είναι : " << "\n"; // Τίτλος
 for ( arithm = 1; arithm <= 20; arithm++ )    // Τυπώνει κάθε δεύτερο αριθμό
  {
    cout << arithm << "\n";
    cout << " Θέλετε να συνεχίσω ( ναι-Υ ή όχι-Ν); : \n";
    cin << apandisi;
    if ( apandisi == 'N' ) || ( apandisi == 'n' )
      { break; }                    // Έξοδος από τον βρόγχο αν το θέλει ο χρήστης
  }
 cout << "\n\n ... Τέλος του Προγράμματος ... ";
 return(0);
```

```
}
```

Το πρόγραμμα παράγει την ακόλουθη έξοδο:

```
Οι ακέραιοι αριθμοί από το 1 έως το 20 είναι :
1
Θέλετε να συνεχίσω ( ναι-Y ή όχι-N); : Y
2
Θέλετε να συνεχίσω ( ναι-Y ή όχι-N); : Y
3
Θέλετε να συνεχίσω ( ναι-Y ή όχι-N); : Y
4
Θέλετε να συνεχίσω ( ναι-Y ή όχι-N); : Y
5
Θέλετε να συνεχίσω ( ναι-Y ή όχι-N); : Y
6
Θέλετε να συνεχίσω ( ναι-Y ή όχι-N); : Y
7
Θέλετε να συνεχίσω ( ναι-Y ή όχι-N); : Y
8
Θέλετε να συνεχίσω ( ναι-Y ή όχι-N); : Y
9
Θέλετε να συνεχίσω ( ναι-Y ή όχι-N); : Y
10
Θέλετε να συνεχίσω ( ναι-Y ή όχι-N); : Y
11
Θέλετε να συνεχίσω ( ναι-Y ή όχι-N); : Y
12
Θέλετε να συνεχίσω ( ναι-Y ή όχι-N); : Y
13
Θέλετε να συνεχίσω ( ναι-Y ή όχι-N); : N


... Τέλος του Προγράμματος ...
```

2. Το πρόγραμμα που ακολουθεί είναι μια βελτιωμένη έκδοση του προγράμματος - παραδείγματος 5 της §4.6.1.4. Η βελτίωση έγκειται στην χρήση της πρότασης **break** υπό συνθήκη, η οποία δίνει την δυνατότητα διακοπής της διαδικασίας πριν από την κανονική του κατάληξη. Το πρόγραμμα υπολογίζει τον μέσο όρο των εξαμηνιαίων βαθμών των μαθητών κάποιας τάξης. Δηλαδή αναλυτικότερα:

1. Διαβάζει το πλήθος των μαθητών και τον εξαμηνιαίο βαθμό για κάθε μαθητή. Επιπλέον δίνει τη δυνατότητα στον δάσκαλο - χρήστη του προγράμματος, να εισάγει αντί

για έναν εξαμηνιαίο βαθμό έναν τερματιστή (ΤΕΡΜΑΤ = - 99 ή vathmos = - 99), με τον οποίο θα διακόπτει τη διαδικασία εισαγωγής άλλων στοιχείων, δηλαδή εξαμηνιαίων βαθμών.

2. Υπολογίζει τον μέσο όρο των δοθέντων βαθμών,

3. Τυπώνει τον μέσο όρο των δοθέντων βαθμών.

Σημείωση: Το πρόγραμμα σχεδιάζεται ώστε να υπολογίζει και να γνωρίζει, κάθε στιγμή, ο δάσκαλος - χρήστης το πλήθος των εισαγομένων εξαμηνιαίων βαθμών, δηλαδή το πλήθος των μαθητών. Διότι, η διαδικασία, με τη χρήση της **break** υπό συνθήκη, μπορεί να διακοπεί πριν από την κανονική του προγράμματος κατάληξη και το τελικό πλήθος των μαθητών των οποίων η εξαμηνιαία βαθμολογία έχει εισαχθεί - που επίσης, θα πάρει μέρος στην διαδικασία υπολογισμού του μέσου όρου - να είναι διαφορετικό από το πλήθος των μαθητών που δόθηκε στην αρχή του προγράμματος.

```
. . .
// Υπολογίζει μέσο όρο με βρόγχο for και
// επιτρέπει διακοπή της διαδικασίας με break υπό συνθήκη
#include <iostream.h>
#include <iomanip.h>
main( )
{
 float vathmos, mesoros;
 float athrisma = 0.0;
 int plithmathit ;              // Αρχικό πλήθος μαθητών-βαθμών
 int metritmathit = 0;          // Μετρητής μαθητών-βαθμών
 int metritis ;                 // Μετρητής για τον έλεγχο των βρόγχων
 cout << " \n            ***Υπολογισμός Βαθμολογίας ***            \n\n"; // Τίτλος
 cout << " Δώστε το πλήθος των μαθητών : ";
 cin >> plithmathit ;           // Αρχικό πλήθος μαθητών-βαθμών
 cout << " \n";
 for ( metritis = 1; metritis <= plithmathit; metritis++ )
  {
   cout << " \n Δώστε τον εξαμηνιαίο βαθμό του επόμενου μαθητή " ;
   cout << " \n ( προσοχή! η διαδικασία τερματίζει όταν vathmos = - 99 ) : ";
   cin >> vathmos;
   cout << " \n";
   if ( vathmos < 0.0 )         // Συνθήκη τερματισμού vathmos = - 99
```

```
        { break; }                    // Έξοδος υπό συνθήκη
        metritmathit++ ;              // Μετρητής εισαγομένων μαθητών-βαθμών
        athrisma += vathmos;
       }
    mesoros = athrisma / metritmathit;
    cout << "\n Ο μέσος όρος των εξαμηνιαίων βαθμών της τάξης είναι : " <<
          setprecision(1) << mesoros;
    return(0);
}
```

Το πρόγραμμα παράγει την ακόλουθη έξοδο:

```
                    *** Υπολογισμός Βαθμολογίας ***

        Δώστε το πλήθος των μαθητών : 15

        Δώστε τον εξαμηνιαίο βαθμό του επόμενου μαθητή
        ( προσοχή! η διαδικασία τερματίζει όταν vathmos = - 99 ) : 87

        Δώστε τον εξαμηνιαίο βαθμό του επόμενου μαθητή
        ( προσοχή! η διαδικασία τερματίζει όταν vathmos = - 99 ) : 67

        Δώστε τον εξαμηνιαίο βαθμό του επόμενου μαθητή
        ( προσοχή! η διαδικασία τερματίζει όταν vathmos = - 99 ) : 94

        Δώστε τον εξαμηνιαίο βαθμό του επόμενου μαθητή
        ( προσοχή! η διαδικασία τερματίζει όταν vathmos = - 99 ) : 89

        Δώστε τον εξαμηνιαίο βαθμό του επόμενου μαθητή
        ( προσοχή! η διαδικασία τερματίζει όταν vathmos = - 99 ) : 97

        Δώστε τον εξαμηνιαίο βαθμό του επόμενου μαθητή
        ( προσοχή! η διαδικασία τερματίζει όταν vathmos = - 99 ) : -99


        Ο μέσος όρος των εξαμηνιαίων βαθμών της τάξης είναι : 86.8
```

4.6.1.5.3 Η Πρόταση continue

Οι προτάσεις **break** και **continue** είναι προτάσεις που εκτρέπουν τις διαδικασίες, στις

οποίες εισάγονται, από την κανονική τους ροή. Όπως είδαμε, εάν η **break** είναι πρόταση του σώματος (<σωμα>) μιας **for** ή μιας **while** τότε εξαναγκάζει την διακοπή του βρόγχου ανακύκλωσής τους. Αντίθετα, εαν η **continue** είναι πρόταση του σώματος (<σωμα>) μιας **for** ή μιας **while** τότε εξαναγκάζει τον Η/Υ να εκτελέσει άλλη μια επανάληψη του βρόγχου, δηλαδή η **continue** οδηγεί στο να αγνοηθεί κάθε πρόταση του βρόγχου που την ακολουθεί.

Παραθέτουμε στον πίνακα 4.6.1.5.3 "σχηματικά" τις διαφορές των προτάσεων **break** και **continue** διά μέσω των χαρακτηριστικών τους.

```
for ( i = 0; i < = 10; i++ )
  {
    break;        ---------- Η break τερματίζει τον βρόγχο άμεσα ----------
    cout << i << "η ανακύκλωση \n"; // Δεν εκτυπώνεται ποτέ !!
  }
//     Υπόλοιπο πρόγραμμα  <----------

for ( i = 0; i < = 10; i++ )  <----------
  {
    continue;     ---------- Η continue κάνει τον βρόγχο να εκτελέσει μια ακόμα ανακύκλωση ----------
    cout << i << "η ανακύκλωση \n"; // Δεν εκτυπώνεται ποτέ !!
  }
//     Υπόλοιπο πρόγραμμα
```

ΠΙΝΑΚΑΣ 4.6.1.5.3 : Διαφορές μεταξύ των προτάσεων break και continue

Η **continue** χρησιμοποιείται σε προγράμματα ως πρόταση βρόγχων που ελέγχουν τις εισερχόμενες τιμές των δεδομένων τους, αν δηλαδή, πράγματι οι τιμές αυτές βρίσκονται εντός των ορίων απαίτησης του προβλήματος. Σε περιπτώσεις που οι τιμές των δεδομένων είναι εκτός των ορίων, με την **continue** ο βρόγχος επιστρέφει στην αρχή, δίνοντας μήνυμα και νέα ευκαιρία στον χρήστη για να εισάγει αποδεκτές τιμές.

Τα περισσότερα προγράμματα χρησιμοποιούν την **continue** σαν απόδοση

(<αποδοση>) μιας πρότασης if, δηλαδή μια **πρόταση continue υπό συνθήκη**, η οποία είναι σαφώς περισσότερο χρήσιμη από μια απλή **continue** (πρβλ π.χ.1με π.χ.2 και π.χ.3).

Όπως προαναφέραμε η **continue** γενικά, οδηγεί στο να αγνοηθεί κάθε πρόταση του βρόγχου που την ακολουθεί (βλπ Πίνακα 4.6.5.3.1). Εάν αυτό συμβεί σε προγράμματα, παρόλο που πρέπει να αποφεύγεται (βλπ π.χ.1), τότε κάποιοι μεταγλωττιστές δίνουν προειδοποιητικό μήνυμα κατά τη μεταγλώττιση, αναγνωρίζοντας τους κωδικούς που δεν εκτελούνται ποτέ λόγω της **continue** .

Παραδείγματα:

1.Το πρόγραμμα που ακολουθεί φαίνεται να τυπώνει τους αριθμούς από 1 μέχρι 10, ακολουθούμενος ο καθένας από την πρόταση: "Προτιμάτε να προγραμματίζετε με C++", όμως, τελικά δεν γίνεται αυτό. Η ύπαρξη της **continue** στο σώμα (<σωμα>) του **for** προκαλεί πρόωρο τέλος του βρόγχου, λόγω δε της θέσης της η πρόταση: "Προτιμάτε να προγραμματίζετε με C++" δεν τυπώνεται ποτέ.

```
. . .
// Εισάγει την continue στον βρόγχο for
#include <iostream.h>
main( )
{
 int metritis;
 for ( metritis = 1; metritis <= 10; metritis++ )  // Βρόγχος 10 ανακυκλώσεων
  {
    cout << metritis << " ";
    continue;        // Κάνει τον βρόγχο να εκτελέσει μια ακόμα επανάληψη και
                     // να αγνοηθεί η πρόταση του βρόγχου που την ακολουθεί
    cout << " Προτιμάτε να προγραμματίζετε με C++ \n";
  }
 return(0);
}
```

Το πρόγραμμα τελικά παράγει ως έξοδο τα εξής:

1 2 3 4 5 6 7 8 9 10

2. Το πρόγραμμα που ακολουθεί διαβάζει πέντε (5) πεζά γράμματα του λατινικού

αλφαβήτου. Κάθε πεζό γράμμα διαβάζεται και μετατρέπεται στο αντίστοιχο κεφαλαίο. Το πρόγραμμα χρησιμοποιεί τον κώδικα ASCII για να ελέγξει αν πράγματι πληκτρολογούνται πεζά γράμματα του λατινικού αλφαβήτου. Σημειώνουμε ότι, τα πεζά γράμματα του λατινικού αλφαβήτου στον κώδικα ASCII βρίσκονται στην περιοχή από το 97 μέχρι το 122. Εάν οι χρήστες δεν πληκτρολογούν πεζό γράμμα το πρόγραμμα αγνοεί το λάθος λόγω της πρότασης **continue** .

```
. . .
// Εισάγει 5 πεζά λατινικά γράμματα και
// εκτυπώνει τα αντίστοιχα κεφαλαία
#include <iostream.h>
main( )
{
 char gramma;
 int metritis ;
 for ( metritis = 1; metritis <= 5; metritis++ )
  {
    cout << " Παρακαλώ δώστε ένα πεζό λατινικό γράμμα : ";
    cin >> gramma;
    if (( gramma < 97 ) || ( gramma > 122 ))   // Έλεγχος αν το γράμμα είναι
                                               // εκτός περιοχής
       { continue; }                           // Εισαγωγή άλλου γράμματος
    gramma- = 32;                              // Αφαιρέστε 32 από τιμή ASCII
    cout << " Το αντίστοιχο κεφαλαίο γράμμα είναι : " << gramma << "\n" ;
  }
 return(0);
}
```

Σημείωση: Λόγω της πρότασης **continue** μόνο τα πεζά γράμματα μετατρέπονται σε κεφαλαία.

3. Το πρόγραμμα που ακολουθεί διαβάζει τις μηνιαίες αποδοχές των υπαλλήλων μιας εταιρίας, υπολογίζει και τυπώνει τον μέσο όρο των μισθών των υπαλλήλων που οι ετήσιες αποδοχές τους είναι μεγαλύτερες ή ίσες από 6,000,000 δρχ. Το πρόγραμμα τερματίζει όταν για μηνιαία αποδοχή υπαλλήλου διαβάσει το -1. Το πρόγραμμα χρησιμοποιεί την **do_while** για τις ανακυκλώσεις που χρειάζεται για την εισαγωγή των δεδομένων και

υπολογισμό των αποτελεσμάτων. Επίσης χρησιμοποιεί αφενός την πρόταση **continue** υπό συνθήκη για να αποκλείει τους υπαλλήλους με ετήσιο μισθό μικρότερο της συνθήκης, αφετέρου την **break** υπό συνθήκη για να διακόπτει την διαδικασία εάν μισθός είναι -1.

```
. . .
// Μέσος όρος μισθών πάνω από 6,000,000 ετησίως
#include <iostream.h>
#include <iomanip.h>
main( )
{
 float misthos, isodima;
 float mesoros = 0.0, athrisma = 0.0;
 int metritis ;
 do
  {
    cout << " Δώστε τον επόμενο μηνιαίο μισθό (-1" << " για τερματισμό) : ";
    cin >> misthos;

    if (( isodima = misthos *12.00 ) < = 6,000,000)      // Μην προσθέτεις
      { continue; }                                      // χαμηλούς μισθούς

    if ( misthos < 0.0 )
      { break; }                                         // Έξοδος εάν εισαχθεί -1
    metritis++ ;              // Καταμέτρηση ετήσιων εισοδημάτων > 6,000,000
     athrisma+ = isodima;     // Πρόσθεση ετήσιων εισοδημάτων > 6,000,000
  }
 while ( misthos > 0.0 );
 mesoros = athrisma / (float)metritis ;                  // Υπολογισμός Μέσου Όρου
 cout << "\n\n Ο μέσος όρος των υψηλών μισθών ετησίως είναι : " <<
          setprecision( 2) << mesoros << " δρχ";
 return(0);
}
```

Παρατήρηση: Το πρόγραμμα χρησιμοποιεί αντίστοιχα την **continue** , την **break** και την **do_while** , για να συνεχίσει με άλλη επανάληψη εάν ο μισθός είναι χαμηλός, για να εγκαταλείψει τον βρόγχο εάν ο χρήστης πληκτρολογήσει το -1 και τέλος για να συνεχίσει την πρόσθεση των υψηλών μισθών με στόχο τον υπολογισμό του μέσου όρου.

Το πρόγραμμα παράγει την εξής έξοδο:

```
Δώστε τον επόμενο μηνιαίο μισθό (-1 για τερματισμό) : 600,000
Δώστε τον επόμενο μηνιαίο μισθό (-1 για τερματισμό) : 700,000
Δώστε τον επόμενο μηνιαίο μισθό (-1 για τερματισμό) : 500,000
Δώστε τον επόμενο μηνιαίο μισθό (-1 για τερματισμό) : 800,000
Δώστε τον επόμενο μηνιαίο μισθό (-1 για τερματισμό) : 900,000
Δώστε τον επόμενο μηνιαίο μισθό (-1 για τερματισμό) : 600,000
Δώστε τον επόμενο μηνιαίο μισθό (-1 για τερματισμό) : 800,000
Δώστε τον επόμενο μηνιαίο μισθό (-1 για τερματισμό) : -1

Ο μέσος όρος των υψηλών ετησίως μισθών είναι : 33902756.57 δρχ
```

4.6.1.5.4 Ερωτήσεις για Επανάληψη και Εμπέδωση

1. Για ποιόν σκοπό χρησιμοποιούνται οι βρόγχοι αναμονής;

2. Γιατί οι περιοχές (<αρχικη_εκφραση>, <εκφραση_ελεγχου> και <εκφραση_μετρητη>) των βρόγχων πρέπει να προσαρμόζονται για διαφορετικούς τύπους υπολογιστών;

3. Γιατί οι προτάσεις **break** και **continue** σπάνια εμφανίζονται χωρίς πρόταση **if** που τις ελέγχει;

4. Ποιά είναι η έξοδος του τμήματος κώδικα που ακολουθεί;

```
for ( i = 1; i <= 10; i++ )
  {
    continue;
    cout << " ********* \n";
  }
```

5. Ποιά είναι η έξοδος του τμήματος κώδικα που ακολουθεί;

```
for ( i = 1; i <= 10; i++ )
  {
    cout << " ********* \n";
    break;
  }
```

6. Για την υλοποίηση ενός βρόγχου αναμονής γιατί, γενικά, θα πρέπει να χρησιμοποιούνται

κιβωτισμένοι βρόγχοι;

4.6.1.6 Οι προτάσεις switch και go to

Η πρόταση **switch** συντονίζει τις πολλαπλές επιλογές του προγράμματος βελτιώνοντας τις προτάσεις **if** και **else_if**. Έτσι, με τη **switch** διευκολύνεται η γραφή επιλογών (menu) και προγραμμάτων πολλαπλών επιλογών εισαγωγής δεδομένων. Σχεδόν κάθε πρόταση **switch** περιλαμβάνει μια πρόταση **break** στο σώμα (<σωμα>) της. Τέλος, η ενότητα των εκτελέσιμων και μη προτάσεων ολοκληρώνεται με την πρόταση **goto**, η οποία εκτρέπει το πρόγραμμα από την κανονική του ροή, για αυτόν τον λόγο θα χρησιμοποιείται πολύ σπάνια και με μεγάλη προσοχή.

4.6.1.6.1 Η Πρόταση switch

Η πρόταση **switch** ονομάζεται και πρόταση πολλαπλής επιλογής, διότι επιτρέπει στο πρόγραμμα να επιλέξει από αρκετές εναλλακτικές εκδοχές ή διαδρομές ή λύσεις.

Η σύνταξη της **switch** ακολουθεί τους εξής κανόνες οι οποίοι διατυπώνονται με τη βοήθεια της BNF:

1. <πρόταση_switch> ::= **switch**(<προσδιοριστής>)
   { <σωμα_switch> } ,

2. <σωμα_switch> ::= <σωμα_επιλογων>
   default: {<συνθετη_προταση>} |
   <σωμα_επιλογων>

3. <σωμα_επιλογων> ::= **case**(<παραγοντας>):{ <συνθετη_προταση> } |
   case(<παραγοντας>):{ <συνθετη_προταση> }
   <σωμα_επιλογων> ,

4. <παραγοντας> ::= <ακεραιος> | <πραγματικος> | <χαρακτηρας> |
   <προσδιοριστής> | ... | <προσδιοριστής_συναρτησης> | ... ,

5. <προσδιοριστής> ::= <αλφαβητικος_χαρακτηρας> |
   <αλφαβητικος_χαρακτηρας><δεκαδικο_ψηφιο> |
   <αλφαβητικος_χαρακτηρας><προσδιοριστής>,

6. <αλφαβητικος_χαρακτηρας> ::= A | B | C | D | ... | Z | a | b | c | d | ... | z ,

7. <δεκαδικο_ψηφιο> ::= 0|1|2|...|9,

8. <συνθετη_προταση> ::= <προταση>;|
<προταση>;<συνθετη_προταση>,

9. <προταση> ::= <αναθεση>|
<αναγνωση _εισοδου> | <γραφη_εξοδου> |
<ανακυκλωση> |
<υποθετικη> |
<εκτροπη>[61] |
<πολλαπλη_επιλογη>[62] |
<κληση_υποπρογραμματος> (δηλαδή <κληση_συναρτησης>) .

Για την καλλίτερη κατανόηση της σύνταξης και της σημασιολογίας, καθώς και την επιτυχή αξιοποίηση της πρότασης πολλαπλής επιλογής **switch**, παραθέτουμε, πριν από τα παραδείγματα, ένα πρότυπο της πρότασης και χρήσιμες για την εφαρμογή της παρατηρήσεις.

Το πρότυπο της πρότασης είναι στη πραγματικότητα μια παραγωγή που προκύπτει από την εφαρμογή των παραπάνω κανόνων ορισμού ή παραγωγής (1έως 3) της πρότασης πολλαπλής επιλογής **switch**:

```
switch( <προσδιοριστης>)
    {
      case(<παραγοντας>):{ <συνθετη_προταση> }
      case(<παραγοντας>):{ <συνθετη_προταση> }
      case(<παραγοντας>):{ <συνθετη_προταση> }

      . . .

      default: {<συνθετη_προταση>}
    }
```

Το πλήθος των επιλογών (**case**) που ακολουθούν τη γραμμή της **switch** εξαρτάται από την εκάστοτε εφαρμογή.

Η γραμμή της **default** είναι προαιρετική. Σχεδόν όλες οι προτάσεις της **switch**

[61] <εκτροπη> ::= break | continue | goto

[62] <πολλαπλη επιλογη> ::= <προταση switch>

περιλαμβάνουν την **default**. Η γραμμή της **default** δεν είναι απαραίτητο να είναι η τελευταία πρόταση του σώματος της **switch** (<σωμα_switch>).

Εάν η τιμή του προσδιοριστή (<προσδιοριστης>) ταιριάζει με κάποιον από τους παράγοντες (<παραγοντας>) κάποιας από τις επιλογές (**case**), τότε εκτελείται η σύνθετη πρόταση (<συνθετη_προταση>) εκείνης της επιλογής (**case**).

Εάν η τιμή του προσδιοριστή (<προσδιοριστης>) δεν ταιριάζει με κανένα από τους παράγοντες (<παραγοντας>) των επιλογών (**case**), τότε εκτελείται η σύνθετη πρόταση (<συνθετη_προταση>) της επιλογής **default**.

Οι προσδιοριστές (<προσδιοριστης>) των επιλογών (**case**) δεν χρειάζονται παρενθέσεις, όμως, η ύπαρξη τους κάνει πιό εμφανή την τιμή των προσδιοριστών.

Σε κάθε σύνθετη πρόταση (<συνθετη_προταση>) κάθε επιλογής (**case**) απαραίτητη είναι η πρόταση **break**, διότι επιταχύνει τη διαδικασία, αφού μετά το τέλος της εκτέλεσης της σύνθετης πρότασης της επιλεγμένης επιλογής διακόπτει την **switch**.

Στην πράξη η πρόταση πολλαπλών επιλογών **switch** εφαρμόζεται πολύ εύκολα, παρά τις όποιες "κακές εντυπώσεις" που ίσως προκαλεί στους αρχάριους ο "σύνθετος" τύπος της.

Οπουδήποτε απαιτείται η χρησιμοποίηση συνδυασμών προτάσεων **if-else-if** είναι δυνατή και ίσως σε κάποιες περιπτώσεις απλούστερη η χρησιμοποίηση της **switch**.

Όμως, είναι δεδομένο ότι οι συνδυασμοί προτάσεων **if-else-if** δεν είναι δύσκολοι στην κατανόηση κσι στον χειρισμό. Θα μπορούσαμε, ανεπιφύλακτα, να υποστηρίξουμε ότι από την πρακτική εμπειρία προκύπτει το εξής συμπέρασμα: "Όταν η λογική έκφραση (<λογικη_εκφραση> ή <υποθεση>) που αποφασίζει την επιλογή είναι πολύπλοκη, δηλαδή περιέχει πολλούς τελεστές **&&** και **||**, η χρησιμοποίηση της πρότασης **if** είναι η καλλίτερη και προτιμάται".

Αντίθετα, προτιμάται η **switch** όταν ο προσδιοριστής (<προσδιοριστης>) της, που καθορίζει κάθε φορά την επιλογή, είναι περιγραφική σταθερά ή μεταβλητή ή λογική

έκφραση (<λογική_εκφραση>).

Αυξάνεται η ταχύτητα του προγράμματος, όταν οι προτάσεις **case** διατάσσονται από αυτήν που εκετελείται συχνότερα σε αυτή που εκετελείται σπανιότερα.

Τα παραδείγματα που ακολουθούν βοηθούν όχι μόνο στην κατανόηση της χρήσης της **switch** , αλλά και της διαφοράς τους με την **if**.

Παραδείγματα:

1. Να γραφεί πρόγραμμα που θα μαθαίνει σε κάποιο νήπιο να μετράει. Το πρόγραμμα ζητά από τον χρήστη-δάσκαλο έναν αριθμό από 1 έως 5. Μετά ηχεί (με τον ήχο του Η/Υ) τόσες φορές όσες χρειάζονται για να αντιστοιχήσει το νήπιο το πλήθος των ήχων με τον αριθμό. Έτσι στο πρόγραμμα:

(α) χρησιμοποιούνται οι συνδυασμοί **if-else-if** για τον ορισμό της μεθόδου διδασκαλίας μέτρησης με ήχο.

(β) περιλαμβάνεται το απαραίτητο μετωπικό αρχείο στο οποίο αποθηκεύονται οι επαναλαμβανόμενοι έξοδοι ένος ήχου **cout** που ορίζεται.

(γ) ορίζεται μια (καθολική) μεταβλητή η BEEP, η οποία εξασφαλίζει τον ήχο και μετακινεί τον δρομέα στην επόμενη γραμμή.

(δ) ορίζεται μια μεταβλητή τύπου ακεραίου η num, η οποία θα περιέχει την απάντηση του χρήστη.

(ε) ζητείται ένας αριθμός από τον χρήστη που ανατίθεται στην num και **εάν** η num είναι 1 **τότε** η BEEP καλείται μια φορά, **εάν** η num είναι 2 **τότε** η BEEP καλείται δύο φορές, **εάν** η num είναι 3 **τότε** η BEEP καλείται τρείς φορές, **εάν** η num είναι 4 **τότε** η BEEP καλείται τέσσερεις φορές, **εάν** η num είναι 5 **τότε** η BEEP καλείται πέντε φορές.

```
. . .
// Ηχεί ορισμένες φορές
#include <iostream.h>

// Όρισε έναν ήχο cout για αποθήκευση των
// επαναλαμβανομένων εξόδων μέσα στο πρόγραμμα
#define BEEP cout < "\a\n";

main( )
{
```

```
int num;
// Ζητήστε έναν αριθμό από τον χρήστη - δάσκαλο
cout << " Δώστε τον αριθμό που θέλετε να μάθει το νήπιο :";
cin >> num;
// Χρήση πολλαπλών προτάσεων if για ήχο
if ( num == 1 )
  { BEEP; }
else if ( num == 2 )
        { BEEP; BEEP; }
      else if ( num == 3 )
             { BEEP; BEEP; BEEP; }
           else if ( num == 4 )
                  { BEEP; BEEP; BEEP; BEEP; }
                else if ( num == 5 )
                       { BEEP; BEEP; BEEP; BEEP; BEEP; }
return(0);
}
```

Παρατηρήσεις:

1. Δεν ηχεί κανένες ήχος μέχρι να εισαχθεί ένας αριθμός μεταξύ 1 και 5.
2. Στο πρόγραμμα χρησιμοποιείται το πλεονέκτημα της οδηγίας του προεπεξεργαστή:

```
#define BEEP cout < "\ a \n";
```

για να οριστεί μια σύντμηση του cout. Η BEEP δεν είναι "εντολή" αλλά αντικαθιστά τον cout όποτε εμφανίζεται.

3. Το μειονέκτημα του προγράμματος είναι η χρήση των if , διότι αφενός υπάρχει περιορισμός με το πλήθος των κιβωτισμένων if, δηλαδή δεν υπάρχει χώρος για περισσότερα από έξι επίπεδα κιβωτισμών, αφετέρου είναι δύσκολη η παρακολούθηση αυτού του τύπου λογικής. Επειδή ακριβώς εμπλέκεται μια πολλαπλή επιλογή, η πρόταση **switch** είναι η καταλληλότερη, αυτό φαίνεται στο παρακάτω παράδειγμα που επαναλαμβάνεται το ίδιο πρόγραμμα αλλά με την χρήση της **switch** .

2. Να γραφεί πρόγραμμα που θα μαθαίνει σε κάποιο νήπιο να μετράει. Το πρόγραμμα ζητά από τον χρήστη-δάσκαλο έναν αριθμό από 1 έως 5. Μετά ηχεί (με τον ήχο του Η/Υ) τόσες

φορές όσες χρειάζονται για να αντιστοιχήσει το νήπιο το πλήθος των ήχων με τον αριθμό. Στο πρόγραμμα χρησιμοποιείται αποκλειστικά η πρόταση πολλαπλής επιλογής **switch** .

```
. . .
// Ηχεί ορισμένες φορές
#include <iostream.h>

// Όρισε έναν ήχο cout για αποθήκευση των
// επαναλαμβανομένων εξόδων μέσα στο πρόγραμμα
#define BEEP cout < "\ a \n";

main( )
{
 int num;

 // Ζητήστε έναν αριθμό από τον χρήστη - δάσκαλο
 cout << " Δώστε τον αριθμό που θέλετε να μάθει το νήπιο : ";
 cin >> num;

 // Χρήση της πρότασης πολλαπλής επιλογής switch
 switch (num)
 { case (1): { BEEP;
               break; }

  case (2): { BEEP; BEEP;
               break; }

  case (3): { BEEP; BEEP; BEEP;
               break; }
  case (4): { BEEP; BEEP; BEEP; BEEP;
               break; }
  case (5): { BEEP; BEEP; BEEP; BEEP; BEEP;
               break; }
 }
 return(0);
}
```

Παρατηρήσεις:

1. Αναμφισβήτητα το πρόγραμμα του παραδείγματος 2 είναι πολύ πιό κατανοητό από αυτό του παραδείγματος 1.

2. Η τιμή της num ελέγχει την εκτέλεση της επιλογής, εκτελείται η επιλογή **case** που ο παράγοντάς της ταιριάζει με την τιμή της num. Τα επίπεδα κιβωτισμού διαχωρίζουν τις επιλογές.
3. Εάν περισσότερες από μια **case** έχουν τον ίδιο παράγοντα τότε εκτελείται μόνο η πρώτη.
4. Εάν ο χρήστης-δάσκαλος εισάγει έναν αριθμό διαφορετικό από το 1 μέχρι το 5, δεν θα υπάρξει καμμία ανταπόκριση, κανένας ήχος, αφού δεν υπάρχει καμμία έκφραση **case** που να ταιριάζει με άλλη τιμή, ακόμα δεν υπάρχει και η **default**. Στο παρακάτω παράδειγμα επαναλαμβάνεται, και πάλι, το ίδιο πρόγραμμα αλλά με την χρήση της **default**.

3. Να γραφεί πρόγραμμα που θα μαθαίνει σε κάποιο νήπιο να μετράει. Το πρόγραμμα ζητά από τον χρήστη-δάσκαλο έναν αριθμό από 1 έως 5. Μετά ηχεί (με τον ήχο του Η/Υ) τόσες φορές όσες χρειάζονται για να αντιστοιχήσει το νήπιο το πλήθος των ήχων με τον αριθμό. Στο πρόγραμμα χρησιμοποιείται αποκλειστικά η πρόταση πολλαπλής επιλογής **switch** με την επιλογή **default**.

```
. . .
// Ηχεί ορισμένες φορές
#include <iostream.h>

// Όρισε έναν ήχο cout για αποθήκευση των
// επαναλαμβανομένων εξόδων μέσα στο πρόγραμμα
#define BEEP cout << "\a\n";

main( )
{
 int num;

 // Ζητήστε έναν αριθμό από τον χρήστη - δάσκαλο
 cout << " Δώστε τον αριθμό που θέλετε να μάθει το νήπιο :";
 cin >> num;

 // Χρήση της πρότασης πολλαπλής επιλογής switch
 switch (num)
 { case (1): { BEEP;
               break; }
   case (2): { BEEP; BEEP;
```

```
                break; }
    case (3): { BEEP; BEEP; BEEP;
                break; }
    case (4): { BEEP; BEEP; BEEP; BEEP;
                break; }
    case (5): { BEEP; BEEP; BEEP; BEEP; BEEP;
                break; }
    default: { cout << "Πρέπει να δώσετε έναν αριθμό μεταξύ 1 και 5 \n";
               cout << "Παρακαλώ ξανατρέξτε το πρόγραμμα \n";
               break; }
  }
  return(0);
}
```

Παρατηρήσεις:

1. Η **break** στο τέλος της επιλογής **default**, που γράφεται χάριν ομοιομορφίας, φαίνεται περιττή, αφού ούτως ή άλλως καμμία άλλη επιλογή **case** δεν εκτελείται μετά την **default**.
2. Δεν είναι απαραίτητο η επιλογή **default** να είναι η τελευταία στο σώμα της **switch**, όμως μετακινούμενη η **default** μετακινείται και η **break**.
3. Η πρόταση **break** συνδυαζόμενη με την την πρόταση **switch** είναι πολύ χρήσιμη αφού συμπληρώνει την λειτουργικότητα και την αποτελεσματικότητα της δεύτερης. Του λόγου το αληθές φαίνεται στο παράδειγμα που έπεται.

4. Να γραφεί πρόγραμμα που θα μαθαίνει σε κάποιο νήπιο να μετράει. Το πρόγραμμα ζητά από τον χρήστη-δάσκαλο έναν αριθμό από 1 έως 5. Μετά ηχεί (με τον ήχο του Η/Υ) τόσες φορές όσες χρειάζονται για να αντιστοιχήσει το νήπιο το πλήθος των ήχων με τον αριθμό. Στο πρόγραμμα χρησιμοποιείται αποκλειστικά η πρόταση πολλαπλής επιλογής **switch** με την επιλογή **default**, χωρίς να χρησιμοποιειθεί σε καμμία επιλογή η πρόταση **break**.

```
. . .
// Ηχεί λανθασμένα διότι δεν χρησιμοποιείται ένας διακόπτης
#include <iostream.h>

// Όρισε έναν ήχο cout για αποθήκευση των
// επαναλαμβανομένων εξόδων μέσα στο πρόγραμμα
#define BEEP cout < "\ a\n";
```

```
main( )
{
 int num;
 // Ζητήστε έναν αριθμό από τον χρήστη - δάσκαλο
 cout << " Δώστε τον αριθμό που θέλετε να μάθει το νήπιο :";
 cin >> num;
 // Χρήση της πρότασης πολλαπλής επιλογής switch
 switch (num)
 {                                  // Προειδοποίηση !!!!
   case (1): { BEEP; }              // Χωρίς την break, ο ήχος του κώδικα
   case (2): { BEEP; BEEP; }        // πέφτει μέσα στους υπόλοιπους ήχους
   case (3): { BEEP; BEEP; BEEP; }
   case (4): { BEEP; BEEP; BEEP; BEEP; }
   case (5): { BEEP; BEEP; BEEP; BEEP; BEEP; }
   default: {  cout << "Πρέπει να δώσετε έναν αριθμό μεταξύ 1 και 5 \n";
               cout << "Παρακαλώ ξανατρέξτε το πρόγραμμα \n";  }
 }
 return(0);
}
```

Παρατηρήσεις:

1. Όταν ο χρήστης εισάγει το 1, το πρόγραμμα ηχεί 15 φορές. Η **break** δεν υπάρχει ενσωματωμένη στη σύνθετη πρόταση (<συνθετη_προταση>) της κάθε επιλογής **case**, για να να διακόπτει τη συνέχιση της εκτέλεσης των επιλογών **case** που έπονται της επιλεγμένης **case**.

2. Αντίθετα με άλλες γλώσσες προγραμματισμού, όπως για παράδειγμα η Pascal, η πρόταση **switch** της C++ απαιτεί και την ύπαρξη της **break** σε κάθε επιλογή **case**, για να εκτελεί μια μόνο **case** κάθε φορά. Αυτό δεν είναι απαραίτητα μειονέκτημα, αφού με τον τρόπο αυτόν η **break** προσφέρει τη δυνατότητα για περισσότερο έλεγχο στη διαχείριση συγκεκριμένων **case**, όπως φαίνεται στο επόμενο παράδειγμα.

5. Να γραφεί πρόγραμμα το οποίο ελέγχει την εκτύπωση των πωλήσεων στο τέλος της ημέρας, της εβδομάδας, του μήνα. Το πρόγραμμα για να ελέγξει την εκτύπωση των

πωλήσεων στο τέλος μιας ημέρας, ζητάει πρώτα την ημέρα της εβδομάδας.

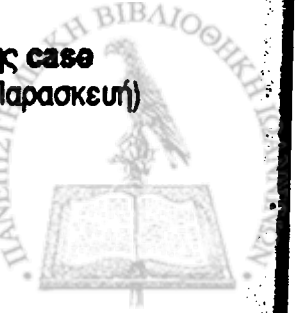
(α) Εάν η ημέρα είναι από Δευτέρα μέχρι Πέμπτη, τότε τυπώνεται ένα ημερήσιο σύνολο. Σημειώνουμε ότι, για κάθε τύπο αναφοράς πωλήσεων γίνεται, όπως είναι λογικό, ανάλογος χειρισμός ιεραρχώντας τις επιλογές **case**. Η ημερήσια ποσότητα είναι η τελευταία (τρίτη) **case**, έτσι, είναι και η μόνη που τυπώνεται εάν η ημέρα είναι από Δευτέρα μέχρι Πέμπτη.

(β) Εάν η ημέρα είναι Παρασκευή, τότε τυπώνονται ένα ημερήσιο και ένα εβδομαδιαίο σύνολο. Βάσει της ιεραρχίας των επιλογών **case**, η δεύτερη τυπώνει τις εβδομαδιαίες πωλήσεις, στης οποίας το σώμα (<συνθετη_προταση>) δεν υπάρχει η πρόταση **break**. Έτσι, αμέσως μετά, ενεργοποιείται αυτόματα η τρίτη **case** και τυπώνει το ημερήσιο σύνολο της Παρασκευής.

(γ) Εάν η ημέρα τυχαίνει να είναι η τελευταία του μήνα, τότε τυπώνεται και ένα μηνιαίο σύνολο. Ακολουθώντας την ιεραρχία των επιλογών **case**, η πρώτη τυπώνει τις μηνιαίες πωλήσεις. Επειδή στο σώμα (<συνθετη_προταση>) της πρώτης **case** δεν υπάρχει η πρόταση **break**, αμέσως μετά, ενεργοποιείται αυτόματα η δεύτερη **case** που τυπώνει τις εβδομαδιαίες πωλήσεις, τέλος ενεργοποιείται αυτόματα και η τρίτη **case** τυπώνοντας και το ημερήσιο σύνολο.

Σημείωση: Υπογραμμίζουμε ότι άλλες γλώσσες προγραμματισμού που δεν προσφέρουν αυτό που η πρόταση **switch** στην C++ προσφέρει, δηλαδή χωρίς την **break** στην προγούμενη επιλογή **case** να ενεργοποιείται αυτόματα η επόμενη **case**, έχουν περιορισμένες δυνατότητες μεταφοράς σε άλλη συνθήκη.

```
. . .
// Εκτυπώνει ημερήσια, εβδομαδιαία και μηνιαία σύνολα πωλήσεων
#include <iostream.h>
#include <stdio.h>
main( )
{
 float imerisia = 12,500;        // Κανονικά οι τιμές αυτές λαμβάνονται
 float evdomadiea = 94,305;      // από αρχείο αποθηκευμένο στον δίσκο
 float miniea = 310,155;         // και όχι όπως ανατίθενται εδώ
 char apandisi;
 int imera; // Τιμή ημέρας για να γίνει η κατάλληλη επιλογή της case
            // η imera παίρνει τιμές από 1 έως 5 (Δευτέρα έως Παρασκευή)
```



```
            // ή 6 εάν είναι τέλος του μήνα.
    cout << " Είναι τέλος του μήνα; Δώστε την κατάλληλη απάντηση (Y/N) : ";
    cin >> apandisi;
    if ( ( apandisi == 'Y' ) || ( apandisi == 'y' ) )
      { imera = 6; } // Τιμή για εκτύπωση μηνιαίας αναφοράς
    else
      { cout <<" Δώστε τιμή (από 1 έως 5) για τον καθορισμό της ημέρας, " <<
                " Πέμπτης έως Παρασκευής, αντίστοιχα " ;
        cin >> imera; }
    // Χρήση της πρότασης πολλαπλής επιλογής switch
    switch (imera)
    {
      case (6): { cout << " Η μηνιαία αναφορά είναι :" << minisa << "\n"; }
      case (5): { cout << " Η εβδομαδιαία αναφορά είναι :" << evdomadisa
                       << " \n"; }
      default: { cout << " Η ημερήσια αναφορά είναι :" << imerisia << " \n"; }
    }
    return(0);
}
```

Παρατηρήσεις:

1. Σε μια πραγματική εφαρμογή, τα σύνολα τιμών που αντιστοιχούν στις ημερήσιες, στις εβδομαδιαίες και στις μηνιαίες πωλήσεις θα λαμβάνονται από δίσκο αντί να ανατίθενται στην αρχή του προγράμματος.
2. Επίσης αντί να εκτυπώνονται ξεχωριστές αναφορές πωλήσεων θα μπορούσε πολλά σύνολα πωλήσεων, πλήρως ημερήσια, εβδομαδιαία και μηνιαία.
3. Η σειρά των προτάσεων **case** δεν είναι καθορισμένη. Είναι δυνατόν ο επανακαθορισμός των προτάσεων ώστε να γίνουν αποδοτικότερες. Εάν συνήθως επιλέγονται μια ή δύο συνθήκες, αυτές γράφονται στην αρχή της πρότασης **switch**. Για παράδειγμα, στο πρόγραμμα που μόλις παρουσιάστηκε, οι περισσότερες αναφορές εταιρών είναι ημερήσιες, αλλά η ημερήσια επιλογή είναι η τρίτη στις προτάσεις **case**. Με τον επανακαθορισμό των προτάσεων **case** έτσι ώστε η ημερήσια αναφορά να βρίσκεται στην αρχή, επιταχύνεται το πρόγραμμα επειδή η C++ δεν χρειάζεται να ελέγξει δύο εκφράσεις **case** που εκτελούνται σπάνια.

6. Το πρόγραμμα που παρουσιάζεται υλοποιεί την τελευταία παρατήρηση για αποτελεσματικό καθορισμό των προτάσεων **case** έτσι ώστε να επιταχύνεται το πρόγραμμμα της C++ .

```
. . .

// Εκτυπώνει μήνυμα ανάλογα με το τμήμα που εισάγεται
#include <iostream.h>
main( )
{
 char epilogi ;
 do    // Εμφάνισε τον κατάλογο των επιλογών (menu),
       // για να επιβεβαιώση ο χρήστης ότι εισάγει τη σωστή επιλογή
   { cout << " \n Επέλεξε το τμήμα σου : \n";
     cout << "P - Πωλήσεων \n";
     cout << "L - Λογιστήριο \n";
     cout << "M - Μηχανογραφικό \n";
     cout << "T - Ταμείο \n";
     cin >> epilogi ;
     // Εάν η επιλογή εισάγεται με πεζά τότε
     // μετάτρεψέ την σε κεφαλαία με τον πίνακα ASCII
     if ( ( epilogi >= 97 ) && (epilogi <= 122 ) )
       { epilogi- = 32; } // Αφαιρούνται οι σωστές τιμές για να μετατραπούν
                          // οι επιλογές σε κεφαλαία
 while ( ( epilogi = 'P' ) && ( epilogi = 'L' ) && ( epilogi = 'M' ) &&
           ( epilogi = 'T' ) );
 // Χρήση της πρότασης πολλαπλής επιλογής switch - Γράφεται πρώτα το
 // Μηχανογραφικό επειδή συμβαίνει συχνότερα
 switch (epilogi)
 {
  case ('M'): { cout << " Η συνάντησή σας είναι στις 10:45 ";
                break; }
  case ('P'): { cout << " Η συνάντησή σας ακυρώθηκε ";
                break; }
  case ('L'): { cout << " Η συνάντησή σας είναι στις 10:00 ";
                break; }
  case ('T'): { cout << " Η συνάντησή σας είναι στις 1:45 ";
                break; }
 }
 return(0);
}
```

4.6.1.6.2 Η Πρόταση goto

Οι πρώτες γλώσσες προγραμματισμού υλοποιούσαν τις κανονικές βαθμίδες, εκλογής και ανακύκλωσης ή και άλλα ευέλικτα βοηθήματα για τον προγραμματισμό, όπως των πολλαπλών επιλογών κ.ά., κατά έμμεσο τρόπο γιαυτό και οι γλώσσες αυτές ανήκουν στις μη κανονικές, όπως λέμε, γλώσσες (βλ. §1.4.2, υποσημείωση13).

Η πρόταση που βοηθούσε αποφασιστικά στην πραγματοποίηση ή στην κανονικοποίηση των παραπάνω βαθμίδων και βοηθημάτων στις μη κανονικές γλώσσες της εποχής ήταν η πρόταση εκτροπής **go to**. Πρόταση τόσο απαραίτητη όσο και επικίνδυνη αφού με την δυνατότητα να εκτρέψει το πρόγραμμα από την κανονική του ροή, βοηθούσε στην κωδικοποίηση αλλά άφηνε τον προγραμματιστή εκτεθειμένο στο να κάνει λογικά λάθη που δύσκολα μπορούσαν να εντοπιστούν. Τα λάθη αυτά οφείλονταν κυρίως σε λανθασμένες εκτροπές του προγράμματος ή μη έγκαιρες διορθώσεις των εκτροπών, ιδίως σε περιπτώσεις επέκτασης των προγραμμάτων. Επίσης, η συνηθισμένη για την εποχή της "τέχνης του προγραμματισμού" κατάχρηση της **go to** οδηγούσε στη γραφή προγραμμάτων μακαρονάδων, εξ αιτίας των πολλών εκτροπών, που τελικά ήταν προγράμματα δυσνόητα, ανεξέλεκτα και δυσκολοσυντηρούμενα.

Η C++ περιέχει την πρόταση **goto** ως ένα ακόμα εργαλείο προγραμματισμού, το οποίο μάλιστα θα πρέπει να το χρησιμοποιούμε όσο πιό σπάνια γίνεται και με πολύ προσοχή. Σημειώνουμε ότι, η κωδικοποίηση των κανονικών βαθμίδων και άλλων βοηθημάτων προγραμματισμού στη C++ γίνεται άμεσα, χωρίς την συνδρομή της **goto**, αφού η C++ ανήκει στις κανονικές γλώσσες (βλ. §1.4.2, υποσημείωση13).

Η σύνταξη της **goto** είναι απλούστατη και διατυπώνεται με τη βοήθεια της BNF ως εξής:

1. <πρόταση_goto> ::= **goto** <προσδιοριστής_ετικέτας>;
   <ετικέτα_πρότασης> : <πρόταση>; ,

2. <ετικέτα_πρότασης> ::= <προσδιοριστής_ετικέτας> ,

3. <προσδιοριστής_ετικέτας> ::= <προσδιοριστής>,

4. <προταση> ::= <αναθεση>|
<αναγνωση_εισοδου>|<γραφη_εξοδου> .

Δίδουμε κάποια παραδείγματα προτάσεων ετικετών (<προταση_ετικετα>):

```
pliromi: cout <<" Τοποθετήστε τα δελτία πληρωμής στον εκτυπωτή \n";
xana: cin >> onoma;
telos: cout <<" Τέλος της διαδικασίας \n";
ypologismos: poso = (oliko_poso / .7) + 32.67 ;
```

Παράδειγμα: Το πρόγραμμα που ακολουθεί έχει ένα πρόβλημα το οποίο είναι αποτέλεσμα της **goto**, είναι όμως μια από τις καλλίτερες εφαρμογές της. Το πρόγραμμα αποτελείται από έναν ατελεύτητο βρόγχο (βλπ §4.6.1.3 και υποσημείωση 59). Το σώμα του ατελεύτητου βρόγχου είναι το σώμα της ανακύκλωσης που δημιουργεί η **goto**. Ο ατελεύτητος βρόγχος διακόπτεται με **Ctrl+Break** ή εάν ζητήσετε από τον διαχειριστή του προγράμματός σας να ακυρώσει το πρόγραμμα.

```
. . .
// Δείχνει την χρήση της goto
// Το πρόγραμμα τελειώνει μόνο όταν ο χρήστης πληκτρολογήσει Ctrl+Break
#include <iostream.h>

main( )
{
 xana : cout << " Αυτό το μήνυμα: \n";
 cout << "\t επαναλαμβάνεται \n";
 cout << "\n\t ξανά και ξανά \n";
 goto xana; // Επαναλάμβανε συνεχώς
 return(0);
}
```

Παρατήρηση: Καθώς εξοικειώνεστε με τον προγραμματισμό και τμηματικοποιείτε τα προγράμματά σας σε μικρότερες ενότητες χρησιμοποιώντας και συναρτήσεις, η **goto** θα γίνεται όλο και λιγότερο σημαντική. Η εξοικείωση με την **goto** χρειάζεται για να γίνονται κατανοητά τα προγράμματα που την χρησιμοποιούν, ίσως κάποτε χρειαστεί να διορθώσετε τον κωδικό κάποιου που την χρησιμοποιεί.

4.6.1.6.3 Ερωτήσεις για Επανάληψη και Εμπέδωση

1. Πως η **goto** εκτρέπει από την κανονική σειρά την εκτέλεση ενός προγράμματος;

2. Ποιά πρόταση μπορεί να αντικαταστήσει μια **if_else_if** ;

3. Ποιά πρόταση τελειώνει, σχεδόν πάντα, κάθε πρόταση **case** με μια **switch** ;

4. Είναι αληθές ή ψευδές ότι η σειρά των προτάσεων **case** δεν επιδρά στην απόδοση του προγράμματος;

5. Ξαναγράψατε το τμήμα κώδικα που ακολουθεί χρησιμοποιώντας μια πρόταση **switch** .

```
if ( metritis —1 )
  { cout << "Alpha";}
else if ( metritis —2 )
        { cout << "Beta";}
      else if ( metritis —3 )
             { cout << "Gamma";}
           else
             { cout << "Other";}
```

6. Ξαναγράψατε το τμήμα κώδικα που ακολουθεί χρησιμοποιώντας ένα βρόγχο **do_while** .

```
Erotisi:
   cout << " Ποιό είναι το όνομά σου; ";
   cin >> onoma;
   if ( ( onoma[0] >'A' ) || ( onoma[0] < 'Z') )
     { goto Erotisi;} // Ρώτα συνέχεια μέχρι ο χρήστης να
                      // εισάγει ένα έγκυρο όνομα
```

4.6.2 Παραδείγματα

4.6.2.1 Διάταξη Ακολουθίας

4.6.2.1.1 Μέθοδος Ανταλλαγής των Μεγίστων Στοιχείων

Αντικειμενικός σκοπός του προβλήματος είναι να γίνουν οι αναγκαίες ανταλλαγές των μελών της ακολουθίας:

$$X[1], X[2], \ldots, X[N] \text{ έτσι ώστε να προκύψει } X[1] \le X[2] \le \ldots \le X[N].$$

Βήματα :
1. Προσδιορισμός στοιχείου που έχει την μεγίστη τιμή,
2. Ανταλλαγής αυτού του στοιχείου με το N-στο στοιχείο,

3. Επανάληψη των πράξεων (1) & (2) γιά τις υπακολουθίες των πρώτων N-1 στοιχείων, κατόπι των πρώτων N-2 στοιχείων, κ.ο.κ. , μέχρι και της υπακολουθίας που έχει δύο μόνο στοιχεία.

Αλγόριθμος :

```
/* Διάταξη των στοιχείων ακολουθίας κατά αύξουσες τιμές        */
/* Υποθέτουμε ότι το πλήθος των στοιχείων της ακολουθίας δεν   */
/* υπερβαίνει το 100                                            */
διαδικασια ΑΝΤΜΕΓΣΤ;
 /* δηλώσεις */
δηλωση Χ [1:100] ακεραια παραταξη;
δηλωση  (Ι, Ν, Μ, ΜΕΓ, Π, Κ) ακεραια;
αρχη
 /* εισαγωγή τιμών */
 διαβασε Ν;
 για Ι ← 1εως Ν επαναλαβε
    ( διαβασε Χ[Ι] ; )
 γραψε Ν;
 για Ι ← 1εως Ν επαναλαβε /* εκτύπωση μη διατεταγμένης ακολουθίας */
    ( διαβασε Χ[Ι] ; )
 Κ ← Ν;  /* καθορισμός αρχικού μήκους υποακολουθίας */
 εφοσον Κ > 1 επαναλαβε
    /* προσδιορισμός μεγίστου στοιχείου υποακολουθίας  μήκους Κ */
      1 ( ΜΕΓ ← Χ[1];
          Μ ← 1;
          Ι ←2;
      εφοσον Ι ≤ Κ επαναλαβε
            2 ( εαν Χ[Ι] > ΜΕΓ
                  τοτε  3 ( ΜΕΓ ← Χ[Ι];
                            Μ ← Ι; )3
                Ι ← Ι + 1; )2
    /* ανταλλαγή τιμών μεγίστου και τελευταίου */
      Π ← Χ[Μ];
```

```
        X[M] ← X[K];
        X[K] ← Π;
    /* καθορισμός μήκους επόμενης υποακολουθίας */
        K ← K - 1; )1
/* εκτύπωση διατεταγμένης ακολουθίας */
 για I ← 1εως N επαναλαβε
   ( διαβασε X[I] ; )
τελος ΑΝΤΜΕΓΣΤ
```

4.6.2.1.2 Μέθοδος Προώθησης των Μεγάλων Τιμών ή της Φυσαλίδας (Bouble Sort)

Αντικειμενικός σκοπός του προβλήματος είναι να γίνουν οι αναγκαίες ανταλλαγές των μελών της ακολουθίας :

$$X[1], X[2], \ldots, X[N] \text{ έτσι ώστε να προκύψει } X[1] \le X[2] \le \ldots \le X[N].$$

Η μέθοδος αυτή διατάσσει τα στοιχεία της ακολουθίας προωθόντας τις μεγάλες τιμές προς τα δεξιά, συγκρίνοντας κάθε στοιχείο με το επόμενό του, και ανταλλάσσει τις τιμές τους, μόνο εάν το προηγούμενο βρέθηκε μεγαλύτερο από το επόμενο. Εάν η διαδικασία αυτή επαναληφθεί γιά τα πρώτα N-1 στοιχεία, η μέγιστη τιμή μεταξύ των στοιχείων αυτών προωθείται στην N-1 θέση. Συνεχίζοντας κατ' αυτόν τον τρόπο φθανουμε σε μία ακολουθία με δύο στοιχεία και επιτυγχάνουμε την διάταξή της. Αν κατά την διάρκεια της διαδικασίας σε μία υποακολουθία διαπιστωθεί ότι δεν έλαβε χώρα καμμία ανταλλαγή, τότε αυτό σημαίνει ότι όλα τα στοιχεία της υποακολουθίας είναι διατεταγμένακαι επομένως η διαδικασία πρέπει να τερματιστεί, αυτό επιτυγχάνεται με την βοήθεια λογικού διακόπτει, δηλαδή μίας λογικής μεταβλητής.

Αλγόριθμος :

```
/* Διάταξη των στοιχείων ακολουθίας κατά αύξουσες τιμές        */
/* Υποθέτουμε ότι το πλήθος των στοιχείων της ακολουθίας δεν   */
/* υπερβαίνει το 100                                            */
διαδικασια ΠΡΟΩΘΜΕΓΤΙΜ;
 /* δηλώσεις */
δηλωση  X [1:100] ακεραια παραταξη;
```

```
δηλωση (I, K, N, Π) ακεραια;
δηλωση ΣΗΜΑ λογικη; /* Σηματοδότης Διακόπτης */
αρχη
 /* εισαγωγή τιμών */
 διαβασε N;
 για I ← 1εως N επαναλαβε
    ( διαβασε X[I] ; )

 γραψε N;
 για I ← 1εως N επαναλαβε /* εκτύπωση μη διατεταγμένης ακολουθίας */
    ( διαβασε X[I] ; )

 K ← N; /* καθορισμός αρχικού μήκους υποακολουθίας */

 εφοσονK > 1 επαναλαβε /* έλεγχος μήκους υποακολουθίας */
       1( I ← 1;
          ΣΗΜΑ ← ΟΧΙ; /* ο ΣΗΜ ορίζεται αρχικά όχι γιά να      */
                       /* διαπιστωθεί αν έγινε προώθηση τιμών */
       εφοσονI ≤ K επαναλαβε /* βρόχος προώθησης */
        2( εαν X[I] > X[I+1]
              τοτε 3( Π ← X[I];
                      X[I] ← X[I+1];
                      X[I+1] ← Π;
                      ΣΗΜΑ ← ΝΑΙ; )3 /* έγινε προώθηση */
          I ← I +1; )2

       εαν ΣΗΜΑ
          τοτε K ← K - 1 /* να συνεχιστεί η προώθηση */
          αλλως K ← 0; )1 /* δεν χρειάζεται προώθηση */

 για I ← 1εως N επαναλαβε /* εκτύπωση διατεταγμένης ακολουθίας */
    ( διαβασε X[I] ; )
τελος ΠΡΟΩΘΜΕΓΠΙΜ
```

4.6.2.2 Συγχώνευση δύο Διατεταγμένων Πινάκων

Αντικειμενικός σκοπός η συγχώνευση δύο διατεταγμένων πινάκων Α και Β σε ένα πίνακα Γ που είναι διατεταγμένος.Η μέθοδος αυτή γιά να εφαρμοστεί θα πρέπει και οι δύο πίνακες να είναι διατεταγμένοι, διαφορετικά πρέπει πρώτα να διαταχθούν. Ο πρώτος

αριθμός του πίνακα Α συγκρίνεται με τον πρώτο του Β, ο μικρότερος των δύο θα αποτελέσει το πρώτο στοιχείο του πίνακα Γ. Ο αριθμός που δεν καταχωρήθηκε συγκρίνεται με τον επόμενο αυτού που καταχωρήθηκε, η διαδικασία συνεχίζεται μέχρι τα στοιχεία κάποιου από τους δύο πίνακες να εξαντληθούν. Στην περίπτωση αυτή τα υπόλοιπα στοιχεία του μη εξαντλημένου πίνακα τα οποία είναι διατεταγμένα, θα αποτελέσουν το ένα μετά το άλλο στοιχεία του Γ. Στην διαδικασία προβλέπεται αν υπαρξουν ίσα στοιχεία στους πίνακες Α και Β, το ένα μόνο θα καταχωρείται στον Γ.

Αλγόριθμος :

```
διαδικασια ΣΥΓΧΩΝ;

 /* δηλωσεις */
δηλωση  A [1: N+1], B [1: M+1], Γ [1: N+M] ακεραια παραταξη;
δηλωση  (I, J K, L, M, N, P) ακεραια;

αρχη
 /* εισαγωγή τιμών */
 διαβασε N, M;

 για I ← 1 εως N επαναλαβε
    ( διαβασε A[I]; )

 για J ← 1 εως M επαναλαβε
    ( διαβασε B[J]; )

 /* καθορισμός αρχικών τιμών */
 A[N+1] ← 0; B[M+1] ← 0;
 K ← 1; I ← 1; J ← 1;

 εφοσον A[I] ≠ 0 & B[J] ≠ 0 επαναλαβε  /* διαδικασία συγχώνευσης */
    1 ( εαν A[I] ≤ B[J]
          τοτε 2 ( εαν A[I] = B[J]
                       τοτε 3 ( Γ[K] ← A[I];
                                K ← K + 1;
                                I ← I + 1;
                                J ← J + 1 )3

                       αλλως 3 ( Γ[K] ← A[I];
```

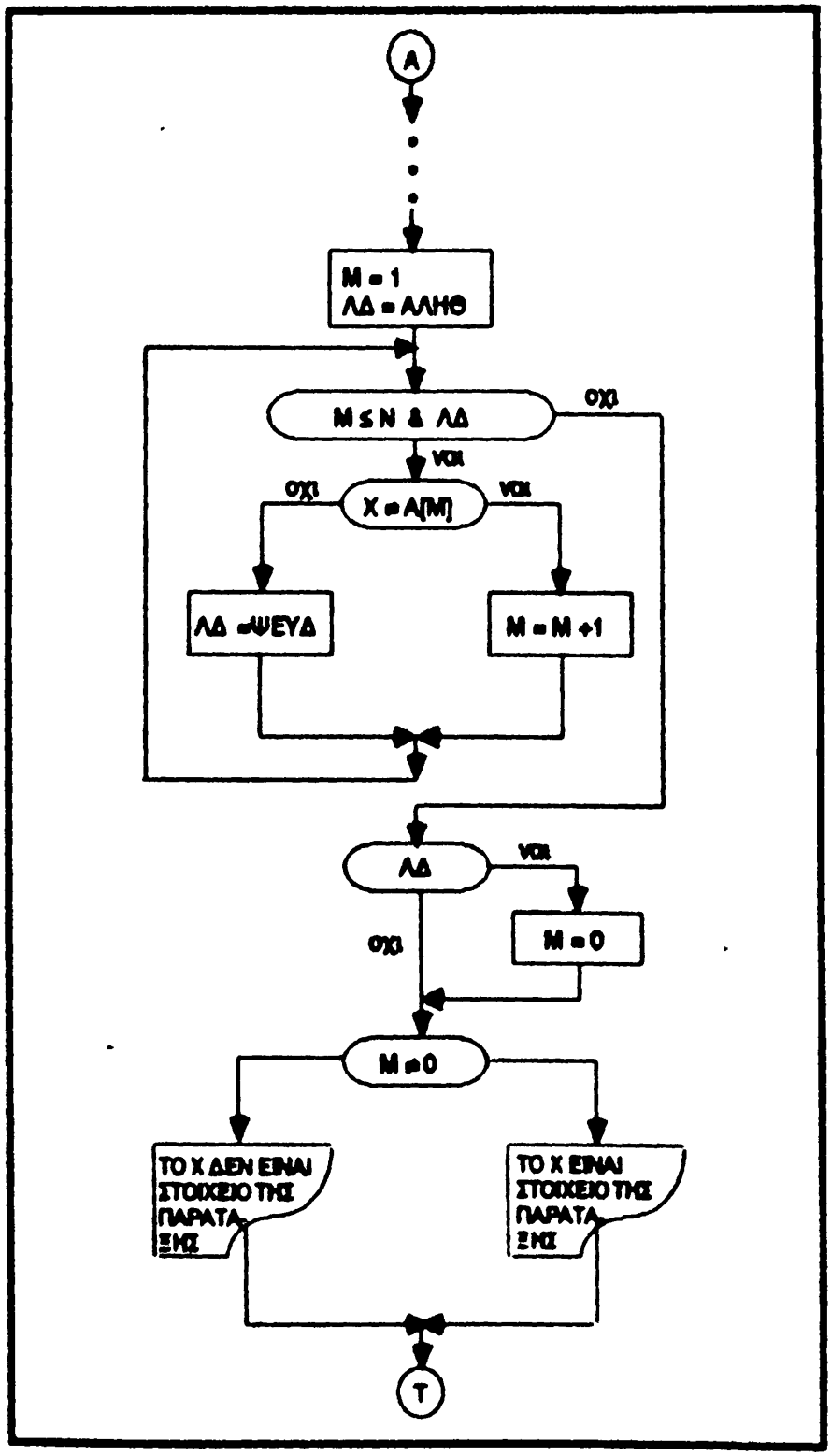
```
                    I ← I + 1;
                    K ← K + 1 )3 )2

        αλλως 2( Γ[K] ← B[J];
                 J ← J + 1;
                 K ← K + 1; )2 )1

  εαν A[I] ≠ 0
     τοτε 1( εαν B[J] = 0
                 τοτε 2( για L ← I εως N επαναλαβε
                         3( Γ[K] ← A[L];
                            K ← K+1)3 )2
     αλλως 1( εαν B[J] ≠ 0
                 τοτε 2( για L ← J εως M επαναλαβε
                         3( Γ[K] ← B[L];
                            K ← K+1; )3 )2 )1
     P ← K - 1;
  /* εκτύπωση διατεταγμένου πινακα Γ */
  γραψε P;
  για L ← 1 εως P επαναλαβε
     ( γραψε Γ[L] ;)
τελος ΣΥΓΧΩΝ
```

4.6.2.3 Διερεύνηση Πινάκων

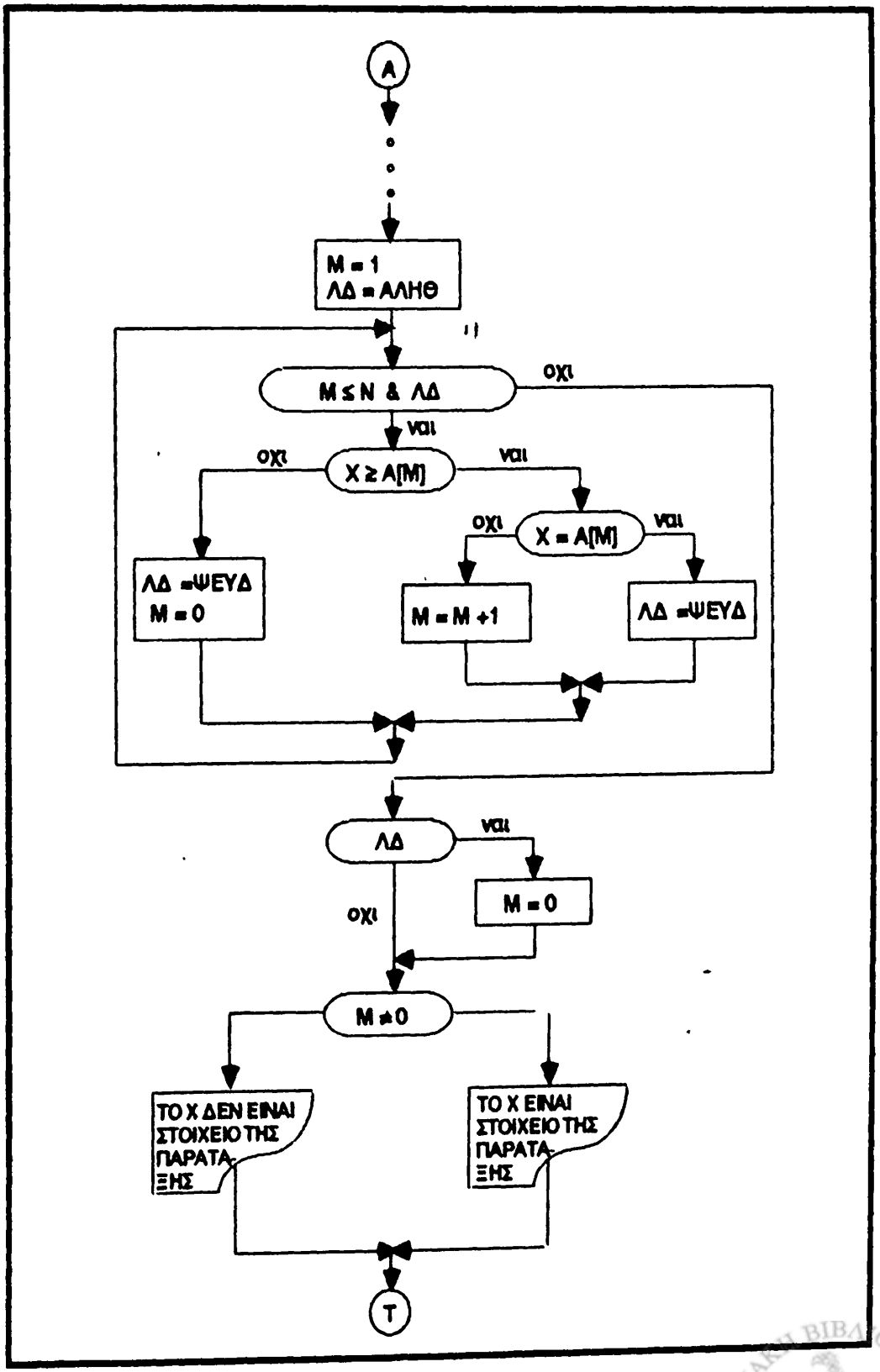
4.6.2.3.1 Η Μέθοδος της Διαδοχικής Διερεύνησης

α) **Αντικειμενικός σκοπός** η διερεύνηση μίας μη διατεταγμένης ακολουθίας γιά να βρεθεί αν ένας αριθμός ανήκει ή όχι στην ακολουθία.

Στο **λογικό διάγραμμα** περιγράφεται η διαδικασία διαπίστωσης αν ένας αριθμός Χ ανήκει στην ακολουθία, συγκρίνει διαδοχικά κάθε στοιχείο A[I],(με I = 1(2)N) της ακολουθίας με το Χ έως ότου είτε η λογική έκφραση A[I] ≠ Χ πάρει την τιμή ψευδής, είτε εξαντληθούν όλα τα στοιχεία του πίνακα, ενώ η έκφραση θα διατηρεί την τιμή αληθής, που θα σημαίνει ότι η τιμή του Χ δεν ανήκει στην ακολουθία. Όπου: Α το όνομα της ακολουθίας,



Λογικό Διάγραμμα 4.6.2.3.1α : Διαδοχική Διερεύνηση



Λογικό Διάγραμμα 4.6.2.3.1β : Διαδοχική Διερεύνηση

Ν το μήκος της, Χ η (εισαγόμενη) τιμή που αναζητείται και Μ ο δείκτης του στοιχείου που έχει την τιμή Χ (εξαγόμενο). Εάν η τιμή Χ δεν ανήκει στην ακολουθία Α, η τιμή του Μ, είναι 0 αν ανήκει η τιμή του Μ είναι η τάξη του στοιχείου στην ακολουθία (βλπ λογικό διάγραμμα 4.6.2.3.1α).

β) **Αντικειμενικός σκοπός** η διερεύνηση μίας διατεταγμένης ακολουθίας γιά να βρεθεί αν ένας αριθμός ανήκει ή όχι στην ακολουθία.

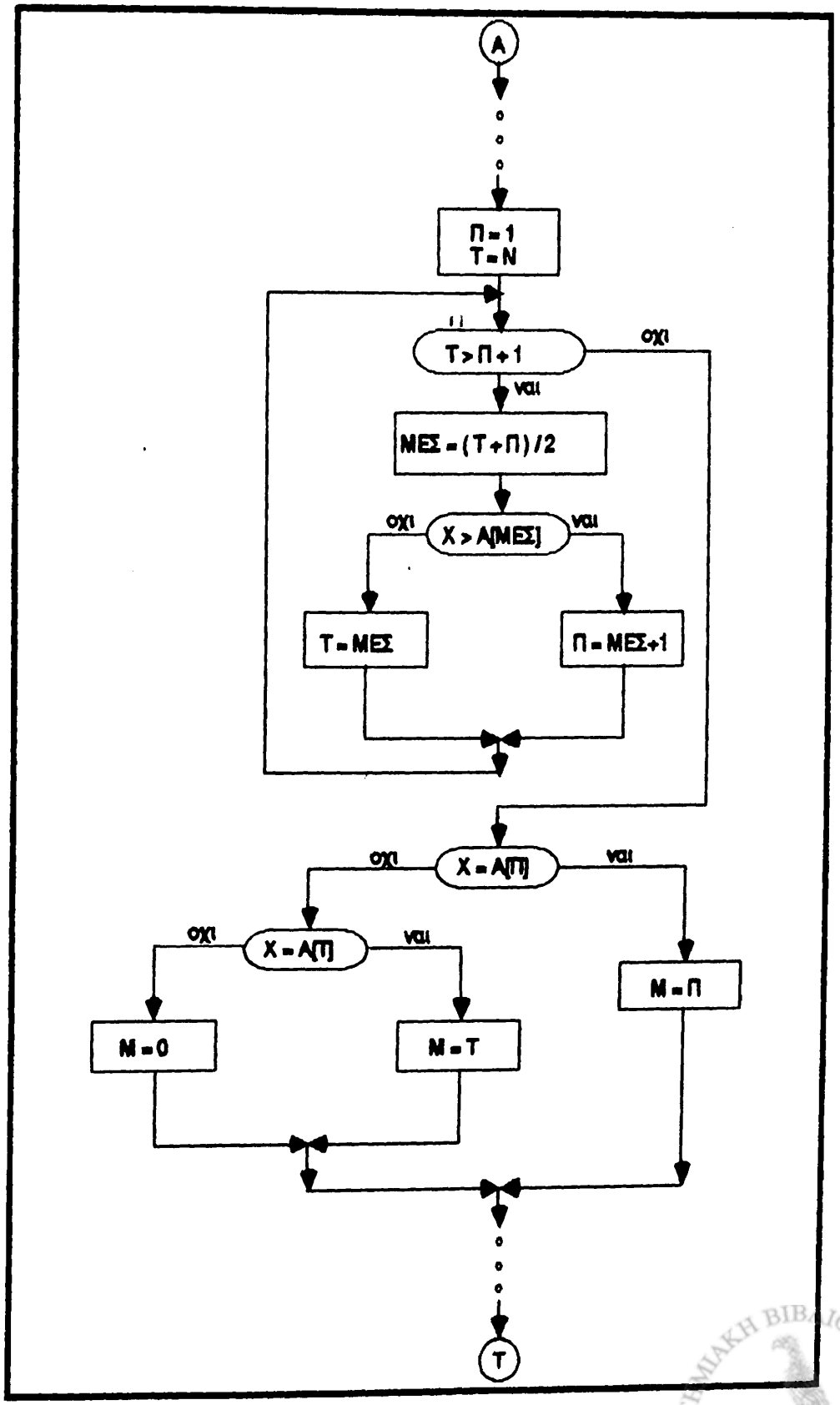
Το **λογικό διάγραμμα** διαμορφώνετα ανάλογα (βλπ λογικό διάγραμμα 4.6.2.3.1β).

4.6.2.3.2 Η Μέθοδος της Δυαδικής Διερεύνησης

Αντικειμενικός σκοπός η διερεύνηση μίας διατεταγμένης ακολουθίας γιά να βρεθεί αν ένας αριθμός ανήκει ή όχι στην ακολουθία.

Στο **λογικό διάγραμμα** (βλπ λογικό διάγραμμα 4.6.2.3.2) περιγράφεται η διαδικασία διαπίστωσης αν ένας αριθμός Χ ανήκει στην ακολουθία, η ακολουθία είναι διατεταγμένη και αυτό μας επιτρέπει να συντομεύσουμε την διαδικασία της διερεύνησης, διαφούμε την ακολουθία σε δύο υποακολουθίες βρίσκοντας το μέσο ΜΕΣ και εξετάζουμε αν Χ > Α[ΜΕΣ] ή Χ < Α[ΜΕΣ] , δηλαδή σε πιά από τις δύο υποακολουθίες μπορεί ο αριθμός μας να ανήκει, η διαδικασία επαναλαμβάνεται γιά τις εκλεγμένες σε κάθε ανακύκλωση υποακολουθίες με τα προαναφερθέντα κριτήρια, μέχρι να καταλήξουμε σε υποακολουθία με δύο αριθμούς, αν ο αριθμός μας ανήκει στην δοθείσα ακολουθία Α τότε θα ταυτίζεται με έναν από τους δύο αριθμούς. Όπου : Α το όνομα της διατεταγμένης ακολουθίας, Ν το μήκος της, Χ η (εισαγόμενη) τιμή που αναζητείται και Μ ο δείκτης του στοιχείου που έχει την τιμή Χ (εξαγόμενο). Εάν η τιμή Χ δεν ανήκει στην ακολουθία Α, η τιμή του Μ, είναι 0 αν ανήκει η τιμή του Μ είναι η τάξη του στοιχείου στην ακολουθία, Π είναι ο πρώτος αριθμός του εκάστοτε διαστήματος, Τ ο τελευταίος αριθμός του εκάστοτε διαστήματος και ΜΕΣ το μέσον του εκάστοτε διαστήματος.

Προσοχή: Στους αναγνώστες αφήνονται σαν άσκήσεις να συμπληρώσουν τις διαδικασίες που παραλείπονται από κάθε μία από τις παραπάνω εφαρμογές δηλαδή το διάγραμμα ροής ή τον αλγορίθμο σε ΕΑΓ ή την κωδικοποίηση σε C++.



Λογικό Διάγραμμα 4.6.2.3.2 : Δυαδική Διερεύνηση

4.6.3 Ασκήσεις για Επανάληψη και Εμπέδωση

1. Να γραφούν σε ΕΑΓ τα παραδείγματα της § 4.6.2 και να κωδικοποιηθούν σε C++.

2. Να ξαναγραφεί ο κώδικας των παραδειγμάτων της § 4.6.2 ώστε να χειρίζονται:
(α) χαρακτήρες και (β) αλυσίδες χαρακτήρων.

4.7 Εισαγωγή Δεδομένων / Εξαγωγή Αποτελεσμάτων

Η C++, όπως έχουμε ήδη αναφέρει δεν έχει προτάσεις εισόδου δεδομένων / εξόδου αποτελεσμάτων, ή απλούστερα προτάσεις εισόδου / εξόδου, αντί αυτών διαθέτει τελεστές και συναρτήσεις για είσοδο / έξοδο. Στην παράγραφο αυτή θα παρουσιάσουμε τους τελεστές εισόδου / εξόδου, **cin** και **cout**.

4.7.1 Ο Τελεστής cout

Ο τελεστής **cout** στέλνει τα αποτελέσματα στην κανονική συσκευή εξόδου που είναι η οθόνη, όμως είναι δυνατόν να ανακατευθυνθεί η κανονική έξοδος και σε άλλη συσκευή.

Η σύνταξη του cout διαφέρει από αυτή των άλλων προτάσεων της C++ και είναι:

cout << <ακολουθια_αποτελεσματων>;

όπου: <ακολουθια_αποτελεσματων> ::= <αποτελεσμα> |
<αποτελεσμα> << <ακολουθια_αποτελεσματων>

4.7.1.1 Εκτύπωση Αλυσίδων Χαρακτήρων

Οδηγίες για την εκτύπωση στην οθόνη:

1. Μιας σταθερής αλυσίδας χαρακτήρων, π.χ. "η λύση είναι: ", εφαρμόζετε τον τύπο της §4.7.1 **cout** << <ακολουθια_αποτελεσματων>; ως εξής:

cout << "η λύση είναι: ";

εκτύπωση στην οθόνη: η λύση είναι:

2. Μιας ακολουθίας σταθερών αλυσίδων χαρακτήρων, έτσι ώστε η κάθε αλυσίδα να εκτυπώνεται σε άλλη γραμμή, συμπληρώνετε στο τέλος κάθε αλυσίδας τον χαρακτήρα

"\n", ο χαρακτήρας αυτός στέλνει τον **δρομέα** (φωτεινός δείκτης της οθόνης - cursor) στην επόμενη γραμμή ανεξάρτητα με τη θέση εισαγωγής της, π.χ. :

cout << "εισαγωγή δεδομένων :**\n**"; ή **cout** << "εισαγωγή δεδομένων :" << "**\n**";
cout << "όρισε τον τερματιστή :**\n**"; ή **cout** << "όρισε τον τερματιστή :" << "**\n**";
cout << "δώσε τιμή στον ARITHM :**\n**"; ή **cout** << "δώσε τιμή στον ARITHM :" << "**\n**";

εκτύπωση στην οθόνη: εισαγωγή δεδομένων :
όρισε τον τερματιστή :
δώσε τιμή στον ARITHM :

3. Μιας αλυσίδας χαρακτήρων που έχει εναποθηκευθεί σε μια παράταξη χαρακτήρων, πληκτρολογείτε το όνομα της παράταξης μέσα στον cout, π.χ.:

σάν : **char** name[ ] = "Georgios Prasinos";
τότε : **cout** << name;

εκτύπωση στην οθόνη: Georgios Prasinos

4. Μιας κενής γραμμής μετά την εκτυπωσή ενός χαρακτήρα ή μιας αλυσίδας, συμπληρώνετε μεταξύ του χαρακτήρα ή του τέλους της αλυσίδας στον cout και του δεξιού διπλού εισαγωγικού ", τους χαρακτήρες **\n\n**, ή πληκρολογείτε <<"**\n\n**" πριν τον οριοθέτη ; , π.χ.:

cout << "εισαγωγή δεδομένων :**\n\n**"; ή **cout** << "εισαγωγή δεδομένων :" << "**\n\n**";
cout << "όρισε τον τερματιστή :**\n**"; ή **cout** << "όρισε τον τερματιστή :" << "**\n**";
cout << "δώσε τιμή στον ARITHM :**\n**"; ή **cout** << "δώσε τιμή στον ARITHM :" << "**\n**";

εκτύπωση στην οθόνη: εισαγωγή δεδομένων :

όρισε τον τερματιστή :
δώσε τιμή στον ARITHM :

5. Ενός πίνακα αριθμών, οι οποίοι στην πραγματικότητα είναι αλυσίδα αριθμητικών χαρακτήρων, χρησιμοποιείστε τον στηλοθέτη **\t**. Ο στηλοθέτης πληκτρολογείται μεταξύ των εκτυπωμένων αριθμών, π.χ., κατάλογος ονομάτων και αριθμών επιτυχιών μια ομάδας για τις τρείς εβδομάδες μιας περιόδου, π.χ.:

cout << "Αίας**\t**Αετός**\t**Δόξα**\t**Νίκη**\t**Ατρόμητος**\n**";

```
cout << "3\t6\t2\t0\t1\n";
cout << "2\t4\t0\t0\t6\n";
cout << "3\t1\t1\t0\t5\n";
```

εκτύπωση στην οθόνη:

Αίας	Αετός	Δόξα	Νίκη	Ατρόμητος
3	6	2	0	1
2	4	0	0	6
3	1	1	0	5

Παρατηρούμε ότι οι αριθμοί τυπώνονται, σε ίση απόσταση μεταξύ τους, κάτω από τα ονόματα ακόμα και αν το μήκος αυτών διαφέρει. Σημειώνουμε ότι ο χαρακτήρας \t αναγκάζει το επόμενο όνομα ή τιμή στην επόμενη θέση του στηλοθέτη, δηλαδή κάθε οκτώ χαρακτήρες.

4.7.2 Τελεστές Ελέγχου Εκτύπωσης

Πολλές φορές στον προγραμματισμό η ανάγκη για επιπρόσθετο έλεγχο της εξόδου των αποτελεσμάτων, πέραν του στοιχειώδους(βλπ §4.7.1), είναι επιτακτική. Παραθέτουμε κάποιες από τις επιπλέον δυνατότητες του cout.

Οδηγίες για Ελεγχόμενες Εκτυπώσεις:

1. Για να καθοριστεί το μήκος του πεδίου εκτύπωσης, ώστε να εκτυπωθούν τα αποτελέσματα με κενά ανάμεσά τους ή σε ομοιόμορφες στήλες, χρησιμοποιείται η συνάρτηση διαχειριστής:

setw(<μηκος_πεδιου>**)**

επιπλέον θα πρέπει να περιληφθεί στο πρόγραμμα η οδηγία **#include <iomanip.h>** , η οποία γράφεται πριν τη συναρτηση **main( )** αλλά μετά την οδηγία **#include**< iostream.h>, ο λόγος είναι ότι το αρχείο iomanip.h περιγράφει τη λειτουργία της **setw(**<μηκος_πεδιου>**)** στον μεταγλωττιστή.

Σημείωση: Αν δεν οριστεί μήκος πεδίου αρκετά μεγάλο, ώστε να περιέχει τον αριθμό, τότε η C++ αγνοεί το μήκος που απαιτείθηκε και εκτυπώνει ολόκληρο τον αριθμό.

Παραδείγματα:
(α) Να τυπωθούν οι αριθμοί 456 και 87654 χρησιμοποιώντας απλά τον cout, θα έχουμε:

```
cout << 456 << 87654 <<"\n";
```

εκτύπωση στην οθόνη: 45687654

(β) Να τυπωθούν οι αριθμοί 456 και 87654 παρεμβάλλοντας κενά ανάμεσά στους και χρησιμοποιώντας τον cout και την setw( ):

```
. . .
#include <iostream.h>
#include <iomanip.h>
main( )
. . .
cout << setw(6) << 456 << setw(6) << 87654 <<"\n";
. . .
```

εκτύπωση στην οθόνη: 456 87654

(γ) Να τυπωθούνσε ομοιόμορφες στήλες οι αριθμοί 456 και 87654 χρησιμοποιώντας τον cout και την setw( ):

```
. . .
#include <iostream.h>
#include <iomanip.h>
main( )
 . . .
 cout << setw(6) << 456 <<"\n";
 cout << setw(6) << 87654 <<"\n";
 . . .
```

εκτύπωση στην οθόνη: 456
87654

2. Για να παρεμβληθούν χαρακτήρες στα κενά που προκύπτουν σε εκτυπώσεις που καθορίζεται το μήκος του πεδίου εκτύπωσης (βλπ οδηγία 1), χρησιμοποιείται η συνάρτηση:

cout.fill('<χαρακτήρας>**')**

Σημείωση: Η κλήση της συνάρτησης αυτής γίνεται μια φορά και αφορά όλα τα πεδία.

Παράδειγμα: Να τυπωθούν οι αριθμοί 456 και 87654 παρεμβάλλοντας τελείες ανάμεσά

στους και χρησιμοποιώντας τον cout, την setw( ) και την cout.fill(' ').

```
. . .
#include <iostream.h>
#include <iomanip.h>
main( )
 . . .
 cout.fill(' .')
 cout << setw(6) << 456 <<"\n";
 cout << setw(6) << 87654 <<"\n";
 . . .
```

εκτύπωση στην οθόνη: . . . 4 5 6
. 8 7 6 5 4

3. Για να καθοριστούν τα δεκαδικά ψηφία, σε αριθμούς κινητού δεκαδικού σημείου, χρησιμοποιούμε τη συνάρτηση:

setprecision(<πλήθος_δεκαδικων_ψηφιων>**)**

πάντοτε σε συνδυασμό με την setw( ). Αν ο αριθμός έχει περισσότερα δεκαδικά ψηφία από αυτά που καθορίζονται από το όρισμα της συνάρτησης, τοτε αυτά που περισεύουν κόβονται και το αποτέλεσμα στρογγυλοποιείται στο τελευταίο δεκαδικό του ψηφίο.

Παράδειγμα: Να τυπωθούν οι αριθμοί 4.56 και 8.7657, κάνοντας συγχρόνως χρήση των συναρτήσεων setw(6) και setprecision(3).

```
. . .
#include <iostream.h>
#include <iomanip.h>
main( )
 . . .
 cout << setw(6) << setprecision(3) << 4.56 <<"\n";
 cout << setw(6) << setprecision(3) << 8.7657 <<"\n";
 . . .
```

εκτύπωση στην οθόνη: 4.56
8.766

4. Για να εμφανιστούν μεμονωμένοι χαρακτήρες μιας αλυσίδας χαρακτήρων,

χρησιμοποιούμε τη συνάρτηση:

cout.put(<στοιχειο_παραταξης_χαρακτηρων> **)**

Υπενθυμίζουμε ότι σε προγράμματα που αναφέρονται συναρτήσεις που χειρίζονται αλυσίδες χαρακτήρων πρέπει να περιέχεται η οδηγία:

#include <string.h>

πριν τη συναρτηση **main( )** αλλά μετά την οδηγία **#include**< iostream.h>, έτσι ώστε ο μεταγλωττιστής να παρέχει το αρχείο string.h για να διευκολύνει την συνάρτηση **cout.put( )** να λειτουργήσει κανονικά.

Παράδειγμα: Να εκτυπωθεί αντιστραμένη η αλυσίδα χαρακτήρων : "ΝΙΨΟΝ ΑΝΟΜΗΜΑΤΑ ΜΗ ΜΟΝΑΝ ΟΨΙΝ", χρησιμοποιώντας τη συνάρτηση cout.put( ).

```
. . .
#include <iostream.h>
#include <string.h>
main( )
{
  int i;
  char str[29] = "ΝΙΨΟΝ ΑΝΟΜΗΜΑΤΑ ΜΗ ΜΟΝΑΝ ΟΨΙΝ"
  for ( i = strlen( str) -1; i > = 0; i-- )
    {  cout.put(str[i]); }
  return(0);
}
```

εκτύπωση στην οθόνη: ΝΙΨΟ ΝΑΝΟΜ ΗΜ ΑΤΑ ΜΗΜΟΝΑ ΝΟΨΙΝ

4.7.3 Ο Τελεστής cin

Ο τελεστής **cin** είναι ένας τρόπος εισαγωγής δεδομένων από το πληκτρολόγιο. Όταν δηλαδή τα προγράμματα φθάνουν στη γραμμή που έχει τον cin ο χρήστης μπορεί να εισάγει τιμές (δεδομένα) απευθείας σε ματαβλητές. Το πρόγραμμα μετά από επεξεργασία αυτών των μεταβλητών (δεδομένων) παράγει τα αποτελέσματα που προωθούνται στην έξοδο.

Ο τελεστής **cin** μοιάζει με τον τελεστή **cout** στην σύνταξη, η οποία είναι ανάλογη:

cin >> <ακολουθια_δεδομενων>**;**

όπου: <ακολουθια_δεδομενων> ::= <δεδομενα>|
<δεδομενα>>> <ακολουθια_δεδομενων>

Το αρχείο **iostream.h** της οδηγίας **#include**< iostream.h> περιέχει την πληροφορία που χρειάζεται η C++ για τη χρησιμοποίηση του τελεστή **cin**, έτσι πρέπει να περιλαμβάνεται παντοτε όταν χρησιμοποιείται ο cin.

Επισημαίνουμε ότι υπάρχει μεγάλη διαφορά μεταξύ του cin και των προτάσεων ανάθεσης π.χ. ARITHM = 235. Με τις προτάσεις ανάθεσης ανατίθενται συγκεκριμένες τιμές στις μεταβλητές, οι οποίες είναι γνωστές από τη συγγραφή του προγράμματος. Έτσι τα αποτελέσματα είναι πάντοτε τα ίδια αφού οι ίδιες τιμές ανατίθενται στις ίδιες μεταβλητές. Αντίθετα με τη χρησιμοποίηση του cin, κάθε φορά που εκτελείται το πρόγραμμα, ανάλογα με τις τιμές που πληκτρολογούνται δημιουργούνται και διαφορετικά αποτελέσματα.

Το πλεονέκτημα του cin έναντι των προτάσεων ανάθεσης είναι το πρόγραμμα γίνεται μεγαλύτερης εμβέλειας, σε ορισμένες δε περιπτώσεις επιβάλλεται. Υπενθυμίζεται στα προβλήματα αγνώστου πλήθους στοιχείων όταν χρειάζεται να εισάγουμε τον ΤΕΡΜΑΤ(ΙΣΤΗ), προτιμάται η χρησιμοποίηση του cin και όχι πρότασης ανάθεσης, διότι προφυλλάσσεται ο προγραμματιστής από το μοιραίο λάθος της ανάθεσης τιμής στον ΤΕΡΜΑΤ που περιέχεται στην ακολουθία δεδομένων που εισάγει. Με τον cin ελέγχεται καλλίτερα ο ΤΕΡΜΑΤ αλλά και το πρόγραμμα καθίσταται μεγαλύτερης εμβέλειας, ως προς τα δεδομένα που μπορεί να χειριστεί.

Ο τελεστής cin απαιτεί τη ακριβή πληκτρολόγηση των δεδομένων εισόδου από τον χρήστη, γιαυτό πριν από κάθε cin, τυπώστε μια προτροπή (μήνυμα προς τον χρήστη) που θα εξηγεί αυτό που πρέπει ο χρήστης να πληκτρολογήσει.

4.7.4 Τελεστές Ελέγχου Εισαγωγής Δεδομένων

Ο τελεστής cin χρησιμοποιεί τις ίδιες συναρτήσεις διαχειριστές,**setw(** <μηκος_πεδιου>**)** και **setprecision(** <πληθος_δεκαδικων_ψηφιων>**)** με τον τελεστή cout.

Οδηγίες για Ελεγχόμενες Εισόδους :

1. Για να εισαχθεί ένας μεγονωμένος χαρακτήρας από το πληκτρολόγιο, χρησιμοποιούμε

τη συνάρτηση: **cin.get( ).**

2. Για να εισαχθεί μια αλυσίδα χαρακτήρων με ενδιάμεσα κενά, χωρίς αυτά να θεωρηθούν σαν όριο του πεδίου εισόδου, εμποδίζοντας τα μέρη της αλυσίδας μετά το πρώτο κενό να διαβαστούν, χρησιμοποιούμε τη συνάρτηση

cin.getline(<προσδιοριστης_αλυσιδας_χαρακτηρων>, n **)**

όπου n είναι το μέγιστο πλήθος στοιχείων που θα διαβαστούν.

4.7.5 Ασκήσεις (Ερωτήσεις για Επανάληψη και Εμπέδωση)

1. Ποιά είναι η διαφορά μεταξύ των **cout** και **cin** ;

2. Γιατί το μήνυμα προς τον χρήστη είναι απαραίτητο πριν από τη χρησιμοποίηση του **cin** για είσοδο δεδομένων;

3. Πόσες τιμές εισάγετε με την ακόλουθη πρόταση;

cin >> i >> j >> k >> l >>;

4. Υπάρχει καμμία διαφορά μεταξύ της ανάθεσης τιμών σε μεταβλητές με προτάσεις ανάθεσης και με τον τελεστή **cin** ;

5. Ποιά έξοδος παράγεται από την πρόταση:

cout << " Το ζευγάρι των εισαγωγικών \ " \\\ " είναι ειδικός χαρακτήρας " ;

6. Ποιό είναι το αποτέλεσμα του τελεστή cout που έπεται :

cout << setw(8) << setprecision(3) << 123.456789 ;

4.8 Συναρτήσεις

Η C++ σχεδιάστηκε με τρόπο ώστε ο προγραμματιστής να μπορεί να γράψει προγράμματα που αποτελούνται από αρκετές ενότητες αντί για προγράμματα που έχουν ένα συμπαγές σώμα. Διαμερίζοντας το πρόγραμμα σε αρκετά μικρότερα υποπρογράμματα, μπορεί ο προγραμματιστής να απομονώνει προβλήματα, να γράφει προγράμματα πιό γρήγορα και να παράγει ευκολοσυντηρούμενα προγράμματα

Οι συναρτήσεις λοιπόν είναι οι ενότητες του προγράμματος που το αποτελούν και που εκτελούνται και ελέγχονται από την συνάρτηση **main( )**. Οι συναρτήσεις της C++

γενικά ακολουθούν τους παρακάτω κανόνες:

1. Κάθε συνάρτηση πρέπει να έχει ένα όνομα (προσδιοριστή).

2. Οι προσδιοριστές των συναρτήσεων ακολουθούν τους ίδιους κανόνες που ακολουθούν οι προσδιοριστές των μεταβλητών. Δηλαδή, μπορούν να περιέχουν μέχρι 32 χαρακτήρες, πρέπει να αρχίζουν πάντοτε από γράμμα και αποτελούνται από γράμματα και αριθμούς και τον χαρακτήρα υπογράμμισης "_" .

3. Όλοι οι προσδιοριστές των συναρτήσεων συνοδεύονται από ένα ζευγάρι παρενθέσεων. Οι παρενθέσεις μπορεί να περιέχουν παραμέτρους ή να είναι κενές.

Το κύριο σώμα της συνάρτησης αρχίζει μετά τη δεξιά παρένθεση και περικλείεται από ένα ζευγάρι αγγύλων.

Παρατηρήσεις:

1. Οι προσδιοριστές των συναρτήσεων πρέπει να ανταποκρίνονται στη διαδικασία που αυτές επιτελούν, π.χ. DIATAXI( ), κλπ.

2. **Προσοχή!** μη χρησιμοποιείσετε την παύλα "-" αντί για τον χαρακτήρα της υπογράμμισης "_" σε προσδιοριστή συνάρτησης, διότι η C++ παράγει παραπλανητικά μηνύματα λάθους.

4.8.1 Δήλωση και Κλήση των Συναρτήσεων

Μια συνάρτηση δηλώνεται ως εξής:

```
[<τυπος>] [void]<προσδιοριστής_συναρτησης> (<παραμετροι>)[61]
        {
          . . .
        }
```

δηλαδή,

1. <τυπος> : το όνομα ένος τύπου δεδομένων ή τη λέξη **void**. Στην πρώτη περίπτωση το όνομα του τύπου δεδομένων είναι ο τύπος της τιμής που η συνάρτηση υπολογίζει. Στη δεύτερη περίπτωση η συνάρτηση δε δίνει καμία τιμή.
2. <προσδιοριστής_συναρτησης> (βλπ προηγούμενη παράγραφο).

[61] Η πρώτη γραμμή αποτελεί και την επικεφαλίδα της συνάρτησης

3. (<παραμετροι>) : είναι μεταβλητές μέσω των οποίων διαβιβάζονται τιμές προς τη συνάρτηση.Αν δεν υπάρχουν παράμετροι μέσα στις παρενθέσεις δεν γράφουμε τίποτα ή γράφουμε τη λέξη void,

4. { } μέσα στις αγγύλες βρίσκεται το σώμα της συνάρτησης, δηλαδή οι δηλώσεις και οι προτάσεις της (πρβλ τη συνάρτηση **main( )**).

Για παράδειγμα γράφουμε τη δήλωση της συνάρτησης που υπολογίζειτον μέγιστο μεταξύ δυο ακεραίων αριθμών:

```
. . .
# include <iostream.h>
void meg (float x, float y)
{
 if ( x > y)
   {cout << x << "\n"; }
 else
   {cout << y << "\n"; }
}
void main ( )
{
 int termat;
 float a,b;
 cin >> termat;
 cin >> a >> b;
 while (ai = termat)
  {
    cout << "μεγαλύτερος είναι ο ";
   meg(a, b);
     cin >> a >> b;
  }
 }
```

Κλήση, δηλαδή, εκτέλεση της συνάρτησης αυτής meg, γίνεται στο κυρίως πρόγραμμα με απλή αναφορά στο όνομά της.

4.8.2 Διαβίβαση Τιμών σε Συναρτήσεις

Σε μια συνάρτηση διαβιβάζουμε τιμές μέσω των παραμέτρων της. Οι παράμετροι όπως έχουμε πεί είναι μεταβλητές που γράφονται μέσα σε παρενθέσεις μετά το όνομα της συνάρτησης (βλπ π.χ. §4.8.2).

Οι παράμετροι της συνάρτησης λέγονται τυπικές παράμετροι ή ψευδοπαράμετροι,

(formal parameters) ενώ οι παράμετροι της κλήσης της συνάρτησης στο κυρίως πρόγραμμα λέγονται πραγματικές παράμετροι (actual parameters).

Υπάρχει πλήρης αντιστοιχία μεταξύ των πραγματικών παραμέτρων και των ψευδοπαραμέτρων. Η αντιστοιχία αφορά το πλήθος, τη σειρά και τον τύπο τους.

Κάθε τυπική παράμετρος είναι μια μεταβλητή. Ενώ κάθε πραγματική παράμετρος μπορεί να είνα μια σταθερά ή μια μεταβλητή ή μια παράσταση.

4.8.3 Επιστροφή Τιμών από Συναρτήσεις

Για να επιστραφή από μια συνάρτηση το αποτέλεσμα στη συνάρτηση που την κάλεσε, πρέπει στην καλούμενη συνάρτηση να γραφεί και μάλιστα η τελευταία του σώματος η εντολή **return( )**. Η return( ) δέχεται σαν παραμέτρους μια σταθερά ή μια ματαβλητή ή μια παράσταση, η τιμή των οποίων θα επιστραφεί στο καλόν πρόγραμμα.

Για να λειτουργήσει η επιστροφή της τιμής από τη συνάρτηση σωστά θα πρέπει:
1. Στη συνάρτηση πρέπει να δοθεί ο τύπος του αποτελέσματος που επιστρέφει. Είναι ο τύπος που όπως ήδη είπαμε (βλπ §4.8.1, <τυπος>).
2. Θα πρέπει επίσης, η κλήση της συνάρτησης να εμφανίζεται σε επιτρεπτό μέρος στο κυρίως πρόγραμμα. Για παράδειγμα, δεξιά του = ή σε cout ή μέσα σε συνθήκη ή στη θέση κάποιας πραγματικής παραμέτρου κλπ. Δεν επιτρέπται μια κλήση συνάρτησης να βρίσκεται αριστερά του ίσον (=) ή ακόμα χειρότερα σε μια cin.

```
. . .
# include <iostream.h>
float megatos (float a, float b)
{
  float meg;
  if ( a > b)
     { meg = a; }
  else
     { meg = b; }
}
void main ( )
{
  float x, y;
for(;;)
```

```
{
 cout<< "δώσε δύο αριθμούς";
 cin >> x >> y;
 if ( x==y ) break;
 cout << "μεγαλύτερος είναι ο";
 meg = megisto(a, b);
 cout << meg <<"\n";
 }
}
```

4.8.4 Εμβέλεια των Μεταβλητών

Μέχρι τώρα βρίσκουμε τις δηλώσεις των μεταβλητών στην αρχή κάθε συνάρτησης, αφορούν δε αποκλειστικά τις μεταβλητές της συνάρτησης, οι οποίες για τον λόγο αυτόν λέγονται **εσωτερικές ή τοπικές μεταβλητές** (local variables). Αυτή η ιδιότητα κατά κάποιο τρόπο των τοπικών μεταβλητών, των μεταβλητών με τοπική (εσωτερική) εμβέλεια χρήσης του ονόματος και της τιμής τους, δίνει τη δυνατότητα σε διαφορετικές συναρτήσεις να χρησιμοποιείται το ίδιο όνομα μεταβλητών, χωρίς η ανάθεση τιμής σε μια μεταβλητή μιας συνάρτησης, να επηρεάζει την τιμή της μεταβλητής με το ίδιο όνομα της άλλης συνάρτησης. Με την έννοια αυτή μπορούμε να υποστηρίξουμε ότι ακόμα και οι παράμετροι είναι τοπικές μεταβλητές.

Παράδειγμα 4.8.4.1:

```
. . .
#include <iostream.h>
void topiki_metavliti (int x)
{  long y;
   cout << " αρχική τιμή topiki_metavliti: x = " << x << " y =" << y <<"\n";
  x = 1;
  y = 0;
   cout << " τελική τιμή topiki_metavliti: x = " << x << " y =" << y <<"\n";
}
void main
{
  int a;
  long y;
 a = 2;
 y = 3;
  cout << " αρχική τιμή main: a = " << a << " y =" << y <<"\n";
```

```
  topiki_metavliti ( a );
   cout << " τελική τιμή main: a = " << a << " y =" << y <<"\n";
}
```

εκτύπωση στην οθόνη:

αρχική τιμή main: a = 2 y = 3

αρχική τιμή topiki_metavliti: x = 1 y = -655359

τελική τιμή topiki_metavliti: x = 1 y = 0

τελική τιμή main: a = 2 y = 3

Σε ορισμένες περιπτώσεις χρειαζόμαστε μεταβλητές οι οποίες είναι προσιτές από μερικές ή από όλες τις συναρτήσεις. Η C++ μας δίνει τη δυνατότητα τέτοιων μεταβλητών και μάλιστα με πολύ απλό τρόπο. Οι μεταβλητές αυτές λέγονται καθολικές (global). Οι δηλώσεις των καθολικών ματαβλητών γίνονται στην αρχή του προγράμματος, πριν από την επικεφαλίδα της πρώτης συνάρτησης. Μια μεταβλητή που δηλώνεται σε αυτό το σημείο ανήκει σε όλες τις συναρτήσεις και οι τιμές της μπορούν να αξιοποιηθούν από όλες τις συναρτήσεις.

Παράδειγμα 4.8.4.2:

```
. . .
#include <iostream.h>
long y;
void topiki_metavliti (int x)
{
   cout << " αρχική τιμή topiki_metavliti: x = " << x << " y =" << y <<"\n";
  x = 1;
  y = 0;
  cout << " τελική τιμή topiki_metavliti: x = " << x << " y =" << y <<"\n";
}
void main( )
{
  int a;
  long y;
  a = 2;
  y = 3;
   cout << " αρχική τιμή main: a = " << a << " y =" << y <<"\n";
  topiki_metavliti ( a );
   cout << " τελική τιμή main: a = " << a << " y =" << y <<"\n";
}
```

εκτύπωση στην οθόνη:
αρχική τιμή main: a = 2 y = 3
αρχική τιμή topiki_metavliti: x = 2 y = 3
τελική τιμή topiki_metavliti: x = 1 y = 0
τελική τιμή main: a = 2 y = 0

Σημείωση: Αν δηλώσουμε μια μεταβλητή έξω από την περιοχή κάποιας συνάρτησης, όχι δηλαδή στην αρχή του προγράμματος, αλλά κάπου ενδιάμεσα, τότε η μεταβλητή λειτουργεί σαν καθολική μόνο από τις συναρτήσεις που βρίσκονται κάτω από το σημείο δήλωσης της συνάρτησης.

4.8.5 Αναφορές στις Παραμέτρους

Η C++ παρέχει την ευκολία του καθορισμού "αναφορών" μεταβλητών. Μια **"αναφορά"** (reference) **μεταβλητής** είναι η δυνατότητα ορισμού δευτέρου ονόματος στη μεταβλητή. Οι αναφορές αυτές είναι ιδιαίτερα χρήσιμες συνδυαζόμενες με τις συναρτήσεις, πέραν του ότι μπορούμε να διαχειριζόμαστε κάποια ποσότητα χρησιμοποιώντας δύο διαφορετικά ονόματα. Με τις αναφορές πετυχαίνουμε να δίνει μια συνάρτηση τιμές στις πραγματικές παραμέτρους, κάτι που κανονικά δεν γίνεται.

Οι αναφορές μεταβλητών ορίζονται υποχρεωτικά μετά τις δηλώσεις των μεταβλητών. Μια αναφορά ορίζεται ως εξής:

<τυπος> & <αναφορα_μεταβλητης> = <προσδιοριστης_μεταβλητης>

Παράδειγμα 4.8.5.1:

```
. . .
#include <iostream.h>
void main( )
{
  int apondes;
  int &astheneis = apondes;
   apondes = 9;
   cout << astheneis << "\n";      // θα εκτυπώσει astheneis = 9
   astheneis = 12;
```

```
    cout << apondes << "\n";        // θα εκτυπώσει apondes = 12
}
```

Παράδειγμα 4.8.5.2: Έστω μια συνάρτηση που υπολογίζει δύο τιμές, το άθροισμα και το γινόμενο δύο αριθμών. Τα υπολογιζόμενα αποτελέσματα είναι δύο, άρα δεν διαβιβάζονται στην **main( )** με την **return( )**. Για τη διαβίβαση θα χρησιμοποιήσουμε παραμέτρους.

Σημείωση: Για να μπορέσει μια συνάρτηση να ανθέσει τιμές σε παραμέτρους θα πρέπει:
1. Να χρησιμοποιούμε αναφορές σαν πραγματικές παραμέτρους.
2. Να βάζουμε τον ειδικό χαρακτήρα "&" πριν από τις τυπικές παραμέτρους.

Έτσι έχουμε:

```
. . .
#include <iostream.h>
void sumprod ( float a, float b, float &sum, float &prod)
{
 sum = a + b;
 prod = a* b;
}
void main( )
{
  float x, y;
  float &xx = x, &yy = y;
  sumprod(4, 8, xx, yy);
  cout << " το άθροισμα είναι :" << x << "  και το γινόμενο είναι : " << y << "\n";
}
```

εκτύπωση στην οθόνη: το άθροισμα είναι : 12 και το γινόμενο είναι : 32

4.8.6 Πρωτότυπα Συναρτήσεων

Οι συναρτήσεις γράφονται πριν από το κύριο πρόγραμμα, δηλαδή, τη συνάρτηση main( ), διότι όταν ο μεταγλωττιστής βρεί κλήση μιας συνάρτησης, για να αποφασίσει τον τρόπο διαχείρισής της, πρέπει πρώτα να έχει συναντήσει και αναγνωρίσει τη δήλωσή της.

Η C++ προβλέπει σε περίπτωση που θα θελήσουμε να γράψουμε τις συναρτήσεις μετά την main( ), να μπορούμε γράφοντας πριν από την main( ) μόνο τα πρωτότυπά τους

(function prototype), δηλαδή την επικεφαλίδα τους χωρίς τα ονόματα των τυπικών παραμέτρων αλλά απαραιτήτως να υπάρχει ο τύπος κάθε παραμέτρου (χωρίς το όνομα) μέσα στην παρένθεση.

4.8.7 Ασκήσεις (Ερωτήσεις για Επανάληψη και Εμπέδωση)

1. Είναι αληθές ή ψευδές ότι, μια συνάρτηση πρέπει πάντα να περιλαμβάνει μια πρόταση **return** σαν τελευταία της εντολή;

2. Ποιό είναι το όνομα της πρώτης συνάρτησης που εκτελείται σε ένα πρόγραμμα της C++;

3. Τι είναι καλλίτερο, μια εκτενής συνάρτηση ή περισσότερες μικρότερες συναρτήσεις και γιατί;

4. Πως διαφέρουν τα ονόματα των συναρτήσεων από τα ονόματα των μεταβλητών;

5. Πως μπορείτε να χρησιμοποιείτε τα σχόλια για να ξεχωρίζετε οπτικά τις συναρτήσεις;

6. Ποιό είναι το λάθος στο παρακάτω τμήμα προγράμματος

```
ypologismos( )
{
  cout
  tetragonarithm( )
  {
    cout << " Το τετράγωνο του 25 είναι : " (25 * 25);
    return(0);
  }
  cout << " Είναι μεγάλος αριθμός ! \n";
  return(0);
}
```

7. Το παρακάτω είναι όνομα μεταβλητής, κλήση συνάρτησης, ορισμός συνάρτησης ή έκφραση;

```
anihneftis_onomat( );
```

8. Είναι αληθές ή ψευδές ότι, η ακόλουθη γραμμή σε ένα πρόγραμμα της C++ είναι μια κλήση συνάρτησης;

```
cout << " Είναι ευχάριστος ο προγραμματισμός με C++ \n";
```

9. Είναι αληθές ή ψευδές ότι, μια συνάρτηση θα πρέπει να περιέχει πάντα μια πρόταση

return σαν την τελευταία εντολή, ακόμα και όταν δεν απαιτείται αυτή.

10. Είναι αληθές ή ψευδές ότι, μια συνάρτηση στην οποία έχουν διαβιβαστεί μεταβλητές από μια άλλη συνάρτηση δεν μπορεί να έχει και τις δικές της τοπικές μεταβλητές;

11. Τι θα πρέπει να εμφανίζεται μέσα στις παρενθέσεις μιας συνάρτησης;

12. Ποιές παράμετροι λέγονται πραγματικές και ποιές τυπικές ή ψευδοπαράμετροι;

13. Ποιές μεταβλητές λέγονται εσωτερικές ή τοπικές και ποιές καθολικές;

14. Τι γνωρίζετε για την αναφορά μεταβλητής;

15. Τι είναι τα πρωτότυπα συναρτήσεων;

4.9 Ασκήσεις

ΟΜΑΔΑ Α

1.Να γραφεί πρόγραμμα που τυπώνει τις οκτώ πρώτες δυνάμεις του 2, δηλαδή: $2^1, 2^2, 2^3, \ldots, 2^8$. Γράψτε σχόλια και το ονοματεπώνυμό σας στην αρχή του προγράμματος. Εκτυπώστε αλυσίδες χαρακτήρων που περιγράφουν την κάθε εκτυπούμενη απάντηση, π.χ. " το 2 υψούμενο στην 1η δύναμη είναι ίσο με 2" κ.λ.π. Το πρόγραμμα να επεκταθεί για περισσότερους αριθμούς αγνώστου πλήθους.

2. Να γραφεί πρόγραμμα που αποθηκεύει τα ύψη και τις ηλικίες τριών ανθρώπων σε μεταβλητές. Υπολογίζει τον μέσο όρο τους και τυπώνει κατάλογο με τίτλους, ύψη και ηλικίες. Επίσης στο κάτω μέρος του καταλόγου τυπώνει τον μέσον όρο των υψών και τον μέσον όρο των ηλικιών. Το πρόγραμμα να επεκταθεί για περισσότερους των τριών ανθρώπων, (α) να γνωρίζουμε το πλήθος τους, (β) να είναι αγνώστου πλήθους.

3. Να γραφεί πρόγραμμα που ζητάει έναν κατάλογο των θερμοκρασιών της τελευταίας εβδομάδας και τυπώνει το μήνυμα "ΠΑΓΩΝΙΑ" κάθε φορά που η θερμοκρασία πέφτει κάτω από το μηδέν. Το πρόγραμμα να επεκταθεί για περισσότερες εβδομάδες, (α) να γνωρίζουμε το πλήθος τους, (β) να είναι αγνώστου πλήθους.

4. Να γραφεί πρόγραμμα που ζητάει έναν αριθμό και τυπώνει την τετραγωνική και την κυβική ρίζα του, εάν ο αριθμός είναι μεγαλύτερος του 1. Σε οποιαδήποτε άλλη περίπτωση

το πρόγραμμα δεν τυπώνει τίποτα. Το πρόγραμμα να επεκταθεί για περισσότερους αριθμούς, (α) να γνωρίζουμε το πλήθος τους, (β) να είναι αγνώστου πλήθους.

5 Να γραφεί πρόγραμμα το οποίο τροφοδοτείται από έναν μικτό μισθό υπαλλήλου, υπολογίζει και τυπώνει του φόρους που του αναλογούν. Οι φόροι είναι 10%, εάν ο υπάλληλος κερδίζει λιγότερα από 3,000,000 δρχ, 15% εάν κερδίζει πάνω από 3,000,000 αλλά λιγότερο από 5,000,000 και 20% εάν υπερβαίνει τα 5,000,000. Το πρόγραμμα να επεκταθεί για περισσότερους υπαλλήλους, (α) να γνωρίζουμε το πλήθος τους, (β) να είναι αγνώστου πλήθους.

6. Να γραφεί πρόγραμμα το οποίο ζητά από τους χρήστες (α) γνωστού πλήθους, (β) αγνώστου πλήθους την ηλικία τους. Εάν αυτοί είναι πάνω από 21, τυπώνεται το μήνυμα: "Δεν είστε ανήλικος". Εάν είναι κάτω των 21, τυπώνει το μήνυμα: "είστε ανήλικος".

7. Να γραφεί πρόγραμμα το οποίο τυπώνει τους 256 χαρακτήρες ASCII. Το πρόγραμμα να διαμορφωθεί κατάλληλα ώστε η εκτύπωση κάθε 20 χαρακτήρων να γίνεται σε νέα οθόνη και η συνέχιση της διαδικασίας για τους επόμενους 20 χαρακτήρες να εξαρτάται από την κατάλληλη απάντηση στην ερώτηση: " να συνεχίσω τη διαδικασία ή να τη διακόψω; "

8. Να γραφεί πρόγραμμα το οποίο διαβάζει μια λέξη και ένα γράμμα, βρίσκει πόσες φορές το γράμμα συναντάται μέσα στη λέξη και τυπώνει τη λέξη που δόθηκε, το γράμμα που δόθηκε και πόσες φορές αυτό βρίσκεται ή όχι στη λέξη.
Η διαδικασία να επαναλαμβάνεται, για διαφορετικά δεδομένα, μέχρι να δοθεί κάποια απάντηση σε ερώτηση για να σταματήσει.

9. Να γραφεί πρόγραμμα το οποίο μετατρέπει κεφαλαία γράμματα σε πεζά και τυπώνει τα κεφαλαίοα και το αντίστοιχα πεζά, μετά το τέλος της διαδικασίας. Εάν κάποιο γράμμα που δόθηκε είναι ήδη πεζό δεν αλλάζει τίποτα και τυπώνει ανάλογο μήνυμα. Το πλήθος των γραμμάτων είναι (α) γνωστό, (β) άγνωστο και (γ) όλα τα γράμματα του λατινικού αλφαβήτου διατεταγμένα: (γ_1) από το A στο Z και (γ_2) από το Z στο A.

10. Να γραφεί πρόγραμμα που βρίσκει όλους τους περιττούς αριθμούς από το 1 έως το 100 και τυπώνει σε κάθε γραμή 2 περιττούς αριθμούς οι οποίοι ισαπέχουν των άκρων. Δηλαδή στην πρώτη γραμμή τυπώνει τον πρώτο περιττό και τον τελευταίο, στη δεύτερη

γραμμή τυπώνει τον δεύτερο περιττό και τον πρότελευταιο κ.ο.κ. ενώ στην τελευταία γραμμή τυπώνει τον τελευταίο περιττό και τον πρώτο.

11. Να γραφεί πρόγραμμα που τυπώνει το μήνυμα "ο προγραμματισμός είναι συχάριστος με την C++" στην οθόνη για δέκα δευτερόλεπτα. (Υπόδειξη: ίσως πρέπει ο βρόγχος αναμονής να προσαρμοστεί ανάλογα).

12. Να προσαρμοστεί το πρόγραμμα της προηγούμενης άσκησης ώστε να αναβοσβήνει το μήνυμα για 10 δευτερόλεπτα. (Υπόδειξη: Ίσως πρέπει να χρησιμοποιήσετε αρκετούς βρόγχους αναμονής).

13. Να γραφεί πρόγραμμα που υπολογίζει τον μέσο όρο βαθμολογίας μιας τάξης 30 μαθητών. Αγνοήστε κάθε βαθμό μικρότερο του 0 και συνεχίστε μέχρι να εισαχθούν και οι 30 βαθμοί των μαθητών ή μέχρι να πληκτρολογηθεί ο -99 για να τερματιστεί νωρίτερα το πρόγραμμα.

14. Να γραφεί πρόγραμμα που τυπώνει τους αριθμούς από 1 μέχρι 14 σε μια στήλη. Επίσης τυπώνει, στα δεξιά των αρτίων αριθμών το τετράγωνο τους και στα δεξιά των περιττών τον κύβο τους.

15. Να γραφεί πρόγραμμα, χρησιμοποιώντας την πρόταση switch, που ζητάει από τους χρήστες την ηλικία τους και ανάλογα τυπώνει κάποιο μήνυμα. Συγκεκριμένα:
(α) Εάν είναι κάτω των 18 τότε τυπώνει "είσαι ακόμα πολύ νέος",
(β) Εάν είναι 18 ετών και άνω τότε τυπώνει "μπορείς να ψηφίσεις",
(γ) Εάν είναι 21μέχρι 65 ετών τότε τυπώνει "μπορείς να υιοθετήσεις",
(δ) Εάν είναι 65 και άνω τότε τυπώνει "δυστυχώς δεν μπορείτε να υιοθετήσετε, μπορείτε όμως αν θέλετε να λάβετε μέρος σε δραστηριότητες φιλανθρωπικού χαρακτήρα".

16. Να γραφεί πρόγραμμα με επιλογές για κάποια τοπική καλωδιακή τηλεόραση. Η χρέωση θα γίνεται ως εξής:
(α) Εάν βρίσκεστε 280 χιλιόμετρα εκτός των ορίων της πόλης, τότε πληρώνεται 25,000 δρχ τον μήνα,
(β) Εάν βρίσκεστε 281μέχρι 380 χιλιόμετρα εκτός των ορίων της πόλης, τότε πληρώνεται 35,000 δρχ τον μήνα,

(γ) Εάν βρίσκεστε 381μέχρι 500 χιλιόμετρα εκτός των ορίων της πόλης, τότε πληρώνεται 45,000 δρχ τον μήνα,
(δ) Εάν βρίσκεστε πάνω από 500 χιλιόμετρα εκτός των ορίων της πόλης, τότε δεν δέχεστε τις υπηρεσίες της εταιρίας.
Ενημερώστε τους χρήστες με μία επιλογή για την απόσταση του σπιτιού τους από τα όρια της πόλης και τις αντίστοιχες υποχρεώσεις τους.

17. Να γραφεί πρόγραμμα που υπολογίζει τα τέλη ενός πολυορόφου σταθμού αυτοκινήτων. Ζητήστε από τον οδηγό εάν το αυτοκίνητό του είναι επιβατηγό ή φορτηγό. Ο οδηγός χρεώνεται 300δρχ για την πρώτη ώρα στάθμευσης, 450δρχ για τη δεύτερη ώρα και 600δρχ για περισσότερες από 2 ώρες. Εάν το αυτοκίνητο είναι φορτηγό, προστίθενται 250δρχ στην συνολική πληρωμή. (Υπόδειξη: Χρησιμοποιήστε μια πρόταση switch και μια πρόταση if).

18. Να τροποποιηθεί το προηγούμενο πρόγραμμα ώστε η χρέωση να εξαρτάται από την ώρα της ημέρας στην οποία γίνεται η στάθμευση του οχήματος. Έτσι:
(α) Εάν το όχημα σταθμεύσει πριν από τις 8 π.μ., τότε χρεώνεται με τον τρόπο της προηγούμενης άσκησης,
(β) Εάν το όχημα σταθμεύσει μετά τις 8 π.μ. και πριν από τις 5 μ.μ., τότε χρεώνεται με 150δρχ επιπλέον.
(γ) Εάν το όχημα σταθμεύσει μετά τις 5 μ.μ., τότε αφαιρούνται 150δρχ από την τιμή που υπολογίστηκε με τον τρόπο της προηγούμενης άσκησης.
Ενημερώστε τους χρήστες με μία επιλογή για την αρχική ώρα στάθμευσης, όπως π.χ. πριν τις 8π.μ., πριν τις 5μ.μ., μετά τις 5μ.μ..

19. Να γραφεί πρόγραμμα με το οποίο θα διαβάζονται και θα αποθηκεύονται οι ηλικίες των φίλων σας, στην συνέχεια θα διατάσσονται κατά αύξουσα τάξη και θα τυπώνονται.

20. Να γραφεί πρόγραμμα με το οποίο θα διαβάζονται και θα αποθηκεύονται οι ηλικίες των φίλων σας, στην συνέχεια θα διατάσσονται κατά φθίνουσα τάξη και θα τυπώνονται.

21. Να γραφεί πρόγραμμα με το οποίο θα διαβάζονται και θα αποθηκεύονται οι ηλικίες των φίλων σας, στην συνέχεια θα επιλέγεται η διατάξη των ηλικιών κατά αύξουσα τάξη ή κατά φθίνουσα τάξη και θα τυπώνονται.

22. Να γραφεί πρόγραμμα που θα διαβάζει και θα αποθηκεύει τους αριθμούς από 1 έως 100 σε μια ακεραία παράταξη 1,000 στοιχείων. (Υπόδειξη: Οι δείκτες πρέπει να αρχίζουν από το 0 και να τελειώνουν στο 99.)

23. Μερικές φορές παρατάξεις που έχουν ίδιες διαστάσεις χρησιμοποιούνται σε προγράμματα για να καταχωρηθούν σε καταλόγους τιμές που συσχετίζονται. Για παράδειγμα, σε ένα πρόγραμμα που χειρίζεται τις πωλήσεις διαφόρων προιόντων μιας επιχείρησης σε πελάτες της, αντιστοιχούμε σε κάθε προιόν μια μονοδιάστατη παράταξη, ενώ το πλήθος των πελατών ταυτίζεται με το μήκος κάθε παράταξης.

Να γραφεί πρόγραμμα με το οποίο μια μικρή επιχείρηση θα μπορεί να διατηρεί ίχνη για τους πελάτες της, το πολύ 100. Σε κάθε πελάτη αντιστοιχεί ένας αύξοντας αριθμός (η αρχή γίνεται από το μηδέν). Έτσι, κάθε φορά που κάποιος πελάτης αγοράζει ένα προιόν, η επιχείρηση διαθέτει 5 προιόντα, η πώληση καταγράφεται στην παράταξη που αντιστοιχεί στο προιόν, με δείκτη τον αύξοντα αριθμό που αντιστοιχεί στον πελάτη. Στο τέλος κάθε μέρας, τυπώνεται μια αναφορά η οποία αποτελείται από τους αύξοντες αριθμούς των πελατών, τις αντίστοιχες πωλήσεις, δηλαδή τις τιμές που είναι καταχωρημένες στα αντίστοιχα στοιχεία των αντίστοιχων παρατάξεων, το συνολικό πλήθος προιόντων που πουλήθηκαν και τον μέσο όρο πωλήσεων ανά πελάτη.

24. Στα δεδομένα της παραπάνω άσκησης εισάγεται μια ακόμα παράταξη στην οποία αποθηκεύονται τα επώνυμα των πελατών. Να γραφεί πρόγραμμα στο οποίο καταχωρούνται τα δεδομένα. Το πρόγραμμα διατάσσει τα δεδομένα ως προς το επώνυμο, με οποιαδήποτε μέθοδο. Στην συνέχεια ζητάει το ένα επώνυμο, εννοείται κάποιου πελάτη, αφού βρεθεί καταχωρημένο, το τυπώνει, όπως επίσης τυπώνει το πλήθος των προιόντων που αγόρασε ανά είδος. Εάν το επώνυμο που δόθηκε δεν συμπεριλαμβάνεται στα επώνυμα των πελατών τότε τυπώνεται κατάλληλο μήνυμα. (Υπόδειξη: για τη διερεύνηση να χρησιμοποιηθεί η διαδοχική διερεύνηση)

25. Να επαναληφθεί η προηγούμενη άσκηση χρησιμοποιώντας τη δυαδική διερεύνηση.

26. Να ξανασχεδιαστεί η άσκηση 24 ώστε ο χρήστης να μπορεί να διαλέξει τη μέθοδο διάταξης όπως και τη μέθοδο διερεύνησης που θέλει να χρησιμοποιήσει. (Υπόδειξη: οι μέθοδοι διάταξης μεταξύ των οποίων μπορεί να διαλέξει ο χρήστης είναι η Ανταλλαγή των

Μεγίστων Τιμών" και η "Προώθηση των Μεγάλων Τιμών". Ενώ οι μέθοδοι διερεύνησης μεταξύ των οποίων μπορεί να διαλέξει ο χρήστης είναι η "Διαδοχική Διερεύνηση" και η "Διαδική Διερεύνηση".)

27. Να γραφεί πρόγραμμα που αποθηκεύει και τυπώνει τους αριθμούς από 1 μέχρι 21 σε ένα πίνακα 3 x 7. (Υπόδειξη η C++ αρχίζει τους δείκτες της από το 0.)

28. Να γραφεί πρόγραμμα που αποθηκεύει τα δεδομένα πωλήσεων, τριών ετών, πέντε πωλητών. Να χρησιμοποιηθούν προτάσεις ανάθεσης, για να δοθούν τα στοιχεία της παράταξης. Στη συνέχεια το πρόγραμμα τυπώνει μια τιμή ανά γραμμή.

29. Αντί να χρησιμοποιηθούν προτάσεις ανάθεσης, να χρησιμοποιηθεί η cin για εισαγωγή των δεδομένων της προηγούμενης άσκησης.

30. Να γραφεί πρόγραμμα που αποθηκεύει τους βαθμούς πέντε τάξεων, η κάθεμία με τριάντα μαθητές. Τα δεδομένα εισάγονται με την cin. Το πρόγραμμα τυπώνει τα αποτελέσματα με τη φυσική μορφοποίηση των γραμμών και στηλών.

31. Να γραφεί πρόγραμμα που ζητάει στην main( ) την ηλικία ενός σκύλου. Να γραφεί μια δεύτερη συνάρτηση, η οποία θα ονομαστεί anthropochronia( ) και θα υπολογίζει την ηλικία του σκύλου σε ανθρώπινα χρόνια. Υπενθυμίζεται ότι, επτά ανθρώπινα χρόνια ισοδυναμούν με ένα χρόνο ζωής του σκύλου.

32. Να γραφεί μια συνάρτηση main( ) και μια συνάρτηση που καλεί η main( ). Στη συνάρτηση main( ) ζητήται από τους χρήστες το ετήσιο εισόδημά τους. Το εισόδημα διαβιβάζεται στη δεύτερη συνάρτηση και εκτυπώνεται ένα μήνυμα "συγχαρητήριο" εάν ο χρήστης κερδίζει περισσότερα από 15,000,000 δρχ τον μήνα ή ένα μήνυμα "ενθάρρυνσης" εάν κερδίζει λιγότερα!!

33. Να γραφεί πρόγραμμα που αποτελείται από τρείς συναρτήσεις, την main( ), την synartisi1( ) και την synartisi2( ). Στην main( ) να δηλώνεται μια παράταξη 10 στοιχείων. Με την synartisi1( ) να γίνεται η εισαγωγή των δεδομένων, π.χ. από το A μέχρι το J. Με την synartisi2( ) να γίνεται η έξοδος των αποτελεσμάτων, η εκτύπωση δηλαδή των στοιχείων της παράταξης.

34. Να γραφεί πρόγραμμα στο οποίο η συνάρτηση main( ) διαβιβάζει έναν ακέραιο αριθμό σε μια συνάρτηση με όνομα typose_asteriskous( ). Η συνάρτηση typose_asteriskous( ) τυπώνει αστερίσκους σε μια γραμμή στην οθόνη. Εάν στη συνάρτηση typose_asteriskous( ) έχει διαβιβαστεί ακέραιος μεγαλύτερος του 80, τότε εμφανίζεται ένα μήνυμα λάθους, επειδή οι περισσότερες οθόνες δεν μπορούν να τυπώσουν περισσότερου από 80 χαρακτήρες στη ίδια γραμμή. Όταν η εκτέλεση τελειώσει, ο έλεγχος επιστρέφεται στην main( ) και μετά στο λειτουργικό σύστημα.

35. Να γραφεί πρόγραμμα που περιέχει δύο συναρτήσεις. Η πρώτη συνάρτηση επιστρέφει την τετραγωνική ρίζα ενός ακεραίου αριθμού που διαβιβάστηκε σε αυτή· καθώς και η δεύτερη συνάρτηση επιστρέφει την κυβική ρίζα.

36. Να γραφεί συνάρτηση που επιστρέφει το εμβαδό κύκλου, με μεταβλητή διπλής ακρίβειας, δεδομένης της ακτίνας, επίσης διπλής ακρίβειας, η οποία διαβιβάζεται από την κύρια συνάρτηση main( ). (Υπενθυμίζεται ότι, το εμβαδό κύκλου δίδεται από τον τύπο: $E = \pi r^2$, όπου $\pi = 3.14159$.)

37. Να γραφεί συνάρτηση που επιστρέφει την τιμή του πολυωνύμου: $9x^4 + 15x^2 + x^1$. Η τιμή του x διαβιβάζεται από την main( ) και δίνεται από τον χρήστη.

ΟΜΑΔΑ Β

1. Να γραφεί ένα πρόγραμμα το οποίο :
(α) θα διαβάζει δύο ακεραίους θετικούς αριθμούς
(β) Θα τυπώνει και θα βρίσκει τον μέγιστο κοινό διαιρέτη τους.

2.Να γραφεί ένα πρόγραμμα για την εύρεση του ΜΚΔ δύο ακεραίων χωρίς να χρησιμοποιηθεί ο αλγόριθμος του Ευκλείδη.

3.Να γραφεί ένα πρόγραμμα για την εύρεση του ΜΚΔ 5 αριθμών.

4.Να γραφεί ένα πρόγραμμα για την εύρεση του ΕΚΠ 5 θετικών ακεραίων αριθμών.

5. Να γραφεί ένα πρόγραμμα το οποίο θα διαβάζει 1000 ακεραίους και θα υπολογίζει και θα

τυπώνει τον μέσο όρο των περιττών.

6. Να γραφεί ένα πρόγραμμα C++ το οποίο θα διαβάζει 50 πραγματικούς αριθμούς $X_1, X_2, X_3, X_4, \ldots$ και θα υπολογίζει και θα τυπώνει το πλήθος αυτών που είναι μεγαλύτεροι από τον μέσο όρο όλων αυτών των αριθμών.

7. Να γραφεί ένα πρόγραμμα C++ το οποίο θα διαβάζει 50 πραγματικούς αριθμούς $X_1, X_2, X_3, X_4, \ldots$ και θα υπολογίζει και θα τυπώνει το άθροισμα αυτών που είναι μικρότεροι από τον μέσο όρο όλων αυτών των αριθμών.

8. Να γραφεί ένα πρόγραμμα C++ το οποίο θα διαβάζει 50 πραγματικούς αριθμούς $X_1, X_2, X_3, X_4, \ldots$ και θα υπολογίζει και θα τυπώνει το ποσοστό αυτών που είναι μεγαλύτεροι από τον X_1.

9. Να γραφεί ένα πρόγραμμα C++ το οποίο:

α) Θα διαβάζει έναν πραγματικό αριθμό Χ.

β) Θα υπολογίζει τις 100 ποσότητες 3(Χ-3), 5(Χ-4), 7(Χ-5), 9(Χ-6), . . .

γ) Θα βρίσκει και θα τυπώνει τη μεγαλύτερη από αυτές.

10. Να γραφεί ένα πρόγραμμα C++ το οποίο:

α) Θα διαβάζει έναν πραγματικό αριθμό Χ.

β) Θα υπολογίζει τις ποσότητες 4(Χ+4), 6(Χ+5), 8(Χ+6), 10(Χ+7), . . . μέχρι να βρεί μια μεγαλύτερη από το 100.

γ) Θα βρίσκει και θα τυπώνει το πλήθος αυτών που είναι έξω από το διάστημα [20, 50].

11. Να γραφεί ένα πρόγραμμα C++ το οποίο:

α) Θα διαβάζει έναν πραγματικό αριθμό Χ.

β) Θα υπολογίζει τις ποσότητες 4(Χ+5), 7(Χ+6), 10(Χ+7), 13(Χ+8), . . . μέχρι να βρεί μια μεγαλύτερη από το 100.

γ) Θα βρίσκει και θα τυπώνει το άθροισμα αυτών που είναι μέσα στο διάστημα [20, 50].

12. Να γραφεί ένα πρόγραμμα C++ το οποίο:

α) Θα διαβάζει έναν πραγματικό αριθμό Χ.

β) Θα υπολογίζει τις ποσότητες 6(Χ-2), 7(Χ-7), 8(Χ-11), 9(Χ-15), . . . μέχρι να βρεί μια τιμή

που να βρίσκεται έξω από το διάστημα [-1000, 1000]. Αυτή η τελευταία τιμή δε συμμετέχει στον παρακάτω υπολογισμό.
γ) Θα βρίσκει και θα τυπώνει το άθροισμα των αρνητικών.

13. Να γραφεί ένα πρόγραμμα C++ το οποίο αρχικά θα διαβάζει τις τιμές των στοιχείων μιας πραγματικής δισδιάστατης δεικτοφόρου μεταβλητής που έχει 10 γραμμές και 15 στήλες. Στη συνέχεια θα υπολογίζει και θα τυπώνει το άθροισμα κάθε γραμμής.

14. Να γραφεί ένα πρόγραμμα C++ το οποίο αρχικά θα διαβάζει τις τιμές των στοιχείων μιας πραγματικής δισδιάστατης δεικτοφόρου μεταβλητής που έχει 15 γραμμές και 10 στήλες. Στη συνέχεια θα υπολογίζει και θα τυπώνει το άθροισμα κάθε στήλης.

15. Να γραφεί ένα πρόγραμμα C++ το οποίο αρχικά θα διαβάζει τις τιμές των στοιχείων μιας πραγματικής δισδιάστατης δεικτοφόρου μεταβλητής που έχει 15 γραμμές και 10 στήλες. Στη συνέχεια θα υπολογίζει και θα τυπώνει τον αριθμό της στήλης που έχει το μεγαλύτερο άθροισμα.

16. Να γραφεί ένα πρόγραμμα C++ το οποίο αρχικά θα διαβάζει τις τιμές των στοιχείων μιας πραγματικής δισδιάστατης δεικτοφόρου μεταβλητής που έχει 10 γραμμές και 15 στήλες. Στη συνέχεια θα υπολογίζει και θα τυπώνει το άθροισμα των αρνητικών στοιχείων κάθε γραμμής.

17. Να γραφεί ένα πρόγραμμα C++ το οποίο αρχικά θα διαβάζει τις τιμές των στοιχείων μιας πραγματικής δισδιάστατης δεικτοφόρου μεταβλητής που έχει 15 γραμμές και 10 στήλες. Στη συνέχεια θα υπολογίζει και θα τυπώνει το πλήθος των θετικών στοιχείων κάθε στήλης.

18. Δίνονται Ν αριθμοί και ζητείται να γραφεί πρόγραμμα το οποίο θα βρίσκει το πλήθος αυτών που είναι μεγαλύτεροι από τον μέσον όρο

19. Να γραφεί ένα πρόγραμμα το οποίο:
α. θα διαβάζει έναν ακέραιο θετικό αριθμό Ν,
β. θα διαβάζει Ν πραγματικούς αριθμούς,
γ. θα υπολογίζει τον μέσον όρο των Ν αριθμών και θα εντοπίζει αυτόν που βρίσκεται πιό

κοντά στον μέσον όρο

δ. θα τυπώνει τον μέσον όρο και τον αριθμό που εντόπισε.

20. Να προσαρμοστεί το προηγούμενο πρόγραμμα έτσι ώστε αν υπάρχουν πολλοί (πάνω από ένας) αριθμοί που ισαπέχουν από τον μέσον όρο, να τυπώνει τη τιμή τους (μια φορά) και επίσης να τυπώνει τον αριθμό της θέσης καθενός από αυτούς.

21. Να γραφεί ένα πρόγραμμα το οποίο:

(α) θα διαβάζει έναν ακέραιο θετικό αριθμό N<100,

(β) θα διαβάζει N+1 πραγματικούς αριθμούς, που είναι οι συντελεστές του πολυωνύμου :

$$a_n x^n + a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + a_{n-3}x^{n-3} + \ldots + a_1 x + a_0$$

βαθμού N,

(γ) θα διαβάζει τρείς πραγματικούς αριθμούς a, b και c,

(δ) θα υπολογίζει και θα τυπώνει τη τιμή του πολυωνύμου στα σημεία x = a, x = a+c, x = a + 2c, . . . εφόσον η τιμή της χ είναι μικρότερη ή ίση της b.

22. Να γραφεί πρόγραμμα το οποίο :

(α) θα διαβάζει τις τιμές των στοιχείων μιας δισδιάστατης δεικτοφόρου μεταβλητής X. Η X έχει 5γραμμές και 7 στήλες.

(β) θα υπολογίζει τις τιμές των στοιχείων δύο μονοδιάστατων δεικτοφόρων μεταβλητών P και Q με 5 και 7 στοιχεία αντίστοιχα. Κάθε στοιχείο της P θα είναι το άθροισμα των στοιχείων της αντίστοιχης γραμμής της X, Κάθε στοιχείο της Q είναι το άθροιαμα των στοιχείων της αντίστοιχης στήλης της X.

(γ) θα τυπώνει τις τιμές των στοιχείων της X, δίπλα σε αυτά της P και μετά (από κάτω) της Q.

23. Εύρεση της στήλης ή της γραμμής μήτρας με το μεγαλύτερο ή μικρότερο άθροισμα των στοιχείων τους.

24. Άθροισμα δυο μητρών. Να γραφεί ένα πρόγραμμα το οποίο:

(α) θα διαβάζει δύο ακεραίους θετικούς αριθμούς K και L, μικρότερους ή ίσους του 100.

(β) θα διαβάζει τις τιμές των στοιχείων δύο μητρών KxL κατά γραμμές.

(γ) θα υπολογίζει και θα τυπώνει το άθροισμα των δύο μητρών

25. Γινόμενο δύο μητρών. Να γραφεί ένα πρόγραμμα το οποίο:
(α) θα διαβάζει τρείς ακεραίους θετικούς αριθμούς K, L και M μικρότερους ή ίσους του 100.
(β) θα διαβάζει τις τιμές των στοιχείων δύο μητρών KxL και LxM κατά γραμμές.
(γ) θα υπολογίζει και θα τυπώνει το γινόμενο της πρώτης μήτρας επί τη δεύτερη

26. Γινόμενο διανύσματος επί μήτρα. Να γραφεί ένα πρόγραμμα το οποίο:
(α) θα διαβάζει δύο ακεραίους θετικούς αριθμούς K και L, μικρότερους ή ίσους του 100.
(β) θα διαβάζει τις τιμές των στοιχείων ενός διανύσματος διάστασης K και μιας μήτρας KxL κατά γραμμές.
(γ) θα υπολογίζει και θα τυπώνει το γινόμενο του διανύσματος επι της μήτρας.

27. Γινόμενο μήτρας επί διάνυσμα. Να γραφεί ένα πρόγραμμα το οποίο:
(α) θα διαβάζει δύο ακεραίους θετικούς αριθμούς K και L, μικρότερους ή ίσους του 100.
(β) θα διαβάζει τις τιμές των στοιχείων μιας μήτρας KxL κατά γραμμές και ενός διανύσματος διάστασης L.
(γ) θα υπολογίζει και θα τυπώνει το γινόμενο της μήτρας επί το διάνυσμα.

28. Τριγωνοποίηση μίας μήτρας: (α) άνω τριγωνική, (β) κάτω τριγωνική.

29. Διαγωνιοποίηση μήτρας.

30. Πήγαμε σε μια υπεραγορά και αγοράσαμε 20 είδη. Για κάθε είδος ξέρουμε την ποσότητα που αγοράσαμε και τη τιμή της μονάδας. Η ποσότητα εκφράζεται πάντα με ένα πραγματικό αριθμό και η τιμή μονάδας με έναν ακέραιο. Να γραφεί ένα πρόγραμμα C++ το οποίο θα διαβάζει αυτά τα 20 ζεύγη αριθμών και θα τυπώνει το συνολικό βάρος των ειδών που είχαν τιμή μονάδας μεγαλύτερη από 500.

31. Πήγαμε σε μια υπεραγορά και αγοράσαμε 20 είδη. Για κάθε είδος ξέρουμε την ποσότητα που αγοράσαμε και τη τιμή της μονάδας. Η ποσότητα εκφράζεται πάντα με ένα πραγματικό αριθμό και η τιμή μονάδας με έναν ακέραιο. Να γραφεί ένα πρόγραμμα C++ το οποίο θα διαβάζει αυτά τα 20 ζεύγη αριθμών και θα τυπώνει το πλήθος των ειδών από τα οποία αγοράσαμε λιγότερο από ένα κιλό.

32. Πήγαμε σε μια υπεραγορά και αγοράσαμε 20 είδη. Για κάθε είδος ξέρουμε την

ποσότητα που αγοράσαμε και τη τιμή της μονάδας. Η ποσότητα εκφράζεται πάντα με ένα πραγματικό αριθμό και η τιμή μονάδας με έναν ακέραιο. Να γραφεί ένα πρόγραμμα C++ το οποίο θα διαβάζει αυτά τα 20 ζεύγη αριθμών και θα τυπώνει το βάρος του είδους που είχε τη μικρότερη τιμή μονάδας.

ΒΙΒΛΙΟΓΡΑΦΙΑ

Α. ΣΥΣΤΗΜΑΤΙΚΟΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΚΑΙ ΓΛΩΣΣΕΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

1. Δανιηλόπουλος, Σ.Δ. "**Εισαγωγή στην Υπολογιστική**". Διδακτικό Βιβλίο. Πανεπιστήμιο ιωαννίνων . ιωάννινα 1980.

2. Δανιηλόπουλος, Σ.Δ. "**Η Ελληνική Αλγοριθμική Γλώσσα (ΕΑΓ)**". Επιμέλεια - Παρουσίαση Σ.Δ. Μπαλτζή. Πανεπιστήμιο ιωαννίνων. Υπό Έκδοση.

3. Μαρινάκης - Μπακογιάννης. "**Αλγοριθμική - Δομημένες Τεχνικές**". Ελιξ. 1990.

4. Μπαλτζής, Σ.Δ. "**Μεθοδολογία και Τεχνικές Προγραμματισμού**". Πανεπιστημιακό Σύγγραμμα. Πανεπιστήμιο ιωαννίνων. 3η έκδοση ιωάννινα 1993.

5. Μπαλτζής, Σ.Δ. - Ζησάκος, Β. "**Η ΕΑΓ- Ελληνική Αλγοριθμική Γλώσσα - ως Γλώσσα Προγραμματισμού**", Τεχνική Αναφορά Αριθμ. 251, Τμήμα ΜΑΘ, Πανεπιστήμιο ιωαννίνων, ιωάννινα, Ιούνιος 1995.

6. Dahl, O.-J., Dijkstra, E.W. and Hoare C.A.R. "**Structured Programming**". Academic Press. 1972

7. Wirth, N. "**Systematic Programming: An Introduction**". Prentice Hall. 1973.

8. MacLennan,B.J. "**Principles of Programming Languages**". HRW The Dryden Press. 1987.

9. Pratt, T.W. "**Programming Languages - Design and Implementation**". Prentice-Hall.1984.

10. Tucker, A.B. "**Programming Languages**". McGraw-Hill Book Company. 1986.

Β. ΓΛΩΣΣΕΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ C, C++ ΚΑΙ UNIX

11. Δελαρόκας, Χ.- Κάβουρας, Ι.- Χαλκιόπουλος, Χ. **"Το Λειτουργικό Σύστημα UNIX και η Γλώσσα Προγραμματισμού C"**. ΕΠΥ. 1986.

12. Λεοντίτσης, Α. **"Προγραμματισμός Ηλεκτρονικών Υπολογιστών C++"**. Πανεπιστημιακό Σύγγραμμα. Πανεπιστήμιο Ιωαννίνων. 1995.

13. Σταμούλης, Α. **"Το Βιβλίο της C++"**. Learning Plan. 1992.

14. Kernighan,B.W.- Pike,R. **"Το Περιβάλλον Προγραμματισμού UNIX"**. Κλειδάριθμος. 1989.

15. Kernighan,B.W.- Ritchie,D.M. **"Η Γλώσσα Προγραμματισμού C"**. Κλειδάριθμος.1988.

16. Perry, G. **"Εγχειρίδιο C++ με Παραδείγματα"**. Β. Γκιούρδας Εκδοτική. 1993.

17. Schildt, H. **"Μάθετε εύκολα τη Γλώσσα C"**. Παρατηρητής. 1988.

18. Waite (ομάδα). **"Πλήρες Εγχειρίδιο της C++ (Καλύπτει Unix και MS-DOS)"**. Εκδότης Μ. Γκιούρδας. 1988.

19. Waite (ομάδα). **"C: Βήμα-προς-Βήμα"**. Εκδότης Μ. Γκιούρδας. 1991.

Γ. ΕΙΣΑΓΩΓΙΚΑ ΚΑΙ ΓΕΝΙΚΟΥ ΠΕΡΙΕΧΟΜΕΝΟΥ Η/Υ ΕΓΚΥΚΛΟΠΑΙΔΕΙΕΣ ΚΑΙ ΛΕΞΙΚΑ Η/Υ

20. Κοίλια, Χρ.- Καλαφατούδη, Στρ. **"Το Πρώτο Βιβλίο της Πληροφορικής"**. Εκδόσεις Νέων Τεχνολογιών. 3η Έκδοση 1993.

21. - **"Πληροφορική : Μία Πρώτη Γνωριμία"**. Εκδόσεις Νέων Τεχνολογιών. 1986.

22. Capron, H.L. **"Essentials of Computing"**. The Benjamin / Cummings Publishing Company, Inc. 1992.

23. Long, L.- Long, N. **"Computers"**. Prentice-Hall. Second Edition 1990.

24. Thomas, R.M. "Ο Μικρός Οδηγός του DOS 6.2". Κλειδάριθμος. 1994.

25. - "Εγκυκλοπαίδεια Πληροφορικής και Τεχνολογίας Υπολογιστών". Software Πληροφορική ΑΕ. 1986.

26. - "Εγκυκλοπαιδικό Λεξικό Πληροφορικής". Εκδόσεις Πελεκάνος. 1987.

27. - "Ελληνικό Γλωσσάριο Πληροφορικής". ΕΠΥ. 1985.

28. Γαρδής, Π.Τ.Κ.- Δεληγιαννάκης, Εμ.Ν. "Σύγχρονο Λεξικό Πληροφορικής". Εκδόσεις Δίαυλος

29. Kraynak - Cagnina. "Πρακτικο Λεξικό Πληροφορικής". Γκιούρδας.

30. QUE (ομάδα). "Το Λεξικό του Προγραμματιστή". Γκιούρδας.

ΠΑΡΑΡΤΗΜΑ 1

ΣΥΝΤΑΚΤΙΚΗ ΠΕΡΙΓΡΑΦΗ ΤΗΣ ΕΛΛΗΝΙΚΗΣ ΑΛΓΟΡΙΘΜΙΚΗΣ ΓΛΩΣΣΑΣ (ΕΑΓ)

Π1.1 Εισαγωγή

Η Ελληνική Αλγοριθμική Γλώσσα (ΕΑΓ)[1] , είναι μια γλώσσα περιγραφής αλγορίθμων, μια τυπική γλώσσα τύπου 2 της ιεραρχίας Chomsky, ακολουθεί τις αρχές του συστηματικού (δομημένου) προγραμματισμού και χρησιμοποιεί,

1. την Ελληνική Γλώσσα για τις:

(α) προκαθορισμένες (δεσμευμένες) λέξεις της, δηλαδή αυτές που χρησιμοποιούνται πάντοτε με τον ίδιο τρόπο, και

(β) λέξεις, προσδιοριστές, που κατασκευάζει ο γράφων τον αλγόριθμο από μικρά ή κεφαλαία γράμματα κυρίως του ελληνικού αλφαβήτου, αλλά και του λατινικού αν χρειαστεί, των οποίων η σημασία ορίζεται στο δηλωτικό τμήμα,

2. τα ψηφία του δεκαδικού συστήματος αρίθμησης, και τέλος

3. τα απαραίτητα ειδικά σύμβολα (τελεστές, οριοθέτες κλπ.).

Η ΕΑΓ επινοήθηκε απο τον αείμνηστο Στυλιανό Δανηλόπουλο[2] , Αναπληρωτή Καθηγητή του Τμήματος Μαθηματικών του Πανεπιστημίου Ιωαννίνων. Ήταν το απόσταγμα μιας μακροχρόνιας εμπειρίας στην εκπαίδευση, με στόχους κυρίως την αποτελεσματικότερη διδασκαλία χρησιμοποιώντας την Ελληνική Γλώσσα.

---

[1] Δανηλόπουλος, Σ.Δ. "Η Ελληνική Αλγοριθμική Γλώσσα (ΕΑΓ)". Επιμέλεια - Παρουσίαση Σ.Δ.Μπαλτζή. Πανεπιστήμιο Ιωαννίνων. Υπό Έκδοση.

[2] Δανηλόπουλος, Σ.Δ. "Εισαγωγή στην Υπολογιστική". Διδακτικό Βιβλίο.Πανεπιστήμιο Ιωαννίνων. Ιωάννινα 1990.

Η ΕΑΓ, ως γλώσσα περιγραφής αλγορίθμων, χρησιμοποιήθηκε σε επίπεδο διδασκαλίας και συγγραφής[3], πάνω από μια δεκαπενταετία στην εκπαίδευση, την ανωτάτη, την ανωτέρα, τη μέση, τη τεχνική, σε παντός τύπου επιμορφωτικά σεμινάρια Η/Υ, όπου δηλαδή κληθήκαμε να διδάξουμε. Πρέπει να επισημάνουμε ότι, η ΕΑΓ έτυχε πάντοτε της άμεσης αποδοχής, υιοθετήθηκε, σημειώνοντας μεγάλη επιτυχία ως προς τα αποτελέσματα της διδασκαλίας - για παράδειγμα, του σχεδιασμού, ή της παρουσίασης ή της ανάλυσης αλγορίθμων, εν γένει των αρχών, της μεθοδολογίας και των τεχνικών του προγραμματισμού ακόμα και των γλωσσών προγραμματισμού - τα οποία ήταν γρήγορα και άμεσα, αφού χρησιμοποιούσαμε μια τυπική γλώσσα που τα νοήματά της ήταν αμέσως κατανοητά από το ακροατήριο. Έτσι, γινόταν πραγματικότητα σε μικρό χρόνο, η πειθαρχία και η εξοικείωση με τις αρχές γραφής και σύνθεσης αλγορίθμων, όπως και της γρήγορης κατανόησης και εκμάθησης σύνθετων αλγορίθμων που περιέγραφαν ειδικές διαδικασίες σε διάφορους τομείς της Ε.Η/Υ. Το αξιοσημείωτο είναι ότι τα παραπάνω παρατηρήθηκαν και σε ακραίες περιπτώσεις, δηλαδή σε αρχάρια άτομα, πολύ νεαρά ή κάποιας ηλικίας ή ακόμα και σε άτομα άλλης νοοτροπίας.

Η υλοποίηση της ΕΑΓ ως γλώσσας προγραμματισμού[4] δεν είχε ως στόχο να προσθέσει μια ακόμα γλώσσα προγραμματισμού στην υπάρχουσα ποικιλία, αλλά εντάσσεται στα πλαίσια της συμβολής μας στη γενικότερη προσπάθειά ώστε:

1. παράλληλα με την αλγοριθμοποίηση και ο προγραμματισμός, να γίνει ευκολόχρηστο και άμεσα κατανοητό εργαλείο στους Ελληνόπαιδες σε όλες τις βαθμίδες της εκπαίδευσης, ιδίως εκεί όπου ο προγραμματισμός εμφανίζεται σε εισαγωγικό στάδιο. Διότι δίδεται η δυνατότητα στον σπουδαστή να τρέξει άμεσα αυτό το οποίο άμεσα κατανοεί και γράφει,

---

[3] Δανηλόπουλος, Σ.Δ. "Εισαγωγή στην Υπολογιστική". Διδακτικό Βιβλίο.Πανεπιστήμιο Ιωαννίνων. Ιωάννινα 1980.

Δανηλόπουλος, Σ.Δ. "Δομές Δεδομένων" Πρώτος Τόμος. Διδακτικό Βιβλίο. Πανεπιστήμιο Ιωαννίνων. Ιωάννινα 1982.

Δανηλόπουλος, Σ.Δ. "Εισαγωγή στη Σχεδίαση και Ανάλυση Αλγορίθμων". Πανεπιστημιακό Σύγγραμμα. Πανεπιστήμιο Ιωαννίνων. Ιωάννινα 1988.

Δανηλόπουλος, Σ.Δ. "Θεμελιώδεις Έννοιες Γλωσσών Προγραμματισμού". Πανεπιστημιακό Σύγγραμμα. Πανεπιστήμιο Ιωαννίνων. Ιωάννινα 1991.

Μπαλτζής, Σ.Δ. "Μεθοδολογία και Τεχνικές Προγραμματισμού".Πανεπιστημιακό Σύγγραμμα. Πανεπιστήμιο Ιωαννίνων. 3η έκδοση Ιωάννινα 1993.

[4] Ζησάκος, Β. "Μια Εκδοχή Διερμηνέα της ΕΑΓ". Διπλωματική Εργασία. Πανεπιστήμιο Ιωαννίνων. Ιωάννινα 1994.

Μπαλτζής,Σ.Δ.- Ζησάκος,Β. "Η ΕΑΓ- Ελληνική Αλγοριθμική Γλώσσα ως Γλώσσα Προγραμματισμού". Τεχνική Αναφορά Αριθμ 251. Τμήμα Μαθηματικών - Πανεπιστήμιο Ιωαννίνων, Ιούνιος 1995

επιπλέον τον προετοιμάζει, χωρίς ιδιαίτερη προσπάθεια, στο να μυείται, να εξοικειώνεται ευκολότερα στη φιλοσοφία και χρήση οποιασδήποτε γλώσσας προγραμματισμού και

2. η τροφοδότης των επιστημών σε όρους, Ελληνική Γλώσσα, η πλούσια και εύπλαστη με τη τρισχιλιετή και πλέον ιστορία της, να πάρει, όλο και δυναμικότερα, τη θέση που της αξίζει, αλλά και μέρος στις νέες πραγματικότητες (όπως π.χ. την αλγοριθμοποίηση, τον προγραμματισμό, κλπ), στις νέες διαστάσεις της καθημερινής μας ζωής (την εν γένει χρήση του Η/Υ), στη νέα επιστήμη των Η/Υ (προτυποίηση, καθιέρωση δόκιμης ορολογίας), τουλάχιστον στη κοιτίδα της, τη μητροπολιτική Ελλάδα.

Π1.1 Δομή της Διαδικασίας

1. <διαδικασια> ::= <επικεφαλιδα><δηλωτικο><εκτελεσιμο>

Οι τρείς συντακτικές κατηγορίες, που περιγράφουν τα τμήματα από τα οποία αποτελείται η διαδικασία, δίδονται από τους ορισμούς:

2. <επικεφαλιδα> ::= **διαδικασια**<ονομα_διαδικασιας>;|
διαδικασια<ονομα_διαδικασιας><τμημα_παραμετρων>;|
διαδικασια<ονομα_διαδικασιας_συναρτησης>
<τμημα_παραμετρων><τμημα_τυπου> ;

3. <δηλωτικο> ::= <ακολουθια_δηλωσεων>;

4. <εκτελεσιμο> ::= **αρχη** <σωμα> **τελος** <ονομα_διαδικασιας> |
αρχη <εσωτερικες_δηλωσεις> <σωμα> **τελος** <ονομα_διαδικασιας>

Οι νέες συντακτικές κατηγορίες που εμφανίστηκαν στους παραπάνω ορισμούς ορίζονται ως εξής:

5. <εσωτερικες_δηλωσεις> ::= <ακολουθια_δηλωσεων>;

6. <ακολουθια_δηλωσεων> ::= <δηλωστ> |
<ακολουθια_δηλωσεων>;<δηλωστ>

7. <δηλωστ> ::= <τμημα_μεταβλητων> | <δηλωση_παραταξεων>

8. <τμημα μεταβλητων> ::= **δηλωση** <μεταβλητη><τυπος> |

δηλωση (<ακολουθια_μεταβλητων>)<τυπος>

9. <δηλωση_παρατάξεων> ::= **δηλωση** <παραταξη> **παραταξη** <τυπος> |
δηλωση (<ακολουθια_παραταξεων>) **παραταξη** <τυπο>

10. <ακολουθια_μεταβλητων> ::= <μεταβλητη> |
<ακολουθια_μεταβλητων> , <μεταβλητη>

11. <ακολουθια_παραταξεων> ::= <παραταξη> |
<ακολουθια_παραταξεων> , <παραταξη>

12. <τυπος> ::= **ακεραια** | **πραγματικη** | **λογικη** | **αλυσιδα_χαρακτηρων**

13. <ονομα_διαδικασιας> ::= <προσδιοριστης>

14. <ονομα_διαδικασιας_συναρτησης> ::= <προσδιοριστης>

15. <μεταβλητη> ::= <προσδιοριστης>

16. <παραταξη> ::= <ονομα_παραταξης><τμημα_δεικτων>

17. <ονομα_παραταξης> ::= <προσδιοριστης>

18. <τμημα_δεικτων> ::= [<ακολουθια_ζευγων_φραγματων>]

19. <ακολουθια_ζευγων_φραγματων> ::=
<ζευγος_φραγματων> |
<ακολουθια_ζευγων_φραγματων> , <ζευγος_φραγματων>

20. <ζευγος_φραγματων> ::= <πρωτος_δεικτης> :<τελευταιος_δεικτης>

21. <πρωτος_δεικτης> ::= <ακεραιος>

22. <τελευταιος_δεικτης> ::= <ακεραιος>

23. <ακεραιος> ::= <μη_αρνητικος_ακεραιος> |
<προσημο> <μη_αρνητικος_ακεραιος>

24. <μη_αρνητικος_ακεραιος> ::= <ψηφιο> |

<μη_αρνητικος_ακεραιος><ψηφιο>

25. <ψηφιο> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

26. <προσημο> ::= -

27. <προσδιοριστης> ::= <αλφαβητικος χαρακτηρας> |
<προσδιοριστης><αλφαριθμητικος χαρακτηρας>

28. <αλφαριθμητικος χαρακτηρας> ::= <αλφαβητικος χαρακτηρας> |
<ψηφιο>

29. <αλφαβητικος χαρακτηρας> ::= <ελληνικος_αλφ. χαρακτηρας> |
<κεφαλαιος_ελλην. αλφ. χαρακτηρας> |
<λατινικος_αλφ. χαρακτηρας> |
<κεφαλαιος_λατιν. αλφ. χαρακτηρας>

30. <ελληνικος_αλφ. χαρακτηρας> ::= α | β | γ | δ | ε | ζ | η | θ | ι | κ | λ | μ |
ν | ξ | ο | π | ρ | σ | τ | υ | φ | χ | ψ | ω

31. <κεφαλαιος_ελλην. αλφ. χαρακτηρας> ::= Α | Β | Γ | Δ | Ε | Ζ | Η | Θ | Ι | Κ | Λ | Μ |
Ν | Ξ | Ο | Π | Ρ | Σ | Τ | Υ | Φ | Χ | Ψ | Ω

32. <λατινικος_αλφ. χαρακτηρας> ::= a | b | c | d | e | f | g | h | i | j | k | l | m |
n | o | p | q | r | s | t | u | v | w | x | y | z

33. <κεφαλαιος_λατιν. αλφ. χαρακτηρας> ::= A | B | C | D | E | F | G | H | I | J | K | L | M |
N | O | P | Q | R | S | T | U | V | W | X | Y | Z

34. <στοιχειο_παραταξεως> ::= <ονομα_παραταξεως>[<ακολουθια_δεικτων>]

35. <ακολουθια_δεικτων> ::= <δεικτης> | <ακολουθια_δεικτων>, <δεικτης>

36. <δεικτης> ::= <ακεραιος>

37. <σωμα> ::= <ακολουθια_προτασεων>

38. <ακολουθια_προτασεων> ::= <προταση> |
<ακολουθια_προτασεων>;<προταση>

39. <προταση> ::= <αναθεση> |
<υποθετικη προταση> |
<βαθμιδα επαναληψεως> |
<κληση διαδικασιας> |
<εισαγωγη-εξαγωγη>

Π1.2 Πρόταση Ανάθεσης

40. <αναθεση> ::= <μεταβλητη> ← <εκφραση>

41. <μεταβλητη> ::= <απλη_μεταβλητη> | <δεικτοφορος_μεταβλητη>

42. <απλη_μεταβλητη> ::= <προσδιοριστης>

43. <δεικτοφορος_μεταβλητη> ::= <στοιχειο_παραταξεως>

44. <εκφραση> ::= <υποεκφραση> |
<εκφραση><τελεστης_1><υποεκφραση>

45. <υποεκφραση> ::= <πρωτ> | <υποεκφραση><τελεστης_2><πρωτ>

46. <πρωτ> ::= <σταθερα> | <μεταβλητη> | (<εκφραση>)

47. <εκφραση> ::= <αριθμητικη_εκφραση> | <σχεσιακη_εκφραση> |
<λογικη_εκφραση> | <εκφραση_αλυσιδων>

48. <αριθμητικη_εκφραση> ::= <ορος> |
<αριθμητικη_εκφραση><τελ +,-><ορος>

49. <ορος> ::= <παραγων> | <ορος><τελ *, /><παραγων>

50. <παραγων> ::= <πρωτ> | <παραγων>**<πρωτ>

51. <πρωτ> ::= <σταθερα> | <μεταβλητη> |
<αναφορα_συναρτησεως> | (<εκφραση>)

52. <τελ +,-> ::= + | -

53. <τελ ',/> ::= ' | /

54. <σχεσιακη_εκφραστ> ::= <αριθμητικη_εκφραστ><σχεσιακος_τελεστης>
<αριθμητικη_εκφραστ>

55. <σχεσιακος_τελεστης> ::= < | ≤ | = | ≠ | ≥ | >

56. <λογικη_εκφραστ> ::= <λογικος_ορος> |
<λογικη_εκφραστ>⊥<λογικος_ορος>

57. <λογικος_ορος> ::= <λογικος_παραγων> |
<λογικος_ορος>&<λογικος_παραγων> |
Χ <λογικος_παραγων>

58. <λογικος_παραγων> ::= <λογικη_σταθερα> | <λογικη_μεταβλητη> |
<σχεσιακη_εκφραστ> | (<λογικη_εκφραστ>)

59. <λογικη_σταθερα> ::= ναι | οχι

Π1.3 Υποθετικές Προτάσεις

60. <υποθετικη_προταση> ::= εαν <υποθεση> τοτε <αποδοση> |
εαν <υποθεση> τοτε <αποδοση> αλλιως <αποδοση>

61. <υποθεση> ::= <λογικη_εκφραστ>

62. <αποδοση> ::= <προταση> | <συνθετη_προταση>

63. <συνθετη_προταση> ::= (<ακολουθια_προτασεων>) |
ι(<ακολουθια_προτασεων>)ι όπου ι = 1, 2, 3, . . .

Π1.4 Βαθμίδες Επανάληψης

64. <βαθμιδα_επαναληψεως> ::= <επαναληψη_εφοσον> |
<επαναληψη_εως> |
<επαναληψη_ορισμενη>

65. <επαναληψη_εφοσον> ::= **εφοσον** <λογικη_εκφραση> **επαναλαβε**
(<συνθετη_προταση>)

66. <επαναληψη_εως> ::= **επαναλαβε** <συνθετη_προταση> **εως**
<λογικη εκφραση>

67. <επαναληψη_ορισμενη>::=
για <ακεραια_αναθεση> **εως** <ακεραια_εκφραση> **επαναλαβε**
(<συνθετη προταση>) |
για <ακεραια αναθεση> **εως** <ακεραια εκφραση> **κατα** <ακεραια εκφραση> **επαναλαβε**
(<συνθετη προταση>)

68. <ακεραια_αναθεση> ::= <ακεραια_μεταβλητη> ← <ακεραια_εκφραση>

69. <ακεραια_μεταβλητη> ::= <προσδιοριστης>

70. <ακεραια_εκφραση> ::= <αριθμητικη_εκφραση>

Π1.5 Κλήσεις Διαδικασιών

Η εκτέλεση ενός αλγόριθμου περιγράφεται ως εκτέλεση, ή "**κλήση**" μιας διαδικασίας. Σύμφωνα με τον κανόνα 2, μια διαδικασία μπορεί να οριστεί έτσι ώστε είτε να έχει παραμέτρους, είτε να μην έχει παραμέτρους. Οι παράμετροι μεταβιβάζουν δεδομένα στον αλγόριθμο, και επιτρέπουν τη χρησιμοποίηση του ίδιου αλγόριθμου με πολλά διαφορετικά σύνολα δεδομένων εισόδου. Το "**τμήμα παραμέτρων**", που, όταν υπάρχει, βρίσκεται σύμφωνα με τον κανόνα στην επικεφαλίδα της διαδικασίας, ορίζεται ως εξής:

71. <τμημα_παραμετρων> ::= (<ακολουθια_τυπικων_παραμετρων>)

72. <ακολουθια_τυπικων_παραμετρων> ::=
<τυπικη_παραμετρος> |
<ακολουθια_τυπικων_παραμετρων> , <τυπικη_παραμετρος>

73. <τυπικη_παραμετρος> ::= <ονομα_μεταβλητης> |
<ονομα_παραταξεως> |
<ονομα_διαδικασιας>

74. <ονομα_μεταβλητης> ::= <προσδιοριστης>

75. <τμημα_τυπου> ::= **επιστρεφει(<τυπος>)**

Κατα την εκτέλεση μιας διαδικασίας κάθε **τυπική παράμετρος** αντικαθιστάται από μια **"πραγματική παράμετρο"** η οποία, γενικά, είναι μια έκφραση (όπως ορίζεται από τους κανόνες 47-59). Οι αντίστοιχοι κανόνες είναι οι εξής:

76. <κληση_διαδικασιας> ::=
κληση <ονομα_διαδικασιας> |
κληση <ονομα_διαδικασιας> <τμημα_πραγματικων_παραμετρων>

77. <αναφορα_συναρτησης> ::=
<ονομα_διαδικασιας_συναρτησης><τμημα_πραγματικων_παραμετρων>

78. <τμημα_πραγματικων_παραμετρων> ::= (<ακολουθια πραγματικων_παραμετρων>)

79. <ακολουθια_πραγματικων_παραμετρων> ::=
<πραγματικη_παραμετρος> |
<ακολουθια_πραγματικων_παραμετρων> ,<πραγματικη_παραμετρος>

80. <πραγματικη_παραμετρος> ::= <εκφραστ>

Π1.6 Εισαγωγή Δεδομένων - Εξαγωγή Αποτελεσμάτων

Σε μια γλώσσα περιγραφής αλγορίθμων δεν απαιτούνται πολύπλοκοι μηχανισμοί εισαγωγής δεδομένων και εξαγωγής αποτελεσμάτων. Γι' αυτό θα περιοριστούμε να ορίσουμε:

81. <εισαγωγη_εξαγωγη> ::= <εισαγωγη> | <εξαγωγη>

82. <εισαγωγη> ::= **διαβασε** <ακολουθια_στοιχειων_εισοδου>;

83. <ακολουθια_ στοιχειων_εισοδου> ::=
<στοιχειο_εισοδου> |
<ακολουθια_στοιχειων_εισοδου>,<στοιχειο_εισοδου>

84. <στοιχειο_εισοδου> ::= <απλη_μεταβλητη> |
<δεικτοφορος_μεταβλητη> |
<ονομα_παραταξεως>

85. <εξαγωγη> ::= **γραψε** <ακολουθια_στοιχειων_εξοδου>; |
τυπωσε <ακολουθια_στοιχειων_εξοδου>;

86. <ακολουθια_στοιχειων_εξοδου> ::=
<στοιχειο_εξοδου> |
<ακολουθια_στοιχειων_εξοδου>,<στοιχειο_εξοδου>

87. <στοιχειο_εξοδου> ::= <εκφραση>

Π1.7 Παραδείγματα Περιγραφής Αλγορίθμων

Παραθέτουμε παραδείγματα απλών αλγορίθμων, τα οποία έχουν ως σκοπό να δείξουν τον τρόπο χρησιμοποιήσης της γλώσσας περιγραφής ΕΑΓ.

Ο καλλίτερος τρόπος γραφής μιας διαδικασίας είναι εκείνος ο οποίος επιτρέπει να διακρίνονται τα σώματα των διαδικασιών, και οι σύνθετες προτάσεις που αποτελούν, κατά κάποιο τρόπο, "σώματα" των αποδόσεων υποθετικών προτάσεων, ή βαθμίδων επανάληψης. Η διάκριση αυτή επιτυγχάνεται με την προς τα δεξιά μετάθεση, στο κείμενο της διαδικασίας, των **εσωτερικών βαθμίδων**, ή **συνθέτων προτάσεων**, όπως φαίνεται στο Σχ. 1.7.1, όπου δίδουμε το κείμενο της διαδικασίας ΑΘΡΟΙΣΜΑ, σε πιό βελτιωμένη μορφή.

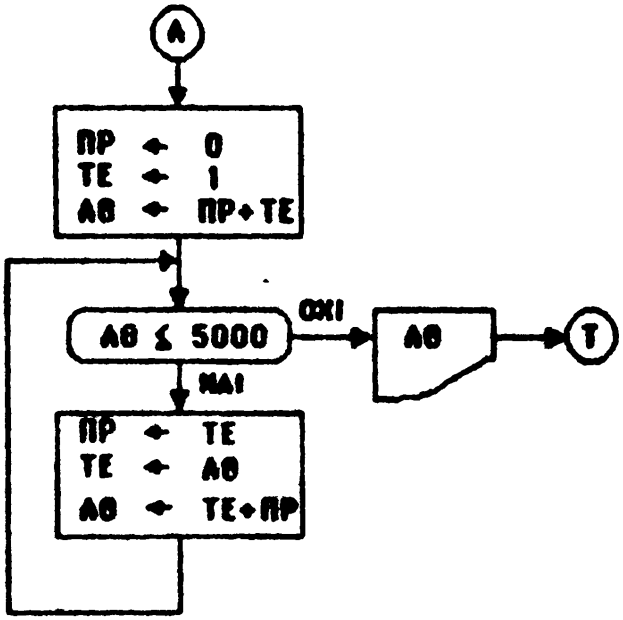
```
1     διαδικασια ΑΘΡΟΙΣΜΑ (Α, Ν, ΑΘΡ);
2     δηλωση (ΑΘΡ, Ν) ακεραια;
3     δηλωση Α[1:50] παραταξη ακεραια;
4     αρχη
4.1       δηλωση i ακεραια;
4.2       ΑΘΡ ← 0;
4.3       για i ← 1 εως Ν επαναλαβε
4.3.1                ( ΑΘΡ ← ΑΘΡ + Α[i] ;)
5     τελος ΑΘΡΟΙΣΜΑ
```

ΣΧΗΜΑ 1.7.1 Δομημένο κείμενο της διαδικασίας ΑΘΡΟΙΣΜΑ.

Στο Σχήμα 1.7.1 αριθμήσαμε τις προτάσεις (ή, ακριβέστερα, τις σειρές του κειμένου) της διαδικασίας, χρησιμοποιώντας μια αρίθμηση την οποία ονομάζουμε **δεκαδική αρίθμηση**

Dewey, κατ' αναλογία με τον **δεκαδικό συμβολισμό Dewey**, που χρησιμοποιείται για την γραμμική παράσταση δένδρων. Παρατηρούμε ότι οι προτάσεις με αρίθμηση 4.1, 4.2, 4.3. είναι "**εσωτερικές**" της βαθμίδας που οριοθετείται από τον οριοθέτη **αρχη**, και αριθμείται με 4. Η 4.3.1 είναι "**εσωτερική**" της 4.3.1. Όσο αυξάνουν οι στάθμες κιβωτισμού, τόσο αυξάνουν οι στάθμες δεκαδικών ψηφίων της αριθμήσης. Ο τρόπος δόμησης του κειμένου μιας διαδικασίας θα φανεί καλλίτερα και από τα παρακάτω παραδείγματα.

Υπολογισμός αριθμών Fibonacci. Οι αριθμοί Fibonacci είναι μια ακολουθία ακεραίων που έχουν την ιδιότητα ότι κάθε όρος της ακολουθίας ισούται με το άθροισμα των δύο προηγουμένων όρων. Ως πρώτοι δύο αριθμοί Fibonacci λαμβάνονται συνήθως οι 0 και 1, και έτσι η αντίστοιχη ακολουθία είναι η 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, . . .



ΣΧΗΜΑ 1.7.2 Διάγραμμα ροής για τον υπολογισμό των αριθμών Fibonacci που δεν υπερβαίνουν την τιμή 5000. Εκτυπώνεται ο τελευταίος αριθμός.

Στο Σχήμα 1.7.2 περιγράφουμε, με την μέθοδο του διαγράμματος ροής, έναν αλγόριθμο ο οποίος υπολογίζει τους αριθμούς Fibonacci που δεν υπερβαίνουν την τιμή 5000 και εκτυπώνει την τιμή του τελευταίου.

Το σώμα του αλγόριθμου αποτελείται από έξι 'κύριες' προτάσεις. Μία εσωτερική δήλωση, τρείς προτάσεις ανάθεσης, μια βαθμίδα επανάληψης, η οποία περιέχει τρείς

προτάσεις ανάθεσης, που αποτελούν τη σύνθετη πρόταση της βαθμίδας επανάληψης και μια πρόταση εξόδου.

Επειδή ο αλγόριθμος υπολογίζει πάντοτε τους ίδιους αριθμούς, δεν υπάρχει θέμα χρησιμοποίησής του με διαφορετικές παραμέτρους εισόδου. Γι' αυτό η επικεφαλίδα της διαδικασίας δεν περιέχει τμήμα παραμέτρων. Με αυτά τα δεδομένα μπορούμε να γράψουμε τη διαδικασία του Σχήματος 1.7.3.

```
1     διαδικασια FIBONACCI;
2     αρχη
2.1         δηλωση (ΤΕΛΕΥΤΑΙΟΣ, ΠΡΟΤΕΛΕΥΤΑΙΟΣ, ΑΘΡΟΙΣΜΑ) ακεραια;
2.2         ΠΡΟΤΕΛΕΥΤΑΙΟΣ ← 0;
2.3         ΤΕΛΕΥΤΑΙΟΣ ← 1;
2.4         ΑΘΡΟΙΣΜΑ ← 1;
2.4         εφοσον ΑΘΡΟΙΣΜΑ ≤ 5000 επαναλαβε
2.4.1           ( ΠΡΟΤΕΛΕΥΤΑΙΟΣ ← ΤΕΛΕΥΤΑΙΟΣ;
2.4.2             ΤΕΛΕΥΤΑΙΟΣ ← ΑΘΡΟΙΣΜΑ;
2.4.3             ΑΘΡΟΙΣΜΑ ← ΤΕΛΕΥΤΑΙΟΣ + ΠΡΟΤΕΛΕΥΤΑΙΟΣ ;)
2.5         γραψε ΑΘΡΟΙΣΜΑ;
3     τελος FIBONACCI
```

ΣΧΗΜΑ 1.7.3 Αλγόριθμος για τον υπολογισμό των αριθμών FIBONACCI.

Παρατηρούμε ότι σ' αυτόν τον αλγόριθμο είναι ανάγκη να χρησιμοποιηθεί η βαθμίδα <επαναληψη_εφοσον>, γιατί ο αριθμός επαναλήψεων δεν είναι γνωστός. Με κάποιες μικρές τροποποιήσεις θα ήταν δυνατή η χρήση της βαθμίδας <επαναληψη_εως>. **Σημειώνουμε**, όμως, ότι, γενικά, για κάθε συγκεκριμένο πρόβλημα η χρήση μιας από τις δύο βαθμίδες δίδει έναν απλούστερο αλγόριθμο.

Υπολογισμός γινομένου μητρών. Θεωρούμε δύο μήτρες Α, Β μεγέθους $n \times n$. Για τον υπολογισμό του γινομένου $C = A \cdot B$, πρέπει να υπολογισθεί κάθε στοιχείο της C, από τη σχέση:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} \times b_{kj}$$

Αυτό απαιτεί τον κιβωτισμό τριών βαθμίδων επανάληψης, όπως φαίνεται από την διαδικασία του Σχήματος 1.7.4. Οι μήτρες Α και Β είναι τα δεδομένα εισόδου, ενώ η μήτρα C αποτελεί τα δεδομένα εξόδου. Οι ακέραιες μεταβλητές i, j, k, και η πραγματική μεταβλητή S, που χρησιμεύει ως αθροιστής για κάθε στοιχείο της C, είναι μεταβλητές που αφορούν μόνο τους εσωτερικούς υπολογισμούς της διαδικασίας. Γι' αυτό είναι ενδεδειγμένο να δηλώνονται ως **"εσωτερικές μεταβλητές"**, δηλαδή σε **εσωτερικές δηλώσεις**.

```
1       διαδικασια ΓΙΝΟΜΗΤΡΩΝ (A, B, C, n);
2       δηλωση n ακεραια;
3       δηλωση ( A[1:n,1:n],B[1:n,1:n], C[1:n,1:n] ) παραταξη πραγματικη;
4       αρχη
4.1             δηλωση (i, j, k) ακεραια;
4.2             δηλωση S πραγματικη;
4.3             για i ← 1 εως n επαναλαβε
4.3.1                   1( για j ← 1 εως n επαναλαβε
4.3.1.1                         2( S ← 0;
4.3.1.2                             για k ← 1 εως n επαναλαβε
4.3.1.2.1                               3( S ← S + A[i,k] * B[k,j] ;)3
4.3.1.3                             C[i,j] ← S; )2  )1
5       τελος ΓΙΝΟΜΗΤΡΩΝ
```

ΣΧΗΜΑ 1.7.4 Διαδικασία για τον υπολογισμό του γινομένου δύο μητρών.

Παρατηρούμε ότι το μέγιστο βάθος κιβωτισμού (μέσα στο σώμα της διαδικασίας, το οποίο αντιστοιχεί στο δεύτερο δεκαδικό της αριθμήσεως Dewey) είναι 3. **Παρατηρούμε** ακόμα την χρησιμότητα των ιεραρχικών παρενθέσεων, οι οποίες συμβάλλουν πολύ στην αναγνώριση της αρχής και του τέλους συνθέτων προτάσεων, όταν υπάρχει κιβωτισμός.

Υπολογισμός μέγιστου κοινού διαιρέτη. Ο αλγόριθμος που περιγράφεται με τη διαδικασία του Σχήματος 1.7.5, υπολογίζει τον μέγιστο κοινό διαιρέτη ΜΚΔ δύο ακεραίων Α, Β με την στοιχειώδη μέθοδο των διαδοχικών αφαιρέσεων. Η μέθοδος δεν είναι η ταχύτερη, αλλά την περιγράφουμε εδώ γιατί μας δίδει την ευκαιρία να χρησιμοποιήσουμε υποθετικές προτάσεις και άλλα χρήσιμα 'συστατικά' για την γραφή αλγορίθμων.

```
1     διαδικασια ΜΕΚΟΙΝΟΔΙΑΙ( Α, Β, ΜΚΔ);
2     δηλωση (Α, Β, ΜΚΔ) ακεραια;
3     αρχη
3.1       δηλωση (ΜΕΓ, ΜΙΚ, Χ) ακεραια;
3.2       εαν Α > Β
3.2.1         τοτε ( ΜΕΓ ← Α;
3.2.2                ΜΙΚ ← Β )
3.2.3         αλλως ( ΜΕΓ ← Β;
3.2.4                ΜΙΚ ← Α ;)
3.3       εφοσον ΜΙΚ ≠ 0 επαναλαβε
3.3.1         1 ( ΜΕΓ ← ΜΕΓ - ΜΙΚ;
3.3.2           εαν ΜΕΓ < ΜΙΚ
3.3.2.1               τοτε 2 ( Χ ← ΜΕΓ;
3.3.2.2                      ΜΕΓ ← ΜΙΚ;
3.3.2.3                      ΜΙΚ ← Χ ;)2 )1
3.4       ΜΚΔ ← ΜΕΓ;
4     τελος ΜΕΚΟΙΝΟΔΙΑΙ
```

ΣΧΗΜΑ 1.7.5 Αλγόριθμος για τον υπολογισμό του μέγιστου κοινού διαιρέτη ΜΚΔ, δύο αριθμών Α και Β.

Δύο αριθμοί a και b, πού έχουν κοινό διαιρέτη, μπορούν να γραφούν ως a = rd, και b = sd. Τότε, όμως, ισχύει και a-b = (r-s)d, απ' όπου προκύπτει ότι οι αριθμοί a, b και c, όπου c = a-b, έχουν κοινό διαιρέτη. Για τον ίδιο λόγο, οι αριθμοί b, c και b-c (υποθέσαμε ότι b > c, αλλιώς θα παίρναμε c-b) έχουν κοινούς διαιρέτες, που είναι και διαιρέτες του a. Συνεχίζοντας με αυτή την διαδικασία, αποκτούμε μια ακολουθία, της οποίας τα μέλη έχουν έναν, τουλάχιστο, κοινό διαιρέτη. Αν και το αποτέλεσμα της αφαίρεσης, από την οποία

προέρχεται ένα νέο μέλος της ακολουθίας (δηλαδή το b-c), δεν είναι κατ' ανάγκη μικρότερο από το μέλος που αφαιρέθηκε (δηλαδή το c), θα επιβάλουμε για κάθε ζεύγος διαδοχικών μελών τη συνθήκη $a_{k-1} > a_k$ η οποία διατηρείται εύκολα, αν αρχίσουμε την ακολουθία παίρνοντας $a_1 = max(a, b)$ και $a_2 = min(a, b)$, και αν υπολογίζουμε το k-στό μέλος σύμφωνα με τον κανόνα:

$u \leftarrow a_k$
$v \leftarrow a_{k-1} - a_k$
εαν $a_k > v$ τότε θέτουμε $a_{k-1} \leftarrow v$
εαν $a_k \leq v$ τότε θέτουμε $a_k \leftarrow v$ και $a_{k-1} \leftarrow u$.

Στην πραγματικότητα, δεν χρειαζόμαστε δεικτοφόρες μεταβλητές, όπως φαίνεται από τη διαδικασία του Σχήματος 1.7.5.

Παρατηρούμε, ακόμα, τη χρήση των δύο ειδών υποθετικών προτάσεων - με μία και με δύο αποδόσεις - και τον μηχανισμό ανταλλαγής τιμών δύο μεταβλητών (προτάσεις 3.3.2.1-3.3.2.3). Η ανταλλαγή γίνεται ώστε να ισχύει πάντοτε η σχέση ΜΕΓ > ΜΙΚ (δηλαδή η συνθήκη $a_{k-1} > a_k$ που επιβάλαμε παραπάνω).

Αναφορές Συναρτήσεων

Όταν το αποτέλεσμα των υπολογισμών που εκτελεί μια διαδικασία είναι ένας και μόνο αριθμός (όχι, π.χ., τα στοιχεία μιας παράταξης), είναι χρήσιμο, σε πολλές περιπτώσεις, αντί να ορίσουμε μια διαδικασία, να ορίσουμε μια "**διαδικασία-συναρτηση**" ή απλώς "**συνάρτηση**" (βλ. κανόνες 2, 14 και 75). Αυτό είναι ενδεδειγμένο, όταν στο πρόβλημα, για το οποίο ορίζουμε τη διαδικασία, η τιμή που υπολογίζεται είναι η τιμή κάποιας συνάρτησης.

Στο Σχήμα 1.7.6 δίδουμε τη διαδικασία-συνάρτηση που υπολογίζει την τιμή του αθροίσματος της σειράς $1 + x/1 + x2/2 + \ldots + xn/n$.

Παρατηρούμε ότι μεταξύ των τυπικών παραμέτρων της διαδικασίας (συναρτήσεως) δεν υπάρχει παράμετρος για την επιστροφή (μεταβίβαση) του αποτελέσματος του υπολογισμού.

Η επιστροφή γίνεται με το όνομα της συνάρτησης, για το οποίο δηλώνεται και τύπος, (βλ. κανόνες 2, 51, 77). Για τη συνάρτηση του Σχήματος 1.7.6 μπορούμε να γράψουμε, π.χ., τις εκφράσεις: A ← F(x, n), B ← (24*F(2.5, 6) + F(0.5, 8)/5)**3, κ.α.

Η αναφορά συνάρτησης F(x,n) μπορεί να χρησιμοποιηθεί όπως οποιαδήποτε

μεταβλητή, ή έκφραση, όπως φαίνεται και από τον κανόνα 51.

```
διαδικασια F(X, N) επιστρεφει (πραγματικη);
δηλωση X πραγματικη;
δηλωση N ακεραια;
αρχη
    δηλωση i ακεραια;
    δηλωση (S, T) πραγματικη;
    S ← 1;
    T ← 1;
    για i ← 1 εως N επαναλαβε
        (T ← T*X;
         S ← S + T/i);
    επιστροφη (S);
τελος F
```

ΣΧΗΜΑ 1.7.6 : Διαδικασία-συνάρτηση που υπολογίζει την τιμή του αθροίσματος N όρων της σειράς $1+x/1 + x^2/2 +\ldots + x^n/n$.

ΠΑΡΑΡΤΗΜΑ 2

ΠΡΩΤΗ ΓΝΩΡΙΜΙΑ ΜΕ ΤΟΝ ΠΡΟΣΩΠΙΚΟ Η/Υ - PC

Π2.1 Γενική Ανασκόπηση του Η/Υ

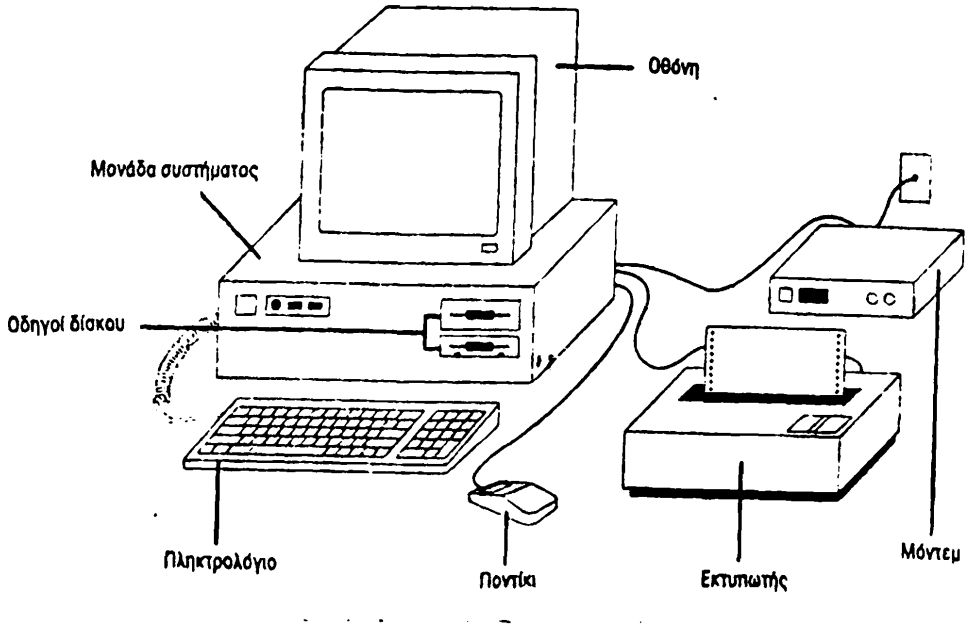
Το σύστημα του Η/Υ αποτελείται από δύο μέρη: **το υλικό ή μηχανικό** (hardware) και το **λογισμικό ή λογικό** (software).

Το **υλικό** αποτελείται από όλα τα φυσικά μέρη της μηχανής (μηχανικό). Πολλές φορές ορίζεται σαν "οτιδήποτε μπορείτε να κλωτσήσετε". Αν και ο ορισμός αυτός είναι άκομψος, εντούτοις δείχνει άμεσα ότι το υλικό του Η/Υ αποτελείται από τα φυσικά εξαρτήματά του. Στο παρόν όταν αναφερόμαστε σε Η/Υ εννοούμε **προσωπικό Η/Υ - PC** (Personal Computer).

Το **λογισμικό** είναι οτιδήποτε άλλο. Το λογισμικό αποτελείται από προγράμματα και δεδομένα που αλληλεπιδρούν με το υλικό. Η γλώσσα C++ είναι ένα παράδειγμα λογισμικού, η οποία μπορεί να χρησιμοποιηθεί για τη δημιουργία και άλλου λογισμικού και δεδομένων.

Π2.2 Το Υλικό

Ένα τυπικό σύστημα προσωπικού Η/Υ (PC) αποτελείται από τη **μονάδα συστήματος** που περιέχει την κεντρική μονάδα επεξεργασίας ή **μικροεπεξεργαστή** (central processing unit ή microprocessor), την **οθόνη** (monitor), τον **εκτυπωτή** (printer), το **πληκτρολόγιο** (keyboard), το **ποντίκι** (mouse) και το **μόντεμ** (modem) (βλ. Σχήμα 2.2.1).



ΣΧΗΜΑ 2.2.1 : Ένα τυπικό σύστημα προσωπικού Η/Υ (PC).

Είναι ευνόητο και επιβάλλεται πριν χρησιμοποιήσετε την C++ να έχετε μια γενική γνώση του τι είναι το υλικό και πως συνεργάζονται τα εξαρτήματά του.

Π2.2.1 Η Μονάδα Συστήματος και η Μνήμη

Η μονάδα συστήματος είναι ένα "μεγάλο κουτί". Είναι η μονάδα που περιέχει τον μικροεπεξεργαστή του προσωπικού Η/Υ (PC), ο οποίος, πολλές φορές, λέγεται και **Κεντρική Μονάδα Επεξεργασίας** ή **ΚΜΕ** (Central Processing Unit ή **CPU**). Η ΚΜΕ ενεργεί σαν τροχονόμος του Η/Υ, αφού κατευθύνει τη ροή των πληροφοριών μέσα στο σύστημα. Δηλαδή, η ΚΜΕ είναι κάτι ανάλογο με τον εγκέφαλο του ανθρώπου, κατά συνέπεια η χρησιμοποίηση ενός Η/Υ γίνεται δια της αλληλεπίδρασης με την ΚΜΕ του Η/Υ και μόνο.

Τα υπόλοιπα εξαρτήματα του υλικού χρησιμοποιούνται από την ΚΜΕ, αφενός για να στέλνει μηνύματα στον χρήστη μέσω της οθόνης και του εκτυπωτή, αφετέρου για να λαμβάνει μηνύματα από τον χρήστη μέσω του πληκτρολογίου ή του ποντικιού.

Η ΚΜΕ περιέχει την **"εσωτερική ή κύρια ή κεντρική" μνήμη** (internal ή main

memory) του Η/Υ. Αν και η μνήμη έχει πολλά ονόματα συνήθως αναφέρεται σαν **RAM**, **Μνήμη Τυχαίας Προσπέλασης** (Random Access Memory). Η RAM είναι το μέρος όπου η ΚΜΕ απευθύνεται για να βρεί λογισμικό και δεδομένα.

Για παράδειγμα προκειμένου να εκτελεστεί ένα πρόγραμμα σε C++, δίνεται εντολή στην ΚΜΕ του Η/Υ να ψάξει στην RAM για το πρόγραμμα και να εκτελέσει τις εντολές του. Η C++ χρησιμοποιεί την RAM όταν φορτώνεται.

Η RAM χρησιμοποείται για πολλές εργασίες του Η/Υ σας. Είναι ένα από τα σημαντικότερα εξαρτήματα του υλικού του, αφού προσφέρει τον χώρο για εντολές και δεδομένα αλλά και επηρεάζει την ταχύτητά του. Γενικά, όσο περισσότερη RAM διαθέτει ο Η/Υ, τόσο περισσότερη εργασία μπορεί να κάνει και τόσο ταχύτερα μπορεί να επεξεργαστεί τα δεδομένα.

Η **χωρητικότητα** της RAM μετράται από τον αριθμό των χαρακτήρων που μπορεί να αποθηκεύει. Για παράδειγμα υποθέτουμε ότι, ένας προσωπικος Η/Υ αποθηκεύει περίπου 640,000 χαρακτήρες στην RAM. Ένας χαρακτήρας στην ορολογία του Η/Υ καλείται **ψηφιολέξη** (byte), όπου μια ψηφιολέξη μπορεί να είναι ένα γράμμα, ένας αριθμός ή ένας ειδικός χαρακτήρας, π.χ. ερωτηματικό κ.λ.π.. Αν για παράδειγμα ο Η/Υ σας έχει 640,000 ψηφιολέξεις (bytes) RAM, τότε μπορεί να περιέχει 640,000 χαρακτήρες.

Το ερώτημα είναι τι γίνεται με τα μηδενικά στην τιμή της RAM, που ίσως και να περιπλέκουν τα πράγματα. Συνήθως βλέπουμε μια συντετμημένη γραφή k (kilo = 1,000) αντί για τρία μηδενικά στο τέλος του αριθμού που δηλώνει χωρητικότητα. Το k για την ακρίβεια σημαίνει 1024 ψηφιολέξεις (bytes), αριθμός που συνήθως στρογγυλεύεται στα 1,000. Έτσι το 640k παριστά περίπου 640,000[1] ψηφιολέξεις (bytes) μνήμης RAM. Άλλες

[1] Η Δύναμη του 2 : Οι Η/Υ λειτουργούν χρησιμοποιώντας ηλεκτρικές καταστάσεις on (άναμμα) και off (σβήσιμο). Αυτές καλούνται δυαδικές καταστάσεις. Στο επίπεδο μηχανής του Η/Υ, αυτό σημαίνει άναμμα και σβήσιμο ηλεκτρισμού με εκατομμύρια διακοπτών που καλούνται τρανζίστορ . Επειδή οι διακόπτες αυτοί έχουν δύο καταστάσεις, ο συνολικός αριθμός των ηλεκτρικών καταστάσεων, εκφράζονται με μια δύναμη του 2.

Η πλησιέστερη προς το 1,000 δύναμη του 2 είναι το 1,024 (= 2^{10}) Οι επινοητές των Η/Υ σχεδίασαν τη μνήμη έτσι ώστε να προστίθενται πάντα kilobytes ή πολλαπλάσια του 1,024 bytes Έτσι, εάν προστεθούν 128k μνήμης RAM σε έναν Η/Υ, στην πραγματικότητα προστίθενται 131,072 bytes RAM δηλαδή:

$$128 \times 1{,}024 = 131{,}072.$$

Επειδή το k > 1,000 στην πραγματικότητα έχετε πάντα λίγο περισσότερη μνήμη, από την ονομαστική. Για παράδειγμα, εάν ο Η/Υ θεωρείται των 640k ουσιαστικά περιέχει περισσότερα των 640,000 bytes, για την ακρίβεια περιέχει 655,360 bytes.

συντμήσεις μέτρησης της μνήμης είναι το **meg** ή **megabytes** που συμβολίζεται με M, ισχύει 1M = 1,048,576 bytes ≅ 1,000,000 bytes, και το **giga** ή **gigabytes** για το ισχύει 1giga ≅ 10^9 bytes.

Οι περιορισμοί της RAM είναι παρόμοιοι με τους περιορισμούς των κασετών μαγνητοφώνου. Εάν μια κασέτα κατασκευάστηκε για να αποθηκεύει μουσική 60', δεν μπορεί να αποθηκεύσει μουσική 75'. Με ανάλογο τρόπο, ο συνολικός αριθμός χαρακτήρων που συνθέτουν ένα πρόγραμμα, π.χ. τα δεδομένα της'C++ και τα προγράμματα του συστήματος δεν μπορεί να υπερβαίνουν το όριο της χωρητικότητας της RAM, εκτός και αν μερικοί από τους χαρακτήρες βρίσκονται σε δίσκο (βλπ §Π2.2.2).

Χρειάζεστε όσο το δυνατόν περισσότερη RAM για την αποθήκευση της C++, δηλαδή των δεδομένων και των προγραμμάτων του συστήματός της. Γενικά, τα 640k είναι αρκετός χώρος για οτιδήποτε θέλετε να κάνετε στην C++.

Η RAM του Η/Υ είναι σχετικά φθηνή, εάν ο Η/Υ σας έχει λιγότερο από 640 bytes μνήμης επιβάλλεται να την αυξήσετε σε 640k. Στους περισσότερους προσωπικούς Η/Υ μπορείτε να βάλετε πάνω από 640k.

Υπάρχουν δύο τύποι πρόσθετης μνήμης RAM, η **επεκταμένη μνήμη** και η **εκτεταμένη μνήμη** και οι δύο προσφέρουν δυνατότητες αποθήκευσης πέραν των 640k.

Μπορείτε να προσπελάσετε αυτή την πρόσθετη RAM με ορισμένα συστήματα C++, αλλά οι περισσότεροι αρχάριοι προγραμματιστές της C++ δεν χρειάζονται να ασχοληθούν με RAM πέραν των 640k.

Ο Η/Υ αποθηκεύει τα προγράμματα της C++ στην RAM, την ώρα που τα γράφετε. Εάν έχετε χρησιμοποιήσει έναν επεξεργαστή κειμένου, π.χ. τον Editor, τότε χρησιμοποιήσατε RAM. Όταν πληκτρολογείτε λέξεις στα έγγραφά σας, οι λέξεις αυτές εμφανίζονται στην οθόνη και επίσης αποθηκεύονται στην RAM.

Παρά τη σημασία της, η RAM είναι μόνο ένας τύπος μνήμης του Η/Υ. Η RAM είναι φθαρτή-προσωρινή, δηλαδή, όταν σβήνετε τον Η/Υ όλο το περιεχόμενο της RAM διαγράφεται.

Έτσι, πρέπει να αποθηκεύετε τα περιεχόμενα της RAM σε μια **μόνιμη μνήμη**, π.χ. δίσκο, πριν σβήσετε τον Η/Υ, διαφορετικά θα έχετε χάσει όλη τη δουλειά σας.

Π2.2.2 Η Αποθήκευση σε Δίσκο

Ο **δίσκος** (disk) είναι ένας άλλος τύπος μνήμης Η/Υ, που λέγεται και **εξωτερική μνήμη** (external memory). Η αποθήκη δίσκου είναι άφθαρτη-μόνιμη, με την έννοια ότι όταν σβήνετε τον Η/Υ τα περιεχόμενα του δίσκου δεν χάνονται.

Αφού λοιπόν πληκτρολογήσετε ένα μεγάλο πρόγραμμα C++ στην RAM, φυσικά δεν θα θέλατε να το ξαναπληκτρολογείτε κάθε φορά που ανάβετε τον Η/Υ σας, έτσι, το αποθηκεύετε σε δίσκο, όπου παραμένει μέχρι να το ανακαλέσετε.

Τα περιεχόμενα του δίσκου δεν μπορούν να τύχουν επεξεργασίας από την ΚΜΕ. Εάν έχετε ένα πρόγραμμα ή δεδομένα στον δίσκο, που θέλετε να τα χρησιμοποιήσετε, τότε πρέπει να τα μεταφέρετε από τον δίσκο στην RAM. Αυτός είναι ο μόνος τρόπος για να μπορέσει να εργαστεί η ΚΜΕ με αυτά. Σημειώνουμε ότι η διαδικασία αυτή της μεταφοράς είναι απλούστατη αλλά και απαραίτητη. Εάν πάλι γεμίσει η RAM, μπορείτε να αποθηκεύσετε τα δεδομένα στον δίσκο και να συνεχίσετε. Συνήθως ένα πρόγραμμα C++ εκτελείται στην RAM και ανακαλεί δεδομένα από τον δίσκο όποτε τα χρειάζεται, όμως αυτό είναι θέμα παραπέρα ύλης και μελέτης.

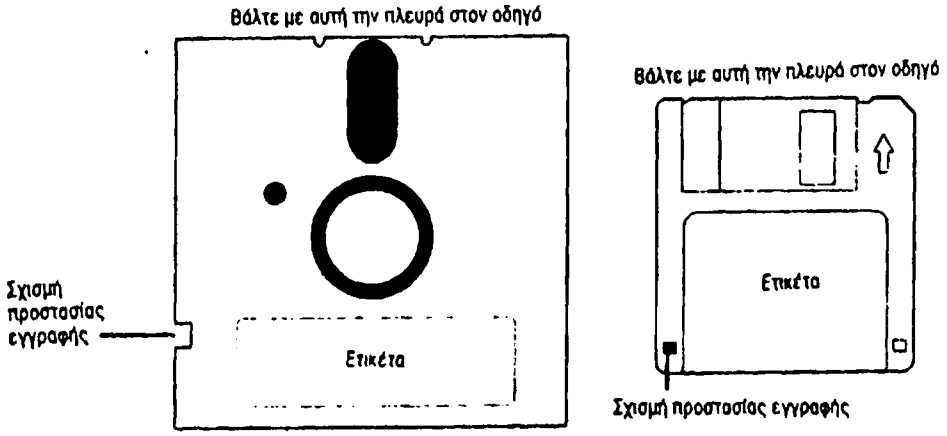
Υπάρχουν δύο τύποι δίσκων, οι **σκληροί** ή **σταθεροί δίσκοι** (hard disk) και οι **ελαστικοί δίσκοι** ή **δισκέττες** (disquettes) (βλπ Σχήματα 2.2.1 και 2.2.2).

Οι σκληροί δίσκοι αποθηκεύουν περισσότερα δεδομένα και είναι ταχύτεροι από τους ελαστικούς. Τα περισσότερα από τα προγράμματα και δεδομένα της C++ πρέπει να αποθηκεύονται σε σκληρούς δίσκους. Οι ελαστικοί δίσκοι είναι χρήσιμοι για εφεδρεία των σκληρών δίσκων και για μεταφορά δεδομένων και προγραμμάτων από έναν υπολογιστή σε άλλον.

Στο Σχήμα 2.2.2 υπάρχουν δύο τύποι δισκεττών συνήθων μεγεθών, η δισκέττα των 5 1/4 ιντσών και αυτή των 3 1/2 ιντσών. Οι δισκέττες αποθηκεύουν από 360k μέχρι 1.4

εκατομμύρια bytes δεδομένων.

Πριν χρησιμοποιήσετε ένα νέο κουτί δισκεττών, πρέπει να τις μορφοποιήσετε (formate), εκτός και εάν τις έχετε αγοράσει ήδη μορφοποιημένες. Η μορφοποίηση προετοιμάζει τις δισκέττες για χρήση στον Η/Υ, γράφοντας μια διάταξη διαδρομών, που καλούνται ίχνη, όπου θα αποθηκεύονται και προγράμματα. Για τη σωστή διαδικασία μορφοποίησης πρέπει ο ενδιαφερόμενος αρχάριος να διαβάσει το εγχειρίδιο του λειτουργικού συστήματος.



ΣΧΗΜΑ 2.2.2 : Ελαστικοί δίσκοι ή δισκέττες των 5 1/4 ιντσών και των 3 1/2 ιντσών.

Οι δισκέττες μπαίνουν και βγαίνουν χειρωνακτικά σε οδηγούς δισκεττών. Συνήθως οι οδηγοί δισκεττών βρίσκονται μέσα στη μονάδα συστήματος.

Ο σκληρός δίσκος είναι σφραγισμένος μέσα στον οδηγό του σκληρού δίσκου και δεν τον βγάζετε ποτέ παρά μόνο για επισκευή.

Οι οδηγοί δίσκων έχουν ονόματα. Ο πρώτος οδηγός δισκέττας του Η/Υ ονομάζεται Α. Ο δεύτερος, αν υπάρχει, ονομάζεται Β. Ο πρώτος σκληρός δίσκος, πολλοί υπολογιστές έχουν μόνο έναν, ονομάζεται οδηγός C. Εάν έχετε και άλλους σκληρούς δίσκους ή εάν ο σκληρός σας δίσκος είναι λογικά διαιρεμένος σε περισσότερους του ενός δίσκους, ονομάζονται D, E κ.λ.π.

Το μέγεθος του δίσκου μετράται σε bytes, megabytes και gigabytes όπως και η RAM. Οι δίσκοι μπορούν να αποθηκεύσουν πολλά εκατομμύρια bytes δεδομένων. Ο σκληρός δίσκος 60M είναι πλέον ξεπερασμένος, μπορεί να αποθηκεύσει περίπου 60 εκατομμύρια χαρακτήρες δεδομένων, πριν γεμίσει. Οι σύγχρονοι δίσκοι είναι της τάξης του giga.

Π2.2.3 Η Οθόνη

Η οθόνη του Η/Υ, που μοιάζει με αυτή της τηλεόρασης, καλείται απλώς **οθόνη** (monitor) ή CRT (Cathod Ray Tube), από το "λυχνία καθοδικών ακτίνων" το κύριο εξάρτημα της οθόνης. Η οθόνη είναι ένα μέσον όπου μπορεί να σταλεί η έξοδος από τον Η/Υ. Όταν για παράδειγμα θέλετε να δείτε τον κατάλογο ονομάτων και διευθύνσεων, μπορείτε να γράψετε ένα πρόγραμμα σε C++, που θα εμφανίζει αυτές τις πληροφορίες στην οθόνη.

Το πλεονέκτημα της εξόδου στην οθόνη έναντι της εκτύπωσης, είναι ότι η πρώτη είναι ταχύτερη και εξοικονομεί χαρτί. Τα μειονεκτήματά της όμως, είναι ότι αφενός δεν είναι μόνιμη, αφετέρου είναι η ταχύτητα με την οποία τα εξερχόμενα στην οθόνη χάνονται, καθώς άλλα εισέρχονται και εμφανίζονται, χωρίς να προλαβαίνετε να τα δείτε.

Όλες οι οθόνες έχουν έναν δρομέα - φωτεινό χαρακτήρα (cursor), ο οποίος κινείται όταν πληκτρολογείτε γράμματα στην οθόνη, ο οποίος πάντα δηλώνει τη θέση του επόμενου χαρακτήρα που θα πληκτρολογείσετε.

Οι οθόνες που μπορούν που μπορούν να εμφανίσουν εικόνες, καλούνται **οθόνες γραφικών**. Υπάρχουν οθόνες, που δεν έχουν ακόμα αποσυρθεί, οι οποίες μπορούν να εμφανίσουν μόνο κείμενο. Εάν η οθόνη σας δεν είναι έγχρωμη λέγεται μονόχρωμη.

Η οθόνη συνδέεται σε έναν προσαρμοστή οθόνης που βρίσκεται μέσα στη μονάδα συστήματος. Ο προσαρμοστής καθορίζει το ποσό της ανάλυσης και τον αριθμό των πιθανών χρωμάτων επί της οθόνης. Η ανάλυση αναφέρεται στον αριθμό των γραμμών και στηλών της οθόνης και το πόσο οξύτερα - καλοσχεδιασμένα εμφανίζονται το κείμενο και τα γραφικά συην οθόνη. Ορισμένοι προσαρμοστές οθόνης είναι οι MCGA, CGA, EGA και VGA.

Π2.2.4 Ο Εκτυπωτής

Ο εκτυπωτής προσφέρει έναν πιό μόνιμο τρόπο καταγραφής των αποτελεσμάτων από τον Η/Υ. Είναι η "γραφομηχανή" του Η/Υ σας. Μπορεί να εκτυπώσει έξοδο απο προγράμματα της C++ σε χαρτί και από οτιδήποτε εμφανίζεται στην οθόνη, επίσης χρησιμοποιείται για να εκτυπώνει φακέλους και επιταγές.

Οι δύο συνηθέστεροι τύποι εκτυπωτών για Η/Υ (PC) είναι ο **εκτυπωτής ακίδων** και ο **εκτυπωτής laser**. Οι εκτυπωτές ακίδων είναι φθηνοί έχουν ταχύτητα και χρησιμοποιούν μια σειρά μικρών τελειών για να σχεδιάσουν το κείμενο και τα γραφικά στο χαρτί. Οι εκτυπωτές laser είναι ταχύτεροι και η έξοδός τους είναι καλλίτερη, επειδή η ακτίνα laser καίει τον γραφίτη πάνω στο χαρτί, είναι όμως και δαπανηρότεροι. Για πολλούς ο εκτυπωτής ακίδων παρέχει όλη την ταχύτητα και ποιότητα που χρειάζονται για τις περισσότερες εφαρμογές. Η C++μπορεί να στείλει έξοδο και στους δύο τύπους εκτυπωτών.

Π2.2.5 Το Πληκτρολόγιο

Στο Σχήμα 2.2.3 είναι ένα χαρακτηριστικό πληκτρολόγιο προσωπικού Η/Υ (PC). Τα πλήκτρα με τα γράμματα και με τους αριθμούς στο κέντρο του πληκτρολογίου εμφανίζουν στην οθόνη ό,τι γράφουν.

Απευθυνόμενοι στους αρχάριους πρωτοετείς, οι οποίοι δικαιολογημένα αντιμετωπίζουν τα των Η/Υ με δέος, παρουσιάζουμε στοιχειώδεις πληροφορίες για τη λειτουργία και χρήση του πληκτρολογίου. Έτσι τολμήστε και ξεκινήστε να πειραματίζεστε.

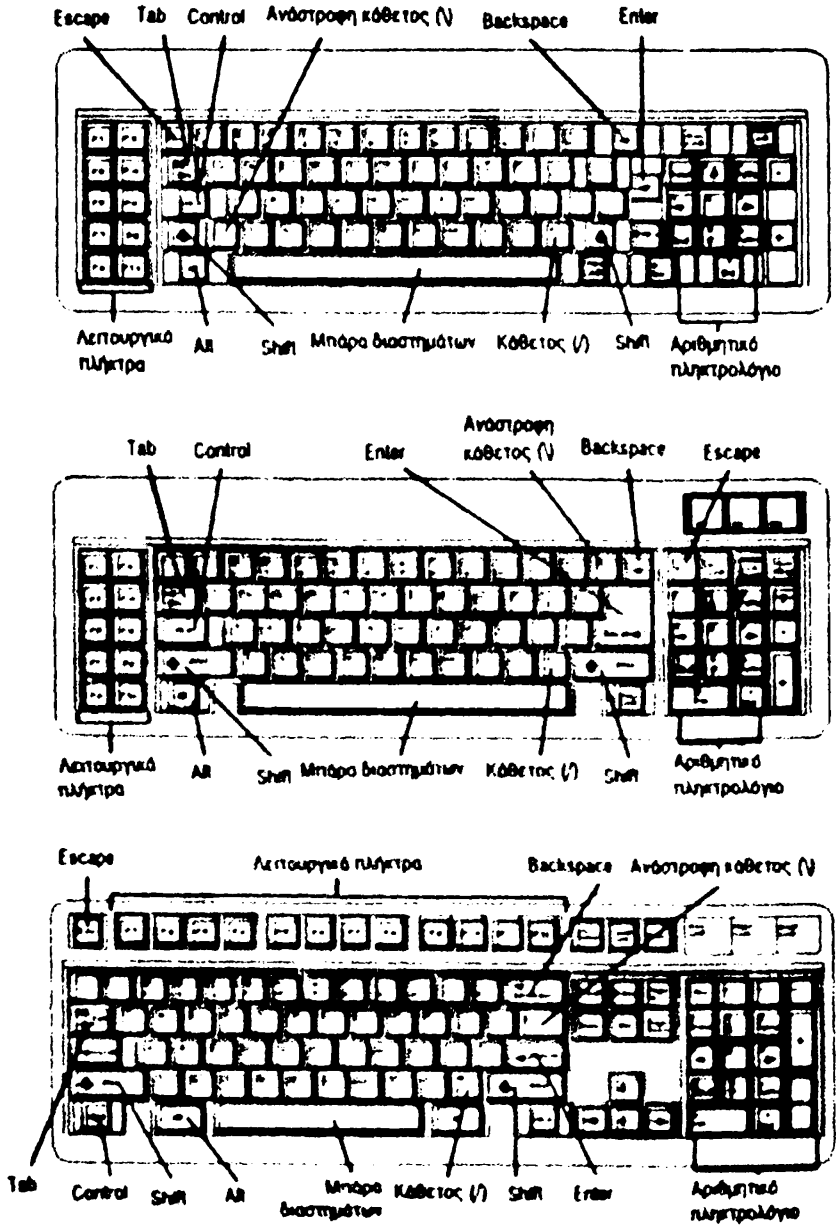
• Για να πληκτρολογήσετε ένα κεφαλαίο γράμμα, πιέστε ένα από τα δύο **shift** καθώς πληκτρολογήτε το γράμμα.

• Η πίεση του πλήκτρου **Capslock** (κλείδωμα κεφαλαίων) κάνει όλα τα γράμματα να γράφονται κεφαλαία.

• Αν θέλετε να πληκτρολογήσετε έναν από τους ειδικούς χαρακτήρες πάνω από τους αριθμούς, πιέστε συγχρόνως και το πλήκτρο **Shift** (μετατόπιση).

• Όπως το πλήκτρο **Shift**, έτσι και τα πλήκτρα **Alt** και **Ctrl** (control = ελέγχω) μπορούν να χρησιμοποιηθούν με ορισμένα άλλα πλήκτρα.

Για παράδειγμα στην C++, ο συνδυασμός των πλήκτρων **Alt-F** οδηγεί στην ενεργοποίηση της επιλογής (menu) **File**.



ΣΧΗΜΑ 2.2.3 : Τα διάφορα πληκτρολόγια προσωπικών Η/Υ (PC).

• Το πλήκτρο **Esc** (escape – διαφεύγω) στην πράξη ακυρώνει μια επιλογή ή οτιδήποτε άρχισε και θέλετε να το σταματήσετε.

Για παράδειγμα στην C++, αν ζητήσετε βοήθεια από τον μεταγλωττιστή και στη συνέχεια δεν χρειάζεστε το μήνυμα βοήθειας, πατώντας το **Esc** απαλείφετε το μήνυμα από την οθόνη.

• Η ομάδα αριθμών και βελών στο δεξί άκρο του πληκτρολογίου λέγεται **αριθμητικό πληκτρολόγιο**. Το αριθμητικό πληκτρολόγιο ενεργοποιείται με το πλήκτρο **NumLock** (κλείδωμα αριθμών), το οποίο πατώντας το ξανά απενεργοποιεί το αριθμητικό πληκτρολόγιο και θέτει στη διάθεσή σας τα **πλήκτρα των βελών**. Πολλά πληκτρολόγια έχουν ξεχωριστά τα πλήκτρα των βελών από το αριθμητικό πληκτρολόγιο.

• Τα **βέλη** βοηθούν στο να κινήσετε τον δρομέα από μια περιοχή της οθόνης στην άλλη. Για να μεταφέρετε τον δρομέα προς την κορυφή της οθόνης, πιέζετε συνεχώς το **βέλος που έχει κατεύθυνση προς τα πάνω**. Για να πάτε δεξιά, πιέζετε το **βέλος που έχει κατεύθυνση προς τα δεξιά**, κ.ο.κ.

Δεν θα πρέπει να συγχέετε το πλήκτρο **Backspace** (προς τα πίσω διάστημα) με το βέλος που έχει κατεύθυνση προς τα αριστερά. Το πλήκτρο **Backspace** μεταφέρει τον δρομέα προς τα πίσω κατά ένα χαρακτήρα, διαγράφοντας τον όπως προχωρεί. Το βέλος που έχει κατεύθυνση προς τα αριστερά απλώς μεταφέρει τον δρομέα προς τα πίσω, χωρίς να διαγράφει.

• Τα πλήκτρα **Insert** ή **Ins** παρεμβάλλω) και **Delete** ή **Del** (διαγράφω) είναι χρήσιμα για επεξεργασία.

Για παράδειγμα στην C++, χρησιμοποιούνται τα πλήκτρα αυτά όπως και στην επεξεργασία κειμένου πατώντας το **Ins** όπου ο δρομέας βρίσκεται, έχετε τη δυνατότητα να παρεμβάλλετε χαρακτήρα ή λέξη ή πρόταση. Ανάλογα πατώντας το **Del** κάνουμε διαγραφές χαρακτήρων ή λέξεων ή προτάσεων.

• Τα πλήκτρα **PgUp** (page up – σελίδα πάνω) και **PgDn** (page down – σελίδα κάτω) μεταφέρουν το κείμενο στην οθόνη κατά μια οθόνη πάνω ή κάτω αντίστοιχα, χρησιμοποιώντας, ίσως, το **NumLock**.

• Τα πλήκτρα **F1**έως **F12**, ορισμένα πληκτρολόγια έχουν μόνο μέχρι **F10**, λέγονται **λειτουργικά πλήκτρα**. Τα πλήκτρα αυτά βρίσκονται είτε στη κορυφή του πληκτρολογίου είτε στα αριστερά του και εκτελούν προχωρημένες λειτουργίες, συγκεκριμένες για κάθε εφαρμογή. Συνήθως οι εφαρμογές που χρησιμοποιούν τα πλήκτρα αυτά ενημερώνουν τον χρήστη, εμφανίζοντας τις σχετικές πληροφορίες στο κάτω μέρος της οθόνης.

• **Προσοχή !!!** Στην πληκτρολόγηση μη συγχέετε το πλήκτρο που αντιστοιχεί στο 1 (ένα) με αυτό που αντιστοιχεί στο l (ελ) ή αυτό που αντιστοιχεί στο 0 (μηδέν) με αυτό που αντιστοιχεί στο ο ή Ο (όμικρον).

Π2.2.6 Το Ποντίκι

Το ποντίκι μεταφέρει τον δρομέα σε κάθε θέση της οθόνης. Ο επεξεργαστής της C++ είναι δυνατόν να χρησιμοποιεί το ποντίκι για επιλογή εντολών από τα παράθυρα των επιλογών (menu). Δηλαδή το ποντίκι υποκαθιστά την απλή ή τη συνδυασμένη χρήση κάποιων πλήκτρων για ενεργοποίηση απλών ή προχωρημένων λειτουργιών.

Τα ποντίκια έχουν δύο ή τρία κουμπιά. Τις περισσότερες φορές η πίεση του τρίτου κουμπιού έχει το ίδιο αποτέλεσμα με την ταυτόχρονη πίεση και των δύο κουμπιών σε ποντίκι με δύο κουμπιά.

Π2.2.7 Το Μόντεμ

Το μόντεμ (**mo**dem) είναι η συσκευή μέσω της οποίας τα σήματα του Η/Υ διαμορφώνονται (**mo**dulate) και αποδιαμορφώνονται (**dem**odulate) από ψηφιακά[1] σε αναλογικά και το αντίστροφο. Όπως γίνεται αντιληπτό η λέξη αγγλική modem προέκυψε από τα ακρωνύμια των αγγλικών λέξεων **mo**dulate και **dem**odulate.

Το μόντεμ χρησιμοποιείται για επικοινωνία ανάμεσα σε δύο ή περισσότερους

[1] Ο όρος ψηφιακός Η/Υ προέρχεται από το γεγονός ότι ο Η/Υ λειτουργεί με δυαδικούς (on και off) ηλεκτρικούς παλμούς . Αυτές οι ψηφιακές καταστάσεις είναι ιδανικές για τον Η/Υ αλλά δεν μπορούν να σταλούν μέσω κανονικών τηλεφωνικών γραμμών. Τα τηλεφωνικά σήματα καλούνται αναλογικά σήματα, τα οποία διαφέρουν κατά πολύ από τα ψηφιακά σήματα του Η/Υ.

απομακρυσμένους Η/Υ. Έτσι πριν στείλει ο Η/Υ σας τα δεδομένα μέσω τηλεφωνικής γραμμής, οι πληροφορίες πρέπει να διαμορφωθούν σε αναλογικά σήματα μέσω του μόντεμ του αποστολέα. Ο παραλήπτης Η/Υ πρέπει μέσω του μόντεμ να αποδιαμορφώσει τα σήματα αυτά σε ψηφιακά.

Υπάρχουν μόντεμ εξωτερικά τα οποία λειτουργούν ως συσκευές εκτός του Η/Υ και μόντεμ εσωτερικά, ενσωματωμένα στη μονάδα συστήματος του Η/Υ.

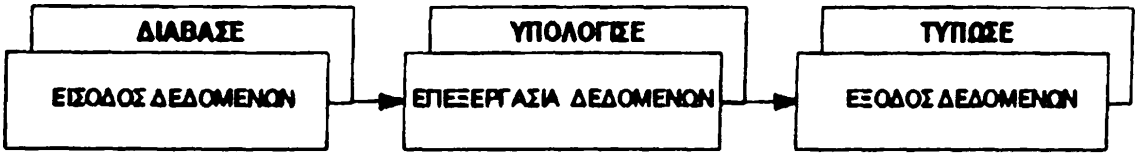
Τέλος μπορείτε να γράψετε προγράμματά της C++ που επικοινωνούν με το μόντεμ σας.

Π2.3 Το Λογισμικό

Ανεξάρτητα από το πόσο ταχύ, μεγάλο και δυνατό είναι το υλικό του Η/Υ σας, το λογισμικό καθορίζει ποιά εργασία γίνεται και πως την εκτελεί ο Η/Υ. Αποθηκεύετε το λογισμικό στον δίσκο του Η/Υ και το φορτώνεται στη μνήμη του, ώστε όταν είστε έτοιμοι να το επεξεργαστείτε.

Π2.3.1 Τα Προγράμματα και τα Δεδομένα

Οι Η/Υ στην πραγματικότητα κάνουν επεξεργασία δεδομένων, δηλαδή διαχειρίζονται τα εισαγόμενα δεδομένα και παράγουν έξοδο, η οποία λέγεται πληροφορία.



ΣΧΗΜΑ 2.2.4 : Επεξεργασία δεδομένων στη βασικότερη μορφή της.

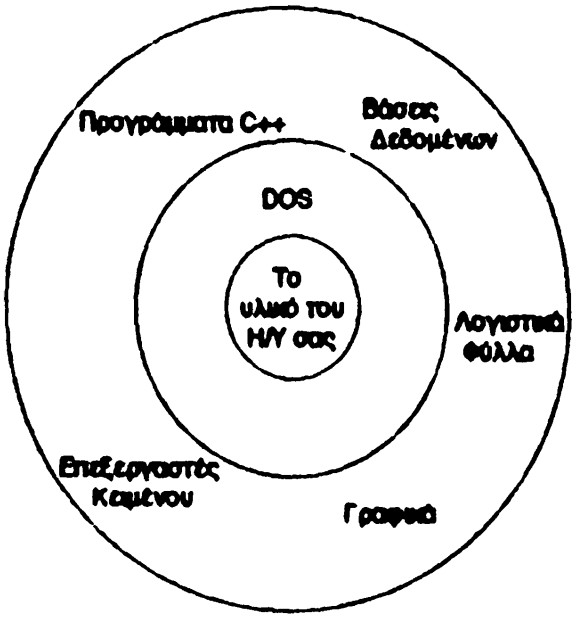
Στο Σχήμα 2.2.4 παρατίθεται το μοντέλο εισόδου - επεξεργασίας - εξόδου (πρβλ διάβασε - υπολόγισε - τύπωσε §1.2 & §2.3), που είναι η βάση για την κατανόηση και κατά συνέπεια χρήση - αξιοποίηση του Η/Υ.

Τα προγράμματα σε C++ που γράφετε και τα δεδομένα που επεξεργάζονται αποτελούν λογισμικό. Το υλικό δρά σαν όχημα, για συλλογή της εισόδου και παραγωγής

της εξόδου. Για παράδειγμα ένα πρόβλημα μισθοδοσίας σε C++ μπορεί να δέχεται δεδομένα εισόδου τις ώρες εργασίας από το πληκτρολόγιο. Κατόπιν δίνει εντολή στην ΚΜΕ να υπολογίσει τα ποσά μισθοδοσίας για κάθε υπάλληλο που είναι καταχωρημένος στο αρχείο που είναι αποθηκευμένο στον δίσκο. Μετά την επεξεργασία της μισθοδοσίας, το πρόγραμμα μπορεί να εκτυπώνει τις επιταγές.

Π2.3.2 Το Λειτουργικό Σύστημα MS-DOS

Το MS-DOS, το λειτουργικό σύστημα (δίσκου) της Microsoft, επιτρέπει στα προγάμματα της C++ να αλληλεπιδρούν με το υλικό. Το MS-DOS ή DOS, όπως συνήθως λέγεται, το βρίσκετε πάντοτοτε φορτωμένο στη RAM, όταν ανάβετε τον Η/Υ σας. Το DOSδεν ελέγχει μόνο τους δίσκους. Χρησιμοποιείται ώστε τα προγράμματά σας να επικοινωνούν με όλο το υλικό του Η/Υ, συμπεριλαμβανομένων της οθόνης, του πληκτρολογίου και του εκτυπωτή.

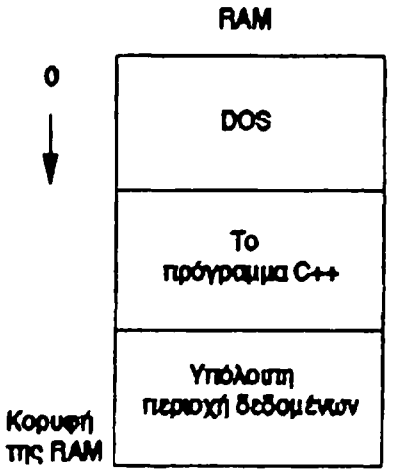


ΣΧΗΜΑ 2.2.5 : Το DOS κάνει τη ζεύξη ανάμεσα στο υλικό και το λογισμικό

Το Σχήμα 2.2.5 δείχνει το DOS σαν ενδιάμεσο κρίκο υλικού και λογισμικού του Η/Υ. Το DOS είναι σχεδιασμένο για να μπορεί να ελέγχει κάθε μηχάνημα συνδεδεμένο με τον Η/Υ, γιαυτό χρησιμοποιεί λίγο χώρο και παραμένει στη RAM, με την έννοια ότι φορτώνεται αυτόματα με την εκκίνηση του Η/Υ, αναμένοντας εντολές. Σε καμμιά περίπτωση δεν πρέπει να ανησυχείτε για τη μεταφορά του DOS στη RAM. Για παράδειγμα, εάν θέλετε να εκτυπώσετε ένα

μήνυμα στον Η/Υ, χρειάζονται πολλές εντολές του Η/Υ προς τον εκτυπωτή. Το DOS, όπως προαναφέραμε, είναι σχεδιασμένο για να στέλνει τις κατάλληλες πληροφορίες και να ενεργοποεί τον εκτυπωτή, όποτε φυσικά, πάρει ανάλογες εντολές από το πρόγραμμά σας, ενεργοποιώντας τη δρομολόγηση των δεδομένων στον εκτυπωτή.

Το Σχήμα 2.2.6 δείχνει τη θέση του DOS, της C++ και της περιοχής δεδομένων της C++ στην RAM. Αυτός ο σχηματισμός είναι ο συνήθης τρόπος αναπαράστασης της RAM, "θυρίδες μνήμης" τοποθετημένες η μια πάνω στην,άλλη. Κάθε θέση μνήμης, κάθε ψηφιολέξη (byte) έχει μια μοναδική διεύθυνση. Η πρώτη διεύθυνση στη μνήμη αρχίζει από το 0, η δεύτερη είναι 1, κ.λ.π., μέχρι την τελευταία διεύθυνση στη RAM, αρκετές χιλιάδες bytes παρακάτω.



ΣΧΗΜΑ 2.2.6 : Μετά το MS-DOS και ένα πρόγραμμα C++, υπάρχει φυσικά λιγότερος χώρος RAM για δεδομένα.

Το λειτουργικό σας σύστημα (MS-DOS, PC DOS, DR DOS ή UNIX) χρησιμοποιεί τις πρώτες εκατοντάδες bytes στη μνήμη. Το ποσό της RAM που καταλαμβάνει το DOS ποικίλει ανάλογα με τον Η/Υ. Όταν δουλεύετε στη C++, το σύστημα C++ κάθεται πάνω στο DOS, αφήνοντάς σας το υπόλοιπο της RAM για τα προγράμματα και τα δεδομένα σας. Αυτός είναι ο λόγος ο οποίος, ενώ μπορεί να έχετε σύνολο 512K RAM, να μην έχετε χώρο για εκτέλεση ορισμένων προγραμμάτων σας, το DOS χρησιμοποιεί μέρος της RAM για τον εαυτό του.

Συμβαίνει πολλές φορές άνθρωποι που προγραμματίζουν για χρόνια να μην έχουν

ασχοληθεί ιδιαίτερα με την εκμάθηση του DOS. Δεν χρειάζεται να είναι κάποιος ειδικός στο DOS, ή ακόμα να γνωρίζει περισσότερες από "μερικές απλές εντολές" για να προγραμματίζει ή να χειρίζεται "καλά" (ικανοποιητικά) τον Η/Υ. Παρόλα αυτά το DOS κάνει ορισμένα πράγματα που δεν μπορεί να κάνει η C++, π.χ. μπρφοποίηση δίσκων και αντιγραφές αρχείων σε δίσκους κ.λ.π. Έχουμε επισημάνει ξανά ότι, οι αυξανόμενες ανάγκες, εν γένει των χρηστών, εξυπηρετήσεώς τους δια του Η/Υ, τους δημιουργούν την επιτακτική ανάγκη για καλλίτερη κατανόηση και πληρέστερη εκμάθηση όλο και περισσότερων πραγμάτων που έχουν σχέση με τον Η/Υ και τη δουλειά τους. Στην πραγματικότητα οι ανάγκες της δουλειάς μας, μας οδηγούν στο τι πρέπει να γνωρίζουμε και καθορίζουν τους τομείς εξοικείωσής μας για να είμαστε αποτελεσματικοί. Έτσι, για την εισαγωγή στη χρήση και στη γρήγορη εκμάθηση του DOS ή οτιδήποτε άλλου σχετικού (π.χ. WINDOWS), συνιστώνται πάντοτε καλά και πρακτικά εγχειρίδια, δηλαδή, ενημερωμένα, απλά, περιεκτικά και συνοπτικά, επίσης τόλμη, υπομονή, επιμονή και πολύς πειραματισμός.

ΚΑΛΗ ΕΠΙΤΥΧΙΑ ! ! !

Τυπώθηκε στο Πανεπιστημιακό Τυπογραφείο με δαπάνη
του Πανεπιστημίου Ιωαννίνων

Κ.Α. Πανεπιστημιακού Τυπογραφείου _____

ΔΙΑΝΕΜΕΤΑΙ ΔΩΡΕΑΝ στους φοιτητές