Τ.Ε.Ι ΗΠΕΙΡΟΥ

ΣΧΟΛΗ: ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ

ΤΜΗΜΑ: ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΓΡΑΦΙΚΗ ΔΙΑΠΡΟΣΩΠΕΙΑ ΓΙΑ ΚΑΤΑΣΚΕΥΑΖΟΜΕΝΑ ΤΕΧΝΗΤΑ ΝΕΥΡΩΝΙΚΑ ΔΙΚΤΥΑ

Μιχαήλ Ζορμπάς

Επιβλέπων: Ιωάννης Τσούλος,

Επίκουρος Καθηγητής

ΤΕΙ Ηπείρου

Άρτα, Νοέμβριος, 2018

Τ.Ε.Ι ΗΠΕΙΡΟΥ

ΣΧΟΛΗ: ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ

ΤΜΗΜΑ: ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΓΡΑΦΙΚΗ ΔΙΑΠΡΟΣΩΠΕΙΑ ΓΙΑ ΚΑΤΑΣΚΕΥΑΖΟΜΕΝΑ ΤΕΧΝΗΤΑ ΝΕΥΡΩΝΙΚΑ ΔΙΚΤΥΑ

Μιχαήλ Ζορμπάς

Επιβλέπων: Ιωάννης Τσούλος,

Επίκουρος Καθηγητής

ΤΕΙ Ηπείρου

Άρτα, Νοέμβριος, 2018

Graphic interface for constructed artificial neural networks

Εγκρίθηκε από τριμελή εξεταστική επιτροπή

Άρτα, Νοέμβριος, 2018

ΕΠΙΤΡΟΠΗ ΑΞΙΟΛΟΓΗΣΗΣ

1. Επιβλέπων καθηγητής
Ιωάννης Τσούλος,
Επίκουρος Καθηγητής ΤΕΙ Ηπείρου

2. Μέλος επιτροπής
Όνομα Επίθετο,
τίτλος, βαθμίδα

3. Μέλος επιτροπής
Όνομα Επίθετο,
τίτλος, βαθμίδα

Ο/Η Προϊστάμενος/η του Τμήματος
Όνομα Επίθετο,
τίτλος, βαθμίδα

Υπογραφή

Δήλωση μη λογοκλοπής

Δηλώνω υπεύθυνα και γνωρίζοντας τις κυρώσεις του Ν. 2121/1993 περί Πνευματικής Ιδιοκτησίας, ότι η παρούσα μεταπτυχιακή εργασία είναι εξ ολοκλήρου αποτέλεσμα δικής μου ερευνητικής εργασίας, δεν αποτελεί προϊόν αντιγραφής ούτε προέρχεται από ανάθεση σε τρίτους. Όλες οι πηγές που χρησιμοποιήθηκαν (κάθε είδους, μορφής και προέλευσης) για τη συγγραφή της περιλαμβάνονται στη βιβλιογραφία.

Ζορμπάς, Μιχαήλ

Υπογραφή

ΕΥΧΑΡΙΣΤΙΕΣ

Σε αυτό το σημείο θα ήθελα να ευχαριστήσω τον επιβλέπων καθηγητή κ. Ιωάννη Τσούλο για την συνεργασία και τις γνώσεις που μου προσέφερε όλο αυτό το διάστημα και κατά τη διάρκεια των σπουδών μου.

Στη συνέχεια, θα ήθελα να ευχαριστήσω τους καθηγητές του τμήματος Μηχανικών Πληροφορικής του ΤΕΙ Ηπείρου για τις γνώσεις και τον χρόνο που μου χάρισαν λύνοντας κάθε μου απορία.

Τέλος, το μεγαλύτερο ευχαριστώ, το οφείλω στην οικογένειά μου για την στήριξη που μου παρείχαν, σε πολλά επίπεδα, όλα αυτά τα χρόνια καθώς και τους φίλους-συμφοιτητές που μοιράστηκα μαζί τους αξέχαστα φοιτητικά χρόνια.

ΠΕΡΙΛΗΨΗ

Οι γενετικοί αλγόριθμοι είναι εμπνευσμένοι από τη φύση και συγκεκριμένα από τη θεωρία εξέλιξης των ειδών. Έχουν εφαρμοστεί σε πολλούς επιστημονικούς τομείς και πολλά από τα προβλήματα που επιλύουν είναι μείζονος σημασίας για αυτό και τα τελευταία χρόνια μεγαλώνει το ενδιαφέρον των ανθρώπων για αυτούς. Στο παρόν κείμενο, υλοποιείται ένα γραφικό περιβάλλον με σκοπό την εύκολη και γρήγορη χρήση ενός κώδικα κατηγοριοποίησης δεδομένων με γενετικούς αλγορίθμους, από έναν μέσο χρήστη. Ο κώδικας που κατηγοριοποιεί τα δεδομένα χρησιμοποιεί τη μέθοδο της γραμματικής εξελίξεως, η οποία μέθοδος παράγει προγράμματα σε οποιαδήποτε γλώσσα προγραμματισμού. Η γραμματική εξέλιξη, για να λειτουργήσει, χρειάζεται δυο βασικά στοιχεία μια γραμματική σε BNF (Backus-Naur Form) και τη σχετική συνάρτηση καταλληλότητας. Στη συνέχεια, δίνεται ο τρόπος εγκατάστασης του ολοκληρωμένου περιβάλλον ανάπτυξης Qt Creator, καθώς και κάποια παραδείγματα μαζί με τον κώδικα υλοποίησης με σκοπό ο αναγνώστης να πάρει μια μικρή γεύση για τον τρόπο με τον οποίο μπορεί να υλοποιηθεί μια μικρή εφαρμογή στο Qt Creator. Τέλος, παρουσιάζεται η εφαρμογή που υλοποιήθηκε όπως και η λύση ενός προβλήματος που αφορά την ταξινόμηση τριών ποικιλιών κρασιών με βάση τα δεδομένα μιας χημικής ανάλυσης.

Λέξεις-κλειδιά: τεχνητά νευρωνικά δίκτυα, γενετικοί αλγόριθμοι, γραμματική εξέλιξη, πρόβλημα κατηγοριοποίησης.

ABSTRACT

Genetic algorithms are inspired by nature and specifically by the theory of species evolution. They have been applied in many scientific areas and many of the problems they solve are of major importance for this, and in recent years people's interest has grown for them. In the present text, a graphical interface is implemented in order to easily and quickly use a data categorization code with genetic algorithms from an average user. The code that categorizes the data uses the grammatical evolution method, which produces programs in any programming language. The grammatical evolution, in order to work, requires two basic elements of a BNF (Backus-Naur Form) grammar and the relative fitness function. Next, we are given the way to install an Integrated Development Environment, the Qt Creator, as well as some examples along with the implementation code for the reader to get a little glimpse into how a small Qt Creator application can be created. Finally, we present the application that we created and the solution of a problem concerning the classification of three varieties of wines based on the data of a chemical analysis.

Keywords: artificial neural networks, genetic algorithms, grammatical evolution, classification problem.

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

ΚΑΤΑΛΟΓΟΣ ΠΙΝΑΚΩΝ

ΚΑΤΑΛΟΓΟΣ ΕΙΚΟΝΩΝ

1.

Το πρόβλημα

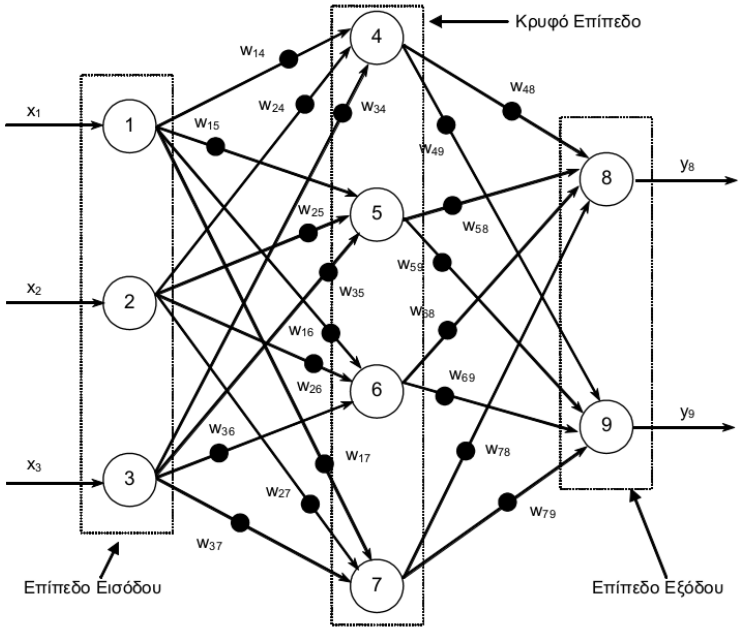
Το πρόβλημα που πραγματεύεται το παρόν κείμενο περιορίζεται στη προσπάθεια συγγραφής κώδικα με σκοπό την υλοποίηση ενός γραφικού περιβάλλοντος για την εύκολη, και όσο το δυνατόν γρήγορη, χρήση ενός κώδικα κατηγοριοποίησης δεδομένων από έναν χρήστη. Το γραφικό περιβάλλον έχει υλοποιηθεί αποκλειστικά σε Qt C++. Ο χρήστης θα πρέπει να εκπαιδεύσει τον αλγόριθμο με ένα σύνολο δεδομένων ή αλλιώς να εισάγει τα δεδομένα εκπαίδευσης, στη συνέχεια να αρχικοποιήσει ένα σύνολο μεταβλητών και τέλος να εισάγει τα δεδομένα που θα πρέπει να κατηγοριοποιηθούν. Ο κώδικας κατηγοριοποιεί τα δεδομένα με βάση μια μέθοδο που ονομάζεται γραμματική εξέλιξη. Η μέθοδος της γραμματικής εξέλιξης είναι μια μέθοδος η οποία έχει εφαρμοστεί με επιτυχία σε διάφορους τομείς όπως για παράδειγμα στον τομέα των οικονομικών, μουσικής συνθέσεως κ.α.

1.1 Τεχνητά νευρωνικά δίκτυα

Τα τεχνητά νευρωνικά δίκτυα, είναι ένα μοντέλο αποτελούμενο από συνδεδεμένους νευρώνες μεταξύ τους τα οποία βασίζονται σε βιολογικά πρότυπα και συγκεκριμένα στον ανθρώπινο εγκέφαλο. Πρωτοπόροι στον τομέα αυτό οι McCulloch και Pitts οι οποίοι τη δεκαετία του ‘40 περιγράψανε ένα απλό μοντέλο της ηλεκτρικής δραστηριότητας του νευρώνα. Ένας νευρώνας μπορεί να βρίσκεται σε δυο καταστάσεις, η πρώτη κατάσταση είναι όταν λέμε ότι ο νευρώνας “πυροβολεί” στη μέγιστη συχνότητα και η δεύτερη κατάσταση είναι όταν ο νευρώνας μένει αδρανής. Οι δύο αυτές καταστάσεις στις οποίες εισέρχεται ο νευρώνας γίνονται κάτω από ορισμένες συνθήκες τις οποίες θα προσπαθήσω να εξηγήσω σε επόμενο κεφάλαιο.

Κάθε νευρώνας είναι συνδεδεμένος με τον αντίστοιχο γειτονικό του νευρώνα και όλοι οι

νευρώνες μαζί βρίσκονται σε ένα κοινό επίπεδο. Υπάρχουν τουλάχιστον δυο επίπεδα νευρώνων τα οποία ονομάζονται επίπεδο εισόδου και επίπεδο εξόδου. Κάθε επιπλέον επίπεδο ανάμεσα στα επίπεδα εισόδου και επίπεδα εξόδου, ονομάζεται κρυφό επίπεδο (βλ. Εικόνα 1.1). Κάθε έξοδος ενός νευρώνα είναι η είσοδος ενός άλλου, με τον τρόπο αυτό όλοι οι νευρώνες μαζί αλληλεπιδρούν μεταξύ τους, ανάλογα με τις εισόδους που λαμβάνουν, και καταλήγουν σε ένα αποτέλεσμα. Εν κατακλείδι, τα τεχνητά νευρωνικά δίκτυα είναι μια ιδιαίτερη προσέγγιση στη δημιουργία συστημάτων με νοημοσύνη. Όλα τα τεχνητά νευρωνικά δίκτυα έχουν κάποιες βασικές ιδιότητες όπως την ικανότητα να μαθαίνουν μέσω παραδειγμάτων, έχουν μεγάλη ανοχή σε σφάλματα και μια εξαιρετική ικανότητα για αναγνώριση προτύπων. (8)



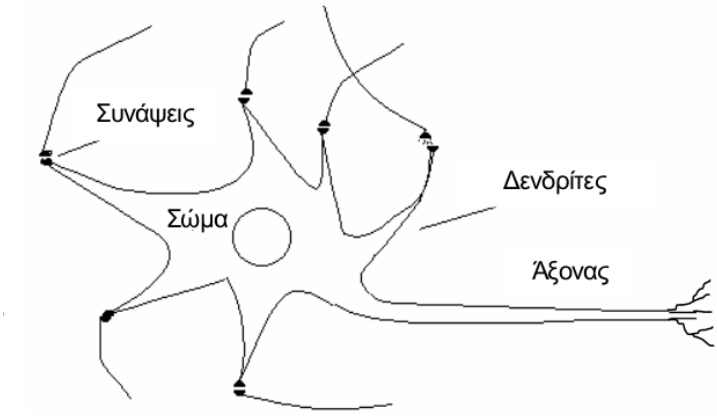
Εικόνα 1.1: Επίπεδα νευρώνων.

1.1.1 Βιολογικός νευρώνας

Ένα βασικό δομικό στοιχείο του ανθρώπινου εγκεφάλου είναι οι νευρώνες του, οι οποίοι

δημιουργούν ένα σχετικά πυκνό δίκτυο επικοινωνίας μεταξύ τους. Ο ανθρώπινος εγκέφαλος θεωρείται από τους πιο πολύπλοκους "υπολογιστές" και ο χρόνος απόκρισης των νευρώνων του ανέρχεται της τάξεως των msec. Ένας ανθρώπινος εγκέφαλος αποτελείται από περίπου 100 δισεκατομμύρια νευρώνες, όπου κάθε νευρώνας έχει περίπου 10^3 συνάψεις.

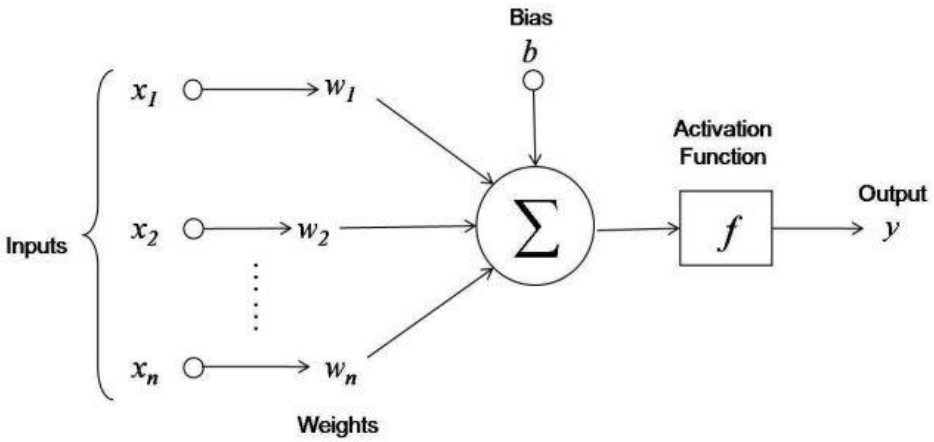
Τα βασικά χαρακτηριστικά ενός νευρώνα είναι το σώμα του νευρώνα, οι δενδρίτες, ο άξονας και οι συνάψεις. Στο σώμα του νευρώνα εισέρχονται σήματα, χρησιμοποιώντας ως μέσο τους δενδρίτες, όπου στη συνέχεια τα σήματα αυτά συνδυάζονται και αν η έξοδος του νευρώνα ξεπερνά κάποιο κατώφλι, τότε παράγεται ένας ηλεκτρικός παλμός από τον άξονα με μεγάλη συχνότητα ο οποίος διαδίδεται προς τους γειτονικούς νευρώνες. Εδώ έχουμε την πρώτη κατάσταση στην οποία εισέρχεται ο νευρώνας, δηλαδή λέμε ότι ο νευρώνας "πυροβολεί", όταν όμως το σύνολο των εισερχόμενων σημάτων δεν ξεπερνάει το κατώφλι που έχει τεθεί, τότε σε τυχαίες στιγμές ο νευρώνας παράγει πολύ αραιά παλμούς. Αυτή είναι η δεύτερη κατάσταση που μπορεί να εισέρθει ένας νευρώνας δηλαδή λέμε ότι ο νευρώνας είναι αδρανής. Στην εικόνα 1.2 βλέπουμε τα δομικά στοιχεία ενός βιολογικού νευρώνα που το απαρτίζουν. (9) (10)



Εικόνα 1.2: Δομικά στοιχεία ενός βιολογικού νευρώνα.

1.1.2 Μοντέλο ενός τεχνητού νευρώνα

Η δομή ενός τεχνητού νευρώνα διαισθητικά και όχι μόνο, βασίζεται σε μεγάλο βαθμό στη δομή ενός βιολογικού νευρώνα. Κάθε νευρώνας διαθέτει τις εισόδους του (x_1,x_2,...x_n), τα βάρη του (w_1,w_2,...w_n), έναν αθροιστή (Σ) ο οποίος δίνει το άθροισμα των σταθμισμένων εισόδων, ένα κατώφλι η αλλιώς bias (b) το οποίο είναι ένας πραγματικός αριθμός και βοηθά το τεχνητό νευρωνικό δίκτυο να έχει καλύτερη ικανότητα ταξινόμησης, μια συνάρτηση ενεργοποίησης f() όπου από εκεί περνάει η έξοδος του αθροιστή, και τέλος την έξοδο y που δίνει ο τεχνητός νευρώνας (βλ. Εικόνα 1.3).



Εικόνα 1.3: Δομικά στοιχεία ενός τεχνητού νευρώνα.

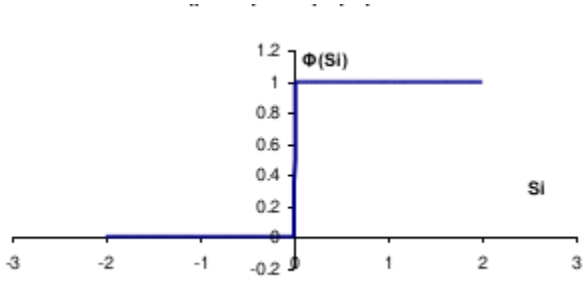
Πιο αναλυτικά, η είσοδος που δέχεται ο νευρώνας αποτελείται από το συνολικό άθροισμα από τα γινόμενα των βαρών (w) (συμπεριλαμβανομένου και του bias) με των εισόδων (x), δηλαδή είναι $u = \sum x_i w_i$, όπου i = 0..n., οπότε η έξοδος ενός τεχνητού νευρώνα προκύπτει από την εξής σχέση: $y_i = f(u_i)$. Σύμφωνα με το μοντέλο των McCulloch-Pitts ,αν το άθροισμα u > b (bias ή αλλιώς κατώφλι) τότε ο νευρώνας εισέρχεται στην πρώτη κατάσταση, δηλαδή λέμε ότι "πυροβολεί", αλλιώς εισέρχεται στη δεύτερη κατάσταση δηλαδή παραμένει αδρανής. (9) (10)

Η συνάρτηση ενεργοποίησης f(), ορίζει την έξοδο ενός νευρώνα συναρτήσει του επιπέδου ενεργοποίησης της εισόδου. Υπάρχουν διάφορες συναρτήσεις ενεργοποίησης μερικές από αυτές είναι:

- Βηματική συνάρτηση 0/1:

f(u) = 0 αν u ≤ 0

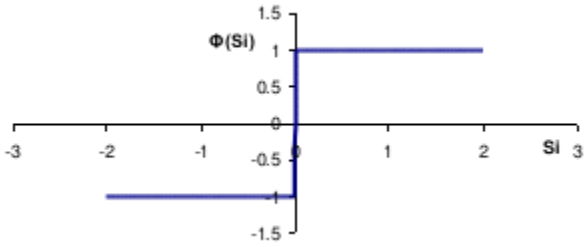
f(u) = 1 αν u > 0



- Βηματική συνάρτηση -1/1:

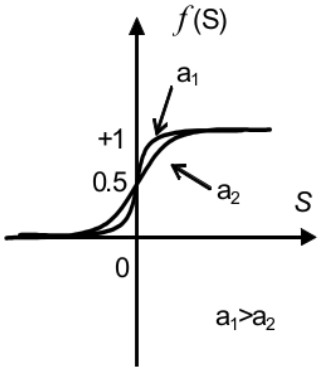
f(u) = -1 αν u ≤ 0

f(u) = 1 αν u > 0



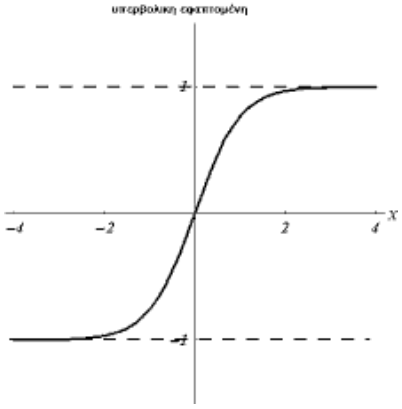
- Σιγμοειδής συνάρτηση:

$f(u) = 1/(1+e^{-u})$



- Υπερβολική εφαπτομένη:

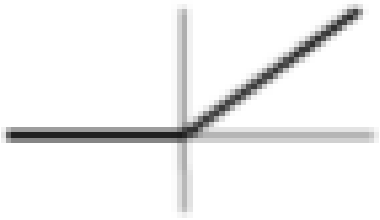
$f(u) = \tanh(u) = (1-e^{-u})/(1+e^{-u})$



- Συνάρτηση ράμπας:

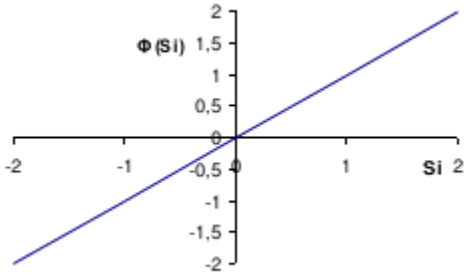
f(u) = 0 αν u ≤ 0

f(u) =u αν u > 0



- Γραμμική συνάρτηση:

f(u) = u



1.1.3 Σχεδιασμός νευρωνικών δικτύων

Ο σχεδιασμός νευρωνικών δικτύων αναφέρεται σε έναν μεγάλο αριθμό δοκιμών και αποφάσεων που θα κληθεί ο εκάστοτε προγραμματιστής να πάρει, προκειμένου να "στήσει" μια ικανοποιητική αρχιτεκτονική ενός δικτύου από νευρώνες. Μια τέτοια αρχιτεκτονική αποτελείται από την τοποθέτηση των νευρώνων σε επίπεδα, από τον τύπο σύνδεσης που θα έχει κάθε νευρώνας σε κάθε επίπεδο, καθώς και τον τύπο σύνδεσης των

νευρώνων που βρίσκονται σε διαφορετικά επίπεδα το καθένα. Τέλος, αλλά εξίσου σημαντικό με τα προηγούμενα, είναι η μάθηση ή αλλιώς η εκπαίδευση του δικτύου δηλαδή, το πως θα μάθει το δίκτυο τις κατάλληλες τιμές για τα βάρη σύνδεσης χρησιμοποιώντας ένα σύνολο δεδομένων εκπαίδευσης. (6) (17) (23)

- **Επίπεδα νευρώνων:** Σε γενικές γραμμές όλα τα τεχνητά νευρωνικά δίκτυα έχουν παρόμοια δομή τοπολογίας μεταξύ τους. Ουσιαστικά εδώ, οι τεχνητοί νευρώνες ομαδοποιούνται σε επίπεδα, τα οποία επικοινωνούν μεταξύ τους. Όπως ειπώθηκε στο κεφάλαιο 1.1, τα νευρωνικά δίκτυα ομαδοποιούνται σε τουλάχιστον δυο επίπεδα, τα οποία ονομάζονται επίπεδο εισόδου και επίπεδο εξόδου (βλ. Εικόνα 1.1). Στο επίπεδο εισόδου παίρνουν είσοδο από το εξωτερικό περιβάλλον ενώ, στο επίπεδο εξόδου οι νευρώνες του συστήματος δίνουν έξοδο προς τον χρήστη ή το εξωτερικό περιβάλλον, τα υπόλοιπα επίπεδα και κατ' επέκταση οι νευρώνες που ανήκουν σε αυτά θεωρούνται κρυμμένοι. Κάθε έξοδος ενός επιπέδου από νευρώνες, είναι είσοδος ενός άλλου. Η διαδικασία αυτή συνεχίζεται μέχρι να "ικανοποιηθεί" μια συγκεκριμένη συνθήκη ή μέχρι το επίπεδο εξόδου να "πυροβολήσει" στο εξωτερικό περιβάλλον. Εάν αυξήσουμε τον αριθμό των κρυμμένων νευρώνων θα έχουμε μια "υπερκατασκευή" και το δίκτυο θα έχει πρόβλημα. Τα δεδομένα εκπαίδευσης θα έχουν αποθηκευτεί στη μνήμη του δικτύου, με αποτέλεσμα το δίκτυο να είναι άχρηστο σε νέα σύνολα δεδομένων.

- **Τύπος σύνδεσης νευρώνων:** Όπως είπαμε και πριν, οι νευρώνες συνδέονται μεταξύ τους μέσω "μονοπατιών" μεταφέροντας την έξοδο ενός νευρώνα ως είσοδο ενός άλλου. Τα "μονοπάτια" αυτά συνήθως είναι μονής κατεύθυνσης, αλλά δεν είναι απίθανο μεταξύ δυο νευρώνων να υπάρχει μια αμφίδρομη επικοινωνία μεταξύ τους. Η αμφίδρομη επικοινωνία οφείλεται στο γεγονός ότι μπορεί να υπάρχει άλλο ένα "μονοπάτι" προς την αντίθετη κατεύθυνση. Κάθε νευρώνας, παράγει μια έξοδο η οποία επικοινωνεί με άλλους νευρώνες και αντίστοιχα δέχεται είσοδο από πολλούς νευρώνες μαζί. Σε ένα κοινό επίπεδο, ένας νευρώνας μπορεί να έχει τουλάχιστον μια ή καμία σύνδεση με άλλους νευρώνες αλλά οι νευρώνες ενός επιπέδου είναι πάντα συνδεδεμένοι με νευρώνες τουλάχιστον ενός διαφορετικού επιπέδου. Αν το σκεφτεί κανείς είναι λογικό διότι με κάποιο τρόπο η

πληροφορία ξεκινώντας από το επίπεδο εισόδου θα πρέπει να φτάσει στο επίπεδο εξόδου. Παρακάτω, δίνονται δύο πίνακες που αναφέρονται στους δυο διαφορετικούς τύπους συνδέσεων· ο πρώτος είναι όταν δυο διαφορετικά επίπεδα νευρώνων συνδέονται μεταξύ τους, και ο δεύτερος είναι όταν υπάρχουν συνδέσεις νευρώνων σε ένα κοινό επίπεδο.

Συνδέσεις νευρώνωνμεταξύ διαφορετικών επιπέδων	
Πλήρως συνδεδεμένα	Κάθε νευρώνας ενός επιπέδου είναι συνδεδεμένος σε κάθε νευρώνα ενός διαφορετικού επιπέδου.
Μερικώς συνδεδεμένα	Ένας νευρώνας ενός επιπέδου δεν είναι απαραίτητο να είναι συνδεδεμένος σε όλους του ς νευρώνες ενός άλλου επιπέδου.
Εμπρόσθιας τροφοδότησης	Οι νευρώνες ενός επιπέδου Α στέλνουν την έξοδό τους σε νευρώνες ενός επιπέδου Β, αλλά δεν λαμβάνουν είσοδο από τους νευρώνες του επιπέδου Β. Μπορεί να είναι πλήρως ή μερικώς συνδεδεμένα.
Διπλής κατεύθυνσης	Οι νευρώνες ενός επιπέδου Α στέλνουν την έξοδό τους σε νευρώνες ενός επιπέδου Β, αλλά η έξοδο των νευρώνων του επιπέδου Β μπορεί να χρησιμοποιηθεί σαν είσοδο στους νευρώνες του επιπέδου Α. Μπορεί να είναι πλήρως ή μερικώς συνδεδεμένα.
Ιεραρχικά	Οι νευρώνες ενός χαμηλότερου επιπέδου μπορούν να επικοινωνήσουν με τους νευρώνες του αμέσως επόμενου επιπέδου.
Απήχησης	Η επικοινωνία μεταξύ αυτών των επιπέδων είναι διπλής κατεύθυνσης και μπορούν να

	στέλνουν μηνύματα στις συνδέσεις αρκετές φορές μέχρι να ικανοποιηθεί μια συγκεκριμένη συνθήκη.

Πίνακας 1.1: Συνδέσεις νευρώνων μεταξύ διαφορετικών επιπέδων.

Συνδέσεις νευρώνων σε κοινό επίπεδο	
Επαναλαμβανόμενα	Οι νευρώνες σε ένα επίπεδο είναι μπορεί να είναι πλήρως ή μερικώς συνδεδεμένοι μεταξύ τους. Μόλις οι νευρώνες αυτοί λάβουν μια πληροφορίαωςείσοδο από ένα άλλο επίπεδο, για να μπορέσει η πληροφορία αυτή να "φύγει" σε άλλο επίπεδο, θα πρέπει να ικανοποιηθεί μια συγκεκριμένη συνθήκη στο τρέχον επίπεδο.
Περιβάλλον γύρω από το κέντρο	Σε ένα επίπεδο, ένας νευρώνας διαθέτει διεγερτικές συνδέσεις με τον εαυτό του και τους γειτονικούς νευρώνες και έχει ανασταλτικές συνδέσεις με άλλους νευρώνες. Μια ομάδα από νευρώνες διεγείρει τον εαυτό της και τα μέλη της ομάδας και εμποδίζει όλα τα μέλη άλλων ομάδων. Μετά από κάποιους γύρους ανταλλαγής σημάτων, οι νευρώνες με ενεργή έξοδο κερδίζουν και τότε αλλάζουν τα βάρη της ομάδας και των μελών της.

Πίνακας 1.2: Συνδέσεις νευρώνων σε κοινό επίπεδο.

- **Μάθηση:** Ο ανθρώπινος εγκέφαλος μαθαίνει μέσω εμπειριών. Η εμπειρία και η γνώση ενός τεχνητού νευρωνικού δικτύου, αποθηκεύεται στα βάρη του και η εκπαίδευσή του είναι μια επαναληπτική διαδικασία όπου γίνεται συνεχείς αναπροσαρμογή των βαρών,

ώστε το τεχνητό νευρωνικό να έχει την επιθυμητή απόκριση. Αυτό επιτυγχάνεται με αλγορίθμους εκπαίδευσης. Η ικανότητα ενός νευρωνικού δικτύου να μάθει, καθορίζεται από την αρχιτεκτονική του και από την αλγοριθμική μέθοδο που επιλέγεται για την εκπαίδευση. Παρακάτω, δίνεται ένας πίνακας με τρεις μεθόδους εκπαίδευσης αλγορίθμων.

Μέθοδοιεκπαίδευσης αλγορίθμων	
Μάθηση χωρίς επίβλεψη	Οι νευρώνες που βρίσκονται στα κρυμμένα επίπεδα, θα πρέπει να βρούνε ένα τρόπο να οργανωθούνε μόνα τους χωρίς κάποια εξωτερική βοήθεια. Στο δίκτυο δεν παρέχονται κάποια δείγματα εξόδου που να μπορούν να υπολογίσουν την πρόβλεψη για ένα δεδομένο διάνυσμα εισόδου. Ουσιαστικά, το δίκτυο μαθαίνει κάνοντας (learning by doing).
Μάθηση με επίβλεψη	Οι νευρώνες εδώ χρειάζονται κάποιας μορφής εξωτερικής βοήθειας η οποία αναλαμβάνει το ρόλο του "δασκάλου". Η βοήθεια συνήθως είναι ένα σύνολο από δεδομένα, τα λεγόμενα δεδομένα εκπαίδευσης, ή ένας παρατηρητής ο οποίος βαθμολογεί την απόδοση του δικτύου. Οι συνδέσεις των νευρώνων που βρίσκονται στα κρυφά επίπεδα, είναι τυχαία διευθετημένες και ανακατατάσσονται ανάλογα με το πόσο κοντά είναι το δίκτυο στην επίλυση του προβλήματος.
Οπισθοδιάδοση(Backpropagation)	Η μέθοδος αυτή έχει αποδειχτεί γενικά επιτυχής στην εκπαίδευση πολυεπίπεδων νευρωνικών δικτύων. Παρέχεται πληροφορία από το ίδιο το δίκτυο για τα σφάλματα τα οποία φιλτράρονται και χρησιμοποιούνται για να προσαρμοστούν οι συνδέσεις μεταξύ των επιπέδων. Με τον τρόπο

	αυτό βελτιώνεται η απόδοση του δικτύου. Εν κατακλείδι,είναι τηςμορφήςμάθηση με επίβλεψη.

Πίνακας 1.3: Μέθοδοι εκπαίδευσης αλγορίθμων

Κανόνες μάθησης	
Κανόνας του Hebb	Είναι ο πρώτος και από τους πιο γνωστούς κανόνες μάθησης ο οποίος προτάθηκε από τον Donald Hebb στο βιβλίο του The organization of Behavior (1949). Ο κανόνας αυτός αναφέρει το εξής: Αν ένας νευρώνας λάβει είσοδο από έναν άλλο νευρώνα και εάν είναι και οι δυο υψηλά ενεργοί, τότε τα βάρη μεταξύ των νευρώνων θα πρέπει να ενισχυθούν.
Νόμος του Hopfield	Ο νόμος αυτός αναφέρει ότι: εάν η επιθυμητή έξοδος και η είσοδος είναι και οι δυο ενεργές ή και οι δυο ανενεργές τότε αύξησε τα βάρη σύνδεσης με το ρυθμό εκμάθησης αλλιώς μείωσε τα βάρη με το ρυθμό εκμάθησης. Ο ρυθμός εκμάθησης συνήθως είναι θετικός αριθμός μεταξύ του μηδέν και ένα.
Κανόνας Δέλτα	Ο κανόνας Δέλταείναι μια περαιτέρω παραλλαγή του κανόνα του Hebb, ο οποίος χρησιμοποιείται αρκετά συχνά. Η ιδέα του κανόνα αυτού βασίζεται στη συνεχή τροποποίησητων δυνατοτήτων των συνδέσεων εισόδου για τη μείωση της διαφοράς μεταξύ της επιθυμητής τιμής εξόδου και της πραγματικής εξόδου ενός νευρώνα. Ο κανόνας αυτός αλλάζει τη σύνδεση των βαρών με τρόπο τέτοιο ώστενα

	ελαχιστοποιεί το μέσο τετραγωνικό σφάλμα. Το σφάλμα διαδίδεται προς τα πίσω σε προηγούμενο επίπεδο. Η διαδικασία αυτή συνεχίζεται μέχρι να φτάσει στο αρχικό επίπεδο.

Πίνακας 1.4: Κανόνες μάθησης.

1.2 Προβλήματα κατηγοριοποίησης

Η κατηγοριοποίηση, είναι ένα μείζον θέμα που απασχολεί άμεσα τον κλάδο της μηχανικής μάθησης και έχει να κάνει με το πως θα "διδάξει" τους αλγορίθμους να κατηγοριοποιούν δεδομένα εισόδου με συγκεκριμένα κριτήρια. Συνήθως, τα δεδομένα εισόδου χωρίζονται σε δυο μέρη, το ένα είναι το σύνολο εκπαίδευσης, όπου χρησιμοποιείται για να κατασκευαστεί το μοντέλο κατηγοριοποίησης και το άλλο είναι το σύνολο ελέγχου, που χρησιμοποιείται για την επικύρωση του μοντέλου. Ουσιαστικά, ο αλγόριθμος κατηγοριοποίησης προσπαθεί να αναγνωρίσει άγνωστα αντικείμενα ή φαινόμενα σαν μέλη κάποιας γνωστής κλάσης αντικειμένων ή φαινομένων αντίστοιχα. Όταν ο αλγόριθμος τοποθετεί σε κλάσεις δεδομένα των οποίων τα χαρακτηριστικά είναι ήδη προκαθορισμένα δηλαδή οι κλάσεις που θα τοποθετηθούν είναι από την αρχή γνωστές, αυτό ορίζεται σαν μια μορφή "μάθηση με επίβλεψη". Βέβαια, υπάρχει και μια άλλη έκδοση της κατηγοριοποίησης όπου δεν καθορίζονται κατηγορίες, οπότε αυτό ορίζεται της μορφής "μάθηση χωρίς επίβλεψη". Ο αλγόριθμος εκεί ομαδοποιεί τα δεδομένα βρίσκοντας τα κοινά χαρακτηριστικά τους. Μερικά ενδεικτικά παραδείγματα είναι η κατηγοριοποίηση των εισερχόμενων emails. Τα email συνήθως κατηγοριοποιούνται ανάλογα με τον τίτλο που φέρουν ή το περιεχόμενο και αντίστοιχα το λογισμικό κατατάσσει το μήνυμα αυτό σε spam ή όχι. Ένα άλλο παράδειγμα θα μπορούσε να είναι η κατηγοριοποίηση τεσσάρων φρούτων που έχουν παρόμοιο σχήμα μεταξύ τους, όπως το μήλο, το πορτοκάλι, το λεμόνι και το μανταρίνι. Θα μπορούσε να υπάρχει μια κάμερα που να "διαβάζει" το σχήμα, το μέγεθος , το χρώμα και ίσως και το βάρος των φρούτων και ανάλογα με τα αποτελέσματα να εκτελεί ταξινόμηση των φρούτων στην κατηγορία που ανήκουν. (11)

1.2.1 Τεχνικές κατηγοριοποίησης

Στο κεφάλαιο αυτό θα ήθελα να αναφέρω ενδεικτικά κάποιες τεχνικές που χρησιμεύουν στην κατηγοριοποίηση δεδομένων. Τέτοιες είναι οι: γραμμικοί ταξινομητές όπως η λογιστική παλινδρόμηση και ο απλοϊκός Bayes, δέντρα απόφασης, νευρωνικά δίκτυα και κοντινότερος γείτονας. (12) (14)

- **Logistic regression (λογιστική παλινδρόμηση):** Είναι μια στατιστική μέθοδος η οποία χρησιμεύει στην ανάλυση ενός συνόλου από δεδομένα στο οποίο υπάρχουν τουλάχιστον μια ανεξάρτητη μεταβλητή που καθορίζει το αποτέλεσμα. Το αποτέλεσμα μετράται με μια διχοτόμο μεταβλητή. Στόχος της τεχνικής αυτής είναι, η εύρεση του καλύτερου μοντέλου για την περιγραφή της σχέσης ανάμεσα στο διχοτόμο χαρακτηριστικό του ενδιαφέροντος (εξαρτώμενη μεταβλητή) και ένα σύνολο ανεξάρτητων μεταβλητών.

- **Naive Bayes (απλοϊκός Bayes):** Η τεχνική αυτή βασίζεται στην θεωρία του Bayes. Δηλαδή, κάνει μια πρόβλεψη και εμφανίζει σαν αποτέλεσμα την πιθανότητα που έχει ένα δεδομένο να ανήκει σε μια συγκεκριμένη κλάση. Το μοντέλο Naive Bayes είναι εύκολο να κατασκευαστεί και είναι ιδιαίτερα χρήσιμο για πολύ μεγάλα σύνολα δεδομένων.

- **Decision Trees (Δέντρα απόφασης):** Τα δέντρα απόφασης "χτίζουν" μοντέλα ταξινόμησης η παλινδρόμησης με τη μορφή δέντρου. Ουσιαστικά, "σπάει" το σύνολο των δεδομένων σε μικρότερα υποσύνολα, ταυτόχρονα όμως αναπτύσσει σταδιακά ένα δέντρο απόφασης.

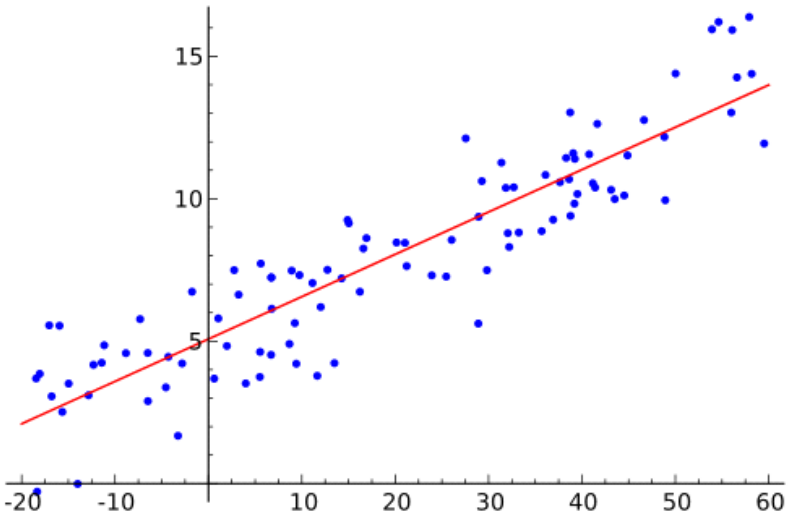
- **Neural Network (Νευρωνικά δίκτυα):** Είναι μια τεχνική η οποία αποτελείται από νευρώνες ομαδοποιημένα σε επίπεδα, τα οποία επικοινωνούν μεταξύ τους. Γενικά, είναι αποτέλεσμα μιας προσπάθειας να μιμηθεί τον ανθρώπινο εγκέφαλο.

- **Nearest Neighbor (K-Κοντινότεροι Γείτονες):** Είναι της μορφής μάθηση με επίβλεψη. Βασίζεται στη μικρότερη απόσταση, από το δείγμα ερωτήματος στα δείγματα

εκπαίδευσης για τον προσδιορισμό του K- γειτόνων. Δηλαδή, υπολογίζει την απόσταση ανάμεσα στο δείγμα ερωτήματος και όλα τα δεδομένα εκπαίδευσης, έπειτα ταξινομεί την απόσταση και προσδιορίζει τους πλησιέστερους γείτονες με βάση τη μικρότερη κ-οστή απόσταση. Τέλος, χρησιμοποιεί την πλειοψηφία των πλησιέστερων γειτόνων ως την τιμή πρόβλεψης του στιγμιότυπου ερωτήματος.

1.3 Προβλήματα παλινδρόμησης (regession)

Η παλινδρόμηση είναι η συνεχής πρόβλεψη μεταβλητών. Η ανάλυση της παλινδρόμησης είναι το στατιστικό εκείνο μοντέλο το οποίο χρησιμεύει στην πρόβλεψη δεδομένων αποτελούμενα από νούμερα. Η παλινδρόμηση, γενικά είναι ένα χρήσιμο εργαλείο στην στατιστική, το οποίο χρησιμοποιείται επίσης για την μέτρηση της σχέσης μεταξύ δυο ή περισσότερων μεταβλητών, όπου η μια είναι ανεξάρτητη μεταβλητή ενώ οι άλλες εξαρτημένες. Η διαφορά με τη κατηγοριοποίηση είναι ότι ομαδοποιεί τα δεδομένα σε κλάσεις, ενώ η παλινδρόμηση προβλέπει την έξοδο χρησιμοποιώντας τα δεδομένα εκπαίδευσης. Ένα απλό παράδειγμα παλινδρόμησης θα μπορούσε να είναι το παρακάτω (βλ. Εικόνα 1.4). (13) (22)



Εικόνα 1.4: Παράδειγμα απλής παλινδρόμησης.

Έστω σε ένα καρτεσιανό επίπεδο έχουμε δείγματα τιμών $\{x_i, y_i\}$, στόχος και λύση του προβλήματος είναι να βρεθεί μια γραμμή της μορφής $f(x) = y = \alpha + \beta x$,τέτοια ώστε να ταιριάζει καλύτερα στο σύνολο των δειγμάτων του καρτεσιανού επιπέδου. Με βάση το μοντέλο αυτό μπορούμε να «προβλέψουμε» τις τιμές του y για νέες τιμές του x. Εδώ ανεξάρτητη μεταβλητή είναι το x, και η εξαρτημένη είναι το y.

2. Κατασκευαζόμενα τεχνητά νευρωνικά δίκτυα

Πριν προχωρήσουμε στα κατασκευαζόμενα τεχνητά νευρωνικά δίκτυα, θα ήθελα πρωτίστως, να κάνω μια εισαγωγή στους γενετικούς αλγορίθμους και σε μια μέθοδο που ονομάζεται γραμματική εξέλιξη. Οι γενετικοί αλγόριθμοι, είναι εμπνευσμένοι από τη φύση και συγκεκριμένα βασίζονται πάνω στη θεωρία του Κάρολου Δαρβίνου, η γραμματική εξέλιξη είναι μια μέθοδος και ανήκει κάτω από την ομπρέλα των εξελικτικών αλγορίθμων.

2.1 Εισαγωγή στους γενετικούς αλγορίθμους

Στο παρών υποκεφάλαιο, θα γίνει μια συζήτηση γύρω από τους γενετικούς αλγορίθμους. Όπως είπαμε και πριν, βασίζονται στην θεωρία του Δαρβίνου περί εξελίξεως των ειδών και με βάση τα στοιχεία που την διέπουν, ο John Holland και οι φοιτητές του το 1975 ανέπτυξαν τους γενετικούς αλγορίθμους. Γενικά, ο αλγόριθμος που ανέπτυξαν μιμείται την φύση και χρησιμοποιείται για την επίλυση προβλημάτων τεχνητής νοημοσύνης και όχι μόνο. Οι γενετικοί αλγόριθμοι είναι μια μορφή μηχανικής μάθησης όπου, με την πάροδο του χρόνου, φαίνεται να κερδίζει όλο και περισσότερους φίλους. Όπως όλα τα συστήματα μηχανικής μάθησης (Βελτιστοποίηση Σμήνους Σωματιδίων, Γενετικός Προγραμματισμός κ.α), οι γενετικοί αλγόριθμοι απαιτούν σημαντικά μεγάλη υπολογιστική ισχύ για την εκτέλεσή τους. (18)

2.1.1 Η θεωρία εξέλιξης των ειδών

Σύμφωνα με τον Δαρβίνο, το «είδος» δεν είναι σταθερό κι αναλλοίωτο. Όλα τα είδη των φυτών και των ζώων, που έζησαν πάνω στη Γη ή που εξακολουθούν να ζουν, είναι το αποτέλεσμα της ασταμάτητης μεταβολής, της εξέλιξης από μια ή ελάχιστες υποτυπώδεις αρχικές μορφές. Αιτία της εξέλιξης είναι η φυσική επιλογή, που οφείλεται στον αγώνα για αναπαραγωγή με σκοπό τη «διαιώνιση του είδους».

Πιο αναλυτικά, η αρχή αυτή διέπεται από κάποια βασικά στοιχεία. Ένα από αυτά είναι ότι σε ένα είδος δεν υπάρχει ανώτερος και κατώτερος οργανισμός, όλοι έχουν ίδιες ανάγκες και την ίδια "υποχρέωση" να διαιωνίσουν το είδος τους. Επίσης, το περιβάλλον γενικά που βρίσκεται το κάθε είδος καθορίζει το πόσους απογόνους μπορεί να αφήσει κάθε μέλος του πληθυσμού. Αν οι συνθήκες διαβίωσης είναι τέτοιες χωρίς φυσικές δυσκολίες, η πιθανότητα μιας ιδιότητας των γονέων να περάσει στους απογόνους είναι αυξημένη.

Έπειτα ισχύει ότι, κάθε είδος αποτελείται από χρωμοσώματα και αυτά με τη σειρά τους αποτελούνται από γονίδια. Ο γενότυπος είναι το σύνολο της πληροφορίας που αποθηκεύεται στα γονίδια. Τα χαρακτηριστικά σε κάθε οργανισμού υπάρχουν κωδικοποιημένα και το σύνολο αυτών που βλέπουμε ονομάζεται φαινότυπος. Τέλος, σε κάθε ζωντανό οργανισμό (χλωρίδα ή πανίδα) βασικές λειτουργίες είναι η αναπαραγωγή και η μετάλλαξη. (15) (18)

2.1.2 Λειτουργία γενετικών αλγορίθμων

Στους γενετικούς αλγορίθμους κάθε χρωμόσωμα παριστάνεται με τη μορφή διανύσματος και τα γονίδια με την μορφή αριθμού (δυαδικό ή δεκαδικό). Στόχος κάθε γενετικού αλγορίθμου είναι η εύρεση της βέλτιστης τιμής της συνάρτησης καταλληλότητας. Η συνάρτηση αυτή, θα πρέπει να αντανακλά το πρόβλημα που υπάρχει, και η τιμή της να διαφοροποιείται για διαφορετικά χρωμοσώματα. Οι γενετικοί αλγόριθμοι ψάχνουν την καλύτερη δυνατή λύση, σε έναν πληθυσμό από υποψήφιες λύσεις οι οποίες αναπαράγονται και μεταλλάσσονται. Γενικά, αποτελούν μια τεχνική παραμετροποίησης με την οποία επιλύονται προβλήματα βελτιστοποίησης. Βελτιστοποίηση, ονομάζεται μια διαδικασία η οποία προσπαθεί να βρει ένα σύνολο παραμέτρων που αποτελούν την καλύτερη δυνατή λύση σε ένα πρόβλημα. Για να γίνει αυτό θα πρέπει αρχικά να οριστεί με ακρίβεια το σύνολο των παραμέτρων και το πλήθος τους, να ορίσουμε τον τύπο της αντικειμενικής συνάρτησης που θέλουμε να βελτιστοποιήσουμε και τέλος να ορίσουμε το σύνολο περιορισμών το οποίο θα πρέπει να πρέπει να πληρεί η επιστρεφόμενη λύση. Όταν η μέθοδος βελτιστοποίησης αποσκοπεί στην εύρεση κάποιου ελαχίστου με περιορισμένη γειτονιά του πεδίου ορισμού, αυτό είναι μέθοδος τοπικής βελτιστοποίησης, αλλιώς είναι μέθοδος καθολικής βελτιστοποίησης.

Στην παράγραφο αυτή, θα γίνει μια προσπάθεια να περιγραφεί η διαδικασία ενός γενετικού αλγορίθμου. Αρχικά λοιπόν, έχουμε την αρχικοποίηση των γενεών και του πληθυσμού,

καθώς και την ανάπτυξη κριτηρίων τερματισμού, με σκοπό να μπορέσουμε να δώσουμε περατότητα στον αλγόριθμο. Τα κριτήρια τερματισμού θα πρέπει να είναι τέτοια ώστε, πρώτον να διακόπτει την επανάληψη όταν αυτή δεν οδηγεί σε κάποιο καλύτερο αποτέλεσμα, και δεύτερον να σταματάει ο αλγόριθμος όταν είμαστε σίγουροι ότι έχει βρεθεί ο στόχος. Γενικά, υπάρχουν τρόποι να ικανοποιούνται, τουλάχιστον το ένα ή και τα δύο κριτήρια μαζί όπως με αριθμό επαναλήψεων, προσέγγιση στόχου, κριτήριο ομοιότητας κ.α. Στην συνέχεια, σε κάθε επανάληψη επιλέγεται ένα υποσύνολο του αρχικού πληθυσμού στο οποίο, εφαρμόζονται οι λειτουργίες της διασταύρωσης και της μετάλλαξης. Τέλος, το υποσύνολο που επιλέξαμε πρώτα, προστίθεται στον τρέχων πληθυσμό με σκοπό να παραχθεί η επόμενη γενιά. Οι λέξεις κλειδιά που θα πρέπει να κρατήσει κανείς από τη διαδικασία που μόλις περιγράψαμε είναι οι εξής: επιλογή, διασταύρωση και μετάλλαξη. Αυτές οι λέξεις περιγράφουν τους γενετικούς τελεστές που θα αναλύσουμε σε επόμενα κεφάλαια.

Βέβαια, κάθε νόμισμα έχει δυο όψεις πράγμα που σημαίνει ότι οι γενετικοί αλγόριθμοι διαθέτουν πλεονεκτήματα και μειονεκτήματα. Τα πλεονεκτήματα αυτών είναι, ότι σε πολλές περιπτώσεις λύνουν προβλήματα γρήγορα και αξιόπιστα, είναι εύκολα επεκτάσιμοι με δυνατότητα τροποποίησης στο εκάστοτε πρόβλημα, έχουν τη δυνατότητα να συνεργαστούν με άλλα συστήματα και τέλος μπορούν να εφαρμοστούν σε μεγάλο πλήθος υπολογιστικών προβλημάτων.

Ένα βασικό τους μειονέκτημα, είναι ότι ο χρόνος σύγκλισής τους είναι μεγάλος δηλαδή είναι αργοί. Επίσης, υπάρχουν περιπτώσεις που γενετικοί αλγόριθμοι έχουν “παγιδευτεί” σε τοπικά ελάχιστα της αντικειμενικής τους συνάρτησης. Για να αποφευχθεί ένα τέτοιο γεγονός απαιτείται από τον προγραμματιστή προσεκτικό σχεδιασμό και να υπάρχει η κατάλληλη προσαρμογή των παραμέτρων του στο πρόβλημα προς επίλυση. (18)

2.1.3 Επιλογή

Στον πραγματικό κόσμο, επιλέγονται τα πιο κατάλληλα μέλη ενός είδους για αναπαραγωγή. Για παράδειγμα, δυο αρσενικά ελάφια προκείμενου να ζευγαρώσουν με ένα θηλυκό ελάφι, θα πρέπει να δοθεί μια μάχη μεταξύ των αρσενικών και ο νικητής θα είναι και αυτός που θα συμμετέχει στο ζευγάρωμα. Συνεπώς, το θηλυκό ελάφι επιλέγει για αναπαραγωγή το καλύτερο αρσενικό ελάφι. Αυτό γίνεται γιατί οι απόγονοι των ελαφιών, και του κάθε είδους γενικότερα, θα πρέπει να κληρονομήσουν όσον το δυνατόν καλύτερα χαρακτηριστικά από τους γονείς με σκοπό την επιβίωση και τη διαιώνιση του είδους. Το ίδιο ακριβώς συμβαίνει και με τους γενετικούς αλγορίθμους, μόνο τα καλύτερα μέλη του πληθυσμού, βάσει κάποιων κριτηρίων, επιλέγονται για διασταύρωση. Για τον σκοπό αυτό, κατασκευάζεται μια λίστα από γονείς οι οποίοι θα ανταλλάξουν γενετικό υλικό, η λίστα αυτή ονομάζεται mating pool. Παρακάτω, θα γίνει μια αναφορά δυο τρόπων επιλογής γονέων. (18)

- **Επιλογή ρουλέτας:** Βασικό χαρακτηριστικό της μεθόδου αυτής είναι η επιλογή ενός στοιχείου, η οποία γίνεται βάση πιθανότητας p_i, δηλαδή

$$p_i = \frac{f_i}{\sum_{j=1}^{M} f_j}$$

$\left[0, \sum_{j=1}^{N} f_j\right)$ όπου f_j είναι η καταλληλότητα του μέλους i. Γενικά, η επιλογή ρουλέτας υλοποιείται με δυο τρόπους: πρώτος τρόπος είναι με χρήση δεκαδικών και ο δεύτερος με πίνακα ακεραίων. Για να υλοποιηθεί με χρήση δεκαδικών θα πρέπει οι καταλληλότητες να ταξινομηθούν από τη χειρότερη στη καλύτερη, έπειτα να επιλεγεί ένας τυχαίος αριθμός r σε διάστημα

$\sum_{j=1}^{i-1} f_j \le r \le \sum_{j=1}^{i} f_j$ και τέλος, να επιλεχθεί για αναπαραγωγή ο ακέραιος r για τον οποίο ισχύει

Η υλοποίηση αυτή έχει το μειονέκτημα ότι είναι αργή. Η υλοποίηση με πίνακα ακεραίων, περιλαμβάνει ένα πίνακα όπου κάθε χρωμόσωμα καταλαμβάνει τόσες θέσεις, ανάλογα με το πόσο μεγάλη είναι η καταλληλότητά του σε σχέση με τις

υπόλοιπες και από εκεί γίνεται και επιλογή χρωμοσωμάτων. Όσες πιο πολλές θέσεις κατέχει ένα χρωμόσωμα τόσες περισσότερες πιθανότητες να επιλεγεί. Το μειονέκτημα της υλοποίησης αυτής είναι η παράμετρος που ορίζει το μέγεθος του πίνακα, δηλαδή ανάλογα με την τιμή της μπορεί να διαφοροποιηθεί σημαντικά το αποτέλεσμα και να μην είναι ακριβής η αναπαράσταση των πιθανοτήτων στον πίνακα. Εν κατακλείδι, η επιλογή ρουλέτας έχει κάποια προβλήματα όπως εκείνο της πρόωρης σύγκλισης, της στασιμότητας και το γεγονός ότι τα χρωμοσώματα με καταλληλότητα μεγαλύτερα της μέσης καταλληλότητας έχουν περισσότερα από ένα αντίγραφα στη mating pool. Αντίθετα, χρωμοσώματα με μικρότερη καταλληλότητα από εκείνη της μέσης καταλληλότητας δεν εμφανίζονται στη mating pool. Παρακάτω, δίνεται μια εικόνα με τον αλγόριθμο επιλογής ρουλέτας με χρήση δεκαδικών και ο αλγόριθμος επιλογής ρουλέτας με χρήση πίνακα ακεραίων.

```
Function select(f)
totalf=0
for i=1..size(f) do
  totalf=totalf+f(i)
end for
sum=0
r=rand(0,totalf)
for i=1..size(f) do
  sum=sum+f(i)
  if(sum>=r) return i
end for
end select
```

```
Function select(p,M)
start=1
for i=1..size(p) do
  k=p(i)*M
  for j=start..start+k do
    v(j)=i
  end for
  start=start+k+1
end for
return v(rand(1,M))
end select
```

Εικόνα 2.1: Αλγόριθμος επιλογής ρουλέτας με χρήση δεκαδικών (αριστερά), με χρήση πίνακα ακεραίων (δεξιά).

- **Επιλογή τουρνουά:** Η μέθοδος αυτή δουλεύει ως εξής: επιλέγει μια σειρά από N χρωμοσώματα ($N \geq 2$) και κρατάει σαν γονέα το χρωμόσωμα που έχει τη μεγαλύτερη τιμή καταλληλότητας. Όπως δηλώνει και το όνομά του είναι ένα τουρνουά μεταξύ

χρωμοσωμάτων. Διαθέτει πλεονεκτήματα, όπως το γεγονός ότι δεν έχει πρόωρη σύγκλιση, ούτε στασιμότητα, δεν διαθέτει ακριβή υπολογισμό της καταλληλότητας και είναι αρκετά γρήγορη. Στην εικόνα 2.2 δίνεται ο αλγόριθμος επιλογής τουρνουά.

```
function select(N)
max_fitness=0
parent=0
for i=1..N do
 r=rand(1,chromosome_count)
 if(fitness(r)>max_fitness) then
   max_fitness = fitness(r)
   parent      = r
 end if
end for
return parent
end select
```

Εικόνα 2.2: Αλγόριθμος επιλογής τουρνουά.

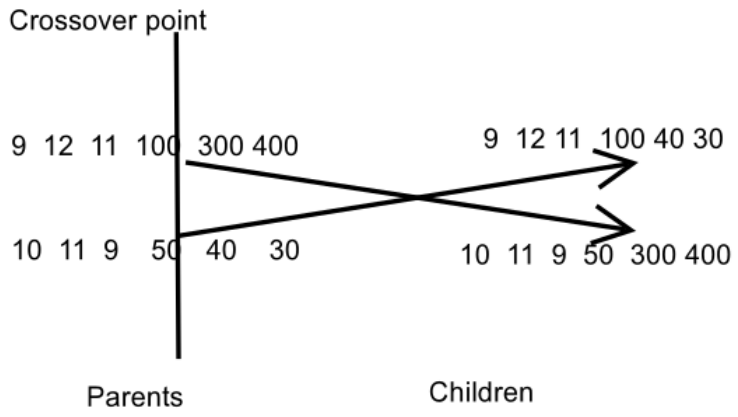
2.1.4 Διασταύρωση

Ο επόμενος γενετικός τελεστής που θα συζητήσουμε είναι εκείνος της διασταύρωσης. Στο σημείο αυτό, ο αλγόριθμός με διάφορες τεχνικές διασταύρωσης θα πρέπει να δημιουργήσει καινούρια σειρά χρωμοσωμάτων με βάση εκείνα των παλιών. Οι τεχνικές που θα συζητήσουμε είναι τρεις, η πιθανότητα διασταύρωσης, διασταύρωση ενός σημείου και ομοιόμορφη διασταύρωση. (18)

- **Πιθανότητα διασταύρωσης:** Υπάρχει η εξής πιθανότητα P_c η οποία καθορίζει το ποσοστό των παλαιών χρωμοσωμάτων που θα αντικατασταθούν από καινούρια με διασταύρωση και τα υπόλοιπα χρωμοσώματα θα αντιγραφούν σε επόμενη γενιά. Για παράδειγμα, έστω N ο πληθυσμός χρωματοσωμάτων και έστω $P_c = x$, $x \in [0\ 1]$, οπότε τα

χρωμοσώματα που θα αντικατασταθούν από καινούρια θα είναι $N \times P_c$. Σε περίπτωση που το ποσοστό των ατόμων που μένει σταθερό σε κάθε γενιά είναι μεγάλο, τότε υπάρχει περίπτωση να παρατηρηθεί ο εγκλωβισμός του γενετικού πληθυσμού σε κάποιο τοπικό ελάχιστο.

- **Διασταύρωση ενός σημείου:** Σε προηγούμενο κεφάλαιο αναφέραμε ότι τα χρωμοσώματα παριστάνονται με τη μορφή διανύσματος και τα γονίδια με τη μορφή ακεραίων ή δυαδικών αριθμών. Η διαδικασία που εκτελεί η διασταύρωση ενός σημείου συνήθως χρησιμοποιείται σε ακέραια χρωμοσώματα και δουλεύει ως εξής. Αρχικά, επιλέγεται τυχαία μια θέση στα χρωμοσώματα τα οποία πρόκειται να διασταυρωθούν και στη συνέχεια γίνεται μια ανταλλαγή του γενετικού υλικού που υπάρχει πριν και μετά από το σημείο που επιλέχθηκε στην αρχή (βλ. Εικόνα 2.2). Υπ' όψιν, ότι τα χρωμοσώματα δεν είναι απαραίτητο να έχουν το ίδιο μέγεθος.



Εικόνα 2.3: Διασταύρωση ενός σημείου.

- **Ομοιόμορφη διασταύρωση:** Στην συγκεκριμένη τεχνική διασταύρωσης, επιλέγονται δύο γονείς με στόχο να δημιουργηθούν δύο απόγονοι. Οι απόγονοι λαμβάνουν τα στοιχεία τους είτε από τον πρώτο είτε από τον δεύτερο γονέα με πιθανότητα 50% για κάθε στοιχείο. Εδώ, τα χρωμοσώματα απαιτείται να έχουν ίδιο μήκος.

Εικόνα 2.4: Ομοιόμορφη διασταύρωση.

2.1.5 Μετάλλαξη

Ο γενετικός τελεστής της μετάλλαξης προκαλεί μικρές αλλαγές στην τιμή ενός ή περισσότερων μεταβλητών του γονιδίου του πληθυσμού. Γενικά, βασίζεται στην χρήση ενός συντελεστή μεταλλάξεως που ανήκει στο διάστημα [0 1]. Όσο μεγαλύτερος είναι αυτός ο αριθμός τόσο μεγαλύτερη και η πιθανότητα να συμβεί μετάλλαξη σε ένα γονίδιο ενός χρωμοσώματος. Στην εικόνα 2.5 δίνεται ενδεικτικά ο αλγόριθμος της μετάλλαξης. (18)

```
for i=1..chromosome_count do
    for j=1..chromosome_size do
        r = rand(0,1)
        if (r<pm) then
            change(chromosome(i),j)
        end if
    end for
end for
```

Εικόνα 2.5: Ενδεικτικός αλγόριθμος μετάλλαξης.

2.2 Εισαγωγή στην γραμματική εξέλιξη

Η μέθοδος της γραμματικής εξελίξεως προτάθηκε από τους Conor Ryan, Michael O'Neil και JJ Colins το 1998, η οποία από τότε έχει εφαρμοστεί με επιτυχία σε διάφορους τομείς όπως στο χειρισμό ρομπότ, στις οικονομικές προβλέψεις, σε αναγνώριση τριγωνομετρικών ταυτοτήτων κ.α. Είναι μια νέα σχετικά μέθοδος εξελικτικού αλγορίθμου, η οποία παράγει προγράμματα σε οποιαδήποτε γλώσσα προγραμματισμού βασιζόμενη στην περιγραφή Backus-Naur Form (BNF) γραμματικής.

2.2.1 Γραμματική σε Backus-Naur Form (BNF)

Η μορφή BNF είναι ένα περιεχόμενο, το οποίο εκφράζει τη γραμματική μιας γλώσσας με τη μορφή παραγωγικών κανόνων. Γενικά, τέτοιου είδους γραμματικές αποτελούνται από τερματικά και μη τερματικά σύμβολα. Τα τερματικά σύμβολα είναι στοιχεία τα οποία εμφανίζονται στη γλώσσα όπως για παράδειγμα +,-,*,/ κ.α, και τα μη τερματικά σύμβολα είναι τα στοιχεία εκείνα τα οποία μπορούν να επεκταθούν σε άλλα μη τερματικά ή τερματικά σύμβολα. Παρακάτω, στην εικόνα 2.6 δίνεται ένα παράδειγμα μιας γραμματικής για εκφράσεις τύπου boolean.

```
(A)  <expr> ::= ( <expr> <biop> <expr> )      (0)
               |  <uop>  <expr>               (1)
               |  <bool>                      (2)

(B)  <biop> ::= and                           (0)
               | or                           (1)
               | xor                          (2)
               | nand                         (3)

(C)  <uop> ::= not

(D)  <bool> ::= true                          (0)
               | false                        (1)
```

Εικόνα 2.6: Γραμματική για εκφράσεις τύπου boolean.

Στην εικόνα 2.6 τα μη τερματικά σύμβολα είναι το <expr>, <biop>, <uop>, <bool> και τα τερματικά σύμβολα είναι and, or, xor, nand, not, true, false. (16) (24)

2.2.2 Λειτουργία της γραμματικής εξελίξεως

Η γραμματική εξέλιξη, όπως είπαμε και πιο πάνω, είναι μια μέθοδος εξελικτικού αλγορίθμου η οποία για να δουλέψει χρειάζεται δυο βασικά συστατικά: τη σχετική συνάρτηση καταλληλότητας και τη γραμματική της αντικειμενικής γλώσσας εκφρασμένη σε BNF. Η γραμματική καθορίζεται ως εξής G = (N,T,S,P), όπου N: είναι το σύνολο των μη τερματικών συμβόλων, T: είναι το πεπερασμένο σύνολο από τερματικά σύμβολα με τον περιορισμό ότι N∩T=0, S: είναι το αρχικό σύμβολο (S ϵ N) και P: είναι το πεπερασμένο σύνολο των παραγωγικών κανόνων της μορφής A→a ή A→aB (A,B ϵ N και a ϵ T). Η γραμματική εξέλιξη, είναι μια μορφή γενετικού προγραμματισμού όπου τα χρωμοσώματα εκφράζονται με τη μορφή ακεραίων διανυσμάτων, όπου κάθε στοιχείο αυτών εκπροσωπεί έναν παραγωγικό κανόνα που δίνεται από τη γραμματική σε BNF.

Γενικά, η όλη η διαδικασία ξεκινάει από το αρχικό σύμβολο, που αναφέραμε πιο πάνω, της γραμματικής και παράγει το αντίστοιχο πρόγραμμα αντικαθιστώντας όλα τα μη τερματικά σύμβολα με το δεξιό τμήμα του κανόνα παραγωγής που έχει επιλεχθεί. Η επιλογή του κανόνα παραγωγής απαιτεί τα εξής δυο βήματα για να πραγματοποιηθεί, πρώτον διαβάζει το στοιχείο του χρωμοσώματος, ας το θέσουμε με τη τιμή V και δεύτερον η επιλογή του κανόνα παραγωγής σύμφωνα με την εξής ισότητα Rule = V mod R , όπου R είναι ο αριθμός των παραγωγικών κανόνων για το τρέχων μη τερματικό σύμβολο. Η διαδικασία που αναφέραμε πιο πάνω γίνεται επαναληπτικά έως ότου έχουν αντικατασταθεί πλέον όλα τα μη τερματικά σύμβολα με τερματικά, πράγμα που σημαίνει ότι έχει παραχθεί

ένα έγκυρο πρόγραμμα στην αντικειμενική γλώσσα προγραμματισμού, ή να έχουν διαβαστεί όλα τα στοιχεία του χρωμοσώματος χωρίς βεβαία να έχουν αντικατασταθεί όλα τα μη τερματικά με τερματικά σύμβολα. Στη δεύτερη περίπτωση εφαρμόζεται το γεγονός περιτυλίγματος (wrapping event), δηλαδή ξαναδιαβάζεται το χρωμόσωμα από την αρχή και όταν το γεγονός περιτυλίγματος φτάσει στο ανώτατο όριο των φορών (συνήθως μια η δυο φορές είναι αρκετές) που θα εκτελεστεί και δεν έχει παραχθεί έκφραση χωρίς μη τερματικά σύμβολα, τότε το χρωμόσωμα θεωρείται άκυρο.

Ένα απλό παράδειγμα είναι το εξής, με βάση την εικόνα 2.6 ας υποθέσουμε ότι το μη τερματικό σύμβολο που έχουμε είναι το <biop> , και ας υποθέσουμε ότι V = 6 και αφού το <biop> αποτελείται από τέσσερις κανόνες παραγωγής, τότε έχουμε R = 4. Με βάση την ισότητα Rule = V mod R, έχουμε το εξής Rule = 6 mod 4, δηλαδή Rule = 2. Συνεπώς, το μη τερματικό σύμβολο <biop> θα αντικατασταθεί από το τερματικό σύμβολο xor. (24)

2.3 Εισαγωγή στα κατασκευαζόμενα τεχνητά νευρωνικά δίκτυα

Σε ένα κατασκευαζόμενο τεχνητό νευρωνικό δίκτυο, για την ορθή επίλυση ενός προβλήματος, σημαντικό ρόλο παίζει η αρχιτεκτονική του δικτύου. Ο προγραμματιστής θα πρέπει να είναι σε θέση να γνωρίζει τον αριθμό επιπέδων που θα πρέπει να αποτελείται το δίκτυο, όπως επίσης θα πρέπει να μπορεί να κάνει και τις ανάλογες προσθέσεις και αφαιρέσεις κόμβων, για να μπορέσει έτσι να αυξήσει όσο το δυνατόν την αποδοτικότητα του δικτύου. Εξίσου σημαντικό βέβαια με την αρχιτεκτονική του δικτύου, ίσως πιο σημαντικό, είναι και ο αλγόριθμος μάθησης αυτού. Οι αλγόριθμοι εκπαίδευσης διαφέρουν μεταξύ τους με τον τρόπο με τον οποίο γίνεται η ρύθμιση σε ένα συναπτικό βάρος ενός νευρώνα. Γενικά, τα τεχνητά νευρωνικά δίκτυα έχουν την ικανότητα να μαθαίνουν από το περιβάλλον τους και να βελτιώνουν την απόδοσή τους μέσω της μάθησης, και σε κάθε

επανάληψη το δίκτυο αποκτά περισσότερη γνώση.

2.4 Χρήση κατασκευαζόμενων τεχνητών νευρωνικών δικτύων σε εφαρμογές

Με την πάροδο του χρόνου, τα νευρωνικά δίκτυα όλο και πιο πολύ εισβάλλουν στην ζωή μας με σκοπό πάντα να κάνουν την καθημερινότητά μας πιο εύκολη. Τα νευρωνικά δίκτυα καλύπτουν μια αρκετά μεγάλη γκάμα εφαρμογών όπως η ιατρική, η πρόβλεψη καιρού, σε γεωλογικές έρευνες, ρομποτική κ.τ.λ. Παρακάτω, γίνεται μια μικρή ανάλυση δυο εφαρμογών που βασίζονται σε κατασκευαζόμενα τεχνητά νευρωνικά δίκτυα.

2.4.1 Αναγνώριση προτύπων

Χρησιμοποιείται αρκετά στη βιομηχανία όπως για παράδειγμα στον αυτόματο έλεγχο βιομηχανικών προϊόντων, στον αυτοματισμό της παραγωγής και στην ποιότητα του τελικού προϊόντος. Ένα απλό παράδειγμα θα μπορούσε να είναι το εξής: ας υποθέσουμε ότι έχουμε ένα εργοστάσιο το οποίο εξάγει 2 ειδών φρούτα όπως μήλα και μπανάνες, και ας υποθέσουμε ότι θέλουμε να βάλουμε σε ένα κουτί τα μήλα και σε ένα άλλο κουτί τις μπανάνες. Υποθέτω, μια πιθανή λύση του προβλήματος αυτού θα ήταν η ύπαρξη ενός συστήματος, το οποίο θα "έβλεπε" το φρούτο, και ανάλογα με τις πληροφορίες που θα λάμβανε (όπως για παράδειγμα το χρώμα και το μέγεθος) θα μπορούσε να κάνει και την κατάλληλη κατηγοριοποίηση. Φυσικά, η υλοποίηση αυτού του παραδείγματος στην πραγματικότητα είναι πολύ πιο σύνθετη και οι προγραμματιστές θα πρέπει να λάβουν υπ΄όψιν και άλλους παράγοντες οι οποίοι δεν αναφέρονται εδώ.

2.4.2 Επίλυση διαφορικών εξισώσεων

Στις θετικές επιστήμες υπάρχουν αρκετά προβλήματα που μπορούν να εκφραστούν με τη μορφή διαφορικών εξισώσεων. Μερικά από τα προβλήματα αυτά είναι ο ρυθμός με τον οποίο λιώνει ένα κομμάτι πάγου, η εκτίμηση κέρδους σε αγοραπωλησίες μετοχών κ.α. Για την επίλυση διαφορικών εξισώσεων, ο χρήστης παίρνει διαφορικές εξισώσεις από το πεδίο ορισμού και με τον τρόπο αυτό δημιουργεί τα δεδομένα εκπαίδευσης και τα δεδομένα ελέγχου. Έπειτα, σε κάθε χρωμόσωμα υπολογίζεται η καταλληλότητα και με τη βοήθεια της γραμματικής εξελίξεως παράγεται μια υποψήφια λύση. Κάθε υποψήφια λύση γίνονται καλύτερες χρησιμοποιώντας μια διαδικασία βελτιστοποίησης. (20)

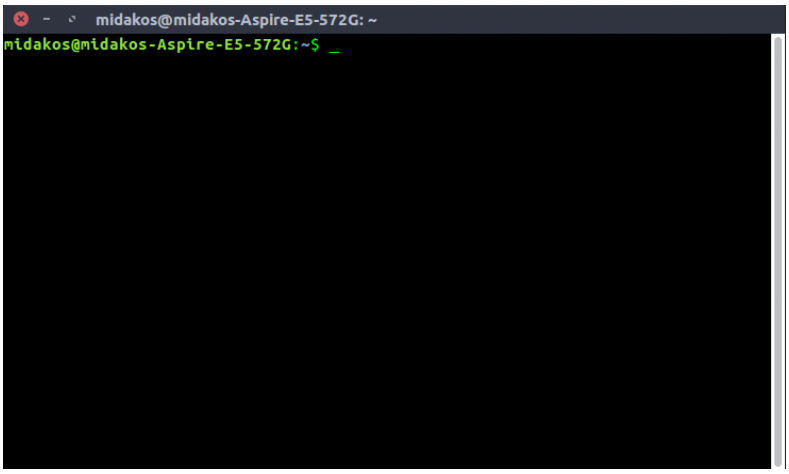
3. Εφαρμογές σε QT

Το Qt είναι μια πλατφόρμα ανάπτυξης λογισμικού, όπου συνήθως ο χρήστης του λογισμικού αυτού, το χρησιμοποιεί για τη δημιουργία κλασικών και ενσωματωμένων γραφικών διεπαφών χρήστη. Το Qt αναπτύσσεται μέχρι και σήμερα τόσο από την εταιρία Qt, όσο και από μεμονωμένους προγραμματιστές και επιχειρήσεις υπό τη διαχείριση ανοιχτού κώδικα με σκοπό την προώθηση του Qt.

3.1 Εγκατάσταση Qt Creator

Το Qt Creator είναι ένα ολοκληρωμένο περιβάλλον ανάπτυξης (IDE) γραμμένο σε C++, το οποίο ανήκει σε μια "εργαλειοθήκη" ανάπτυξης λογισμικού (SDK) και χρησιμοποιείται για ανάπτυξη εφαρμογών. Το Qt Creator χρησιμοποιεί τον μεταγλωττιστή C++ από τη συλλογή του GNU Compiler σε Linux και FreeBSD, ενώ στα Windows το Qt Creator, μπορεί να χρησιμοποιήσει το MinGW ή το MSVC με την προεπιλεγμένη εγκατάσταση. Η εφαρμογή έχει υλοποιηθεί σε υπολογιστή με λειτουργικό σύστημα Ubuntu 16.04 (Linux) συνεπώς το κεφάλαιο 3.1 πραγματεύεται την εγκατάσταση του Qt Creator σε λειτουργικό των Linux.

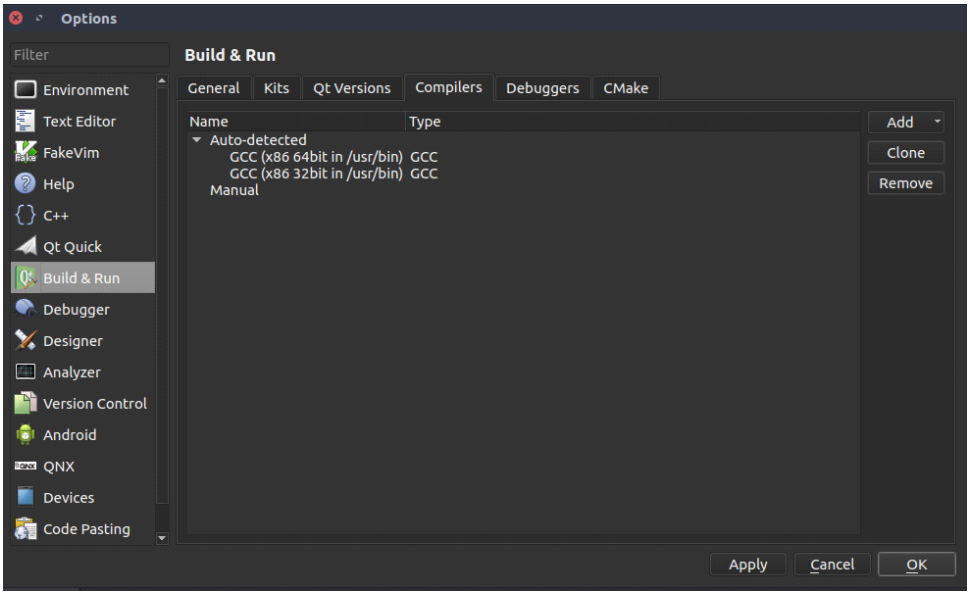
Η εγκατάσταση, επιλέγω να γίνει από τη γραμμή εντολών (cmd) του υπολογιστή· με τον τρόπο αυτό αποφεύγουμε τη σπατάλη χρόνου ψάχνοντας σε διάφορες ιστοσελίδες στο διαδίκτυο. Το παράθυρο της γραμμής εντολών ανοίγει πληκτρολογώντας τα πλήκτρα Ctrl+Alt+T ταυτόχρονα, και στην επιφάνεια εργασίας εμφανίζεται ένα παράθυρο αντίστοιχο με αυτό της εικόνας 3.1. Έπειτα, θα προχωρήσουμε στην εγκατάσταση κάποιων βασικών πακέτων, όπως για παράδειγμα τον μεταγλωττιστή gcc, δίνοντας στη γραμμή εντολών την εξής εντολή: sudo apt-get install build-essential.



Εικόνα 3.1: Γραμμή εντολών

Στη συνέχεια εγκαθιστούμε το qt-creator δίνοντας διαδοχικά τις εξής δύο εντολές : sudo apt-get install qtcreator και sudo apt-get install qt5-default και μετά μπορούμε προαιρετικά να δώσουμε την εντολή sudo apt-get update για να κάνουμε ενημέρωση στο σύστημά μας. Τέλος, αφού οι εντολές έχουν εκτελεστεί με επιτυχία ανοίγουμε το Qt Creator και κάνουμε τον απαραίτητο έλεγχο για να δούμε αν έχει γίνει σωστά η εγκατάσταση των μεταγλωττιστών. Επιλέγουμε από το οριζόντιο μενού Tools > Options > Build & Run > Compilers και θα πρέπει στο παράθυρο που θα ανοίξει να υπάρχει κάτι αντίστοιχο με αυτό

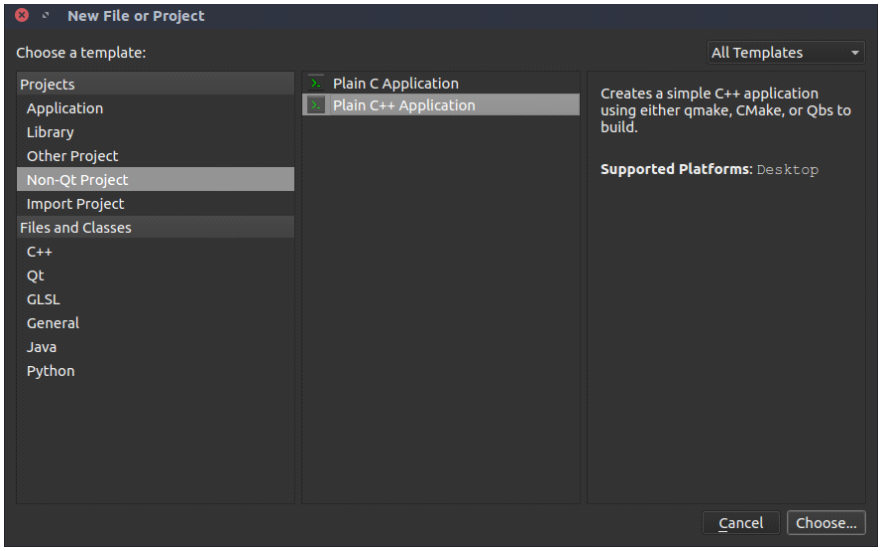
της εικόνας 3.2. Για περισσότερες πληροφορίες, μπορούμε να κατεβάσουμε το αντίστοιχο εγχειρίδιο (documentation) του Qt πληκτρολογώντας στη γραμμή εντολών την εντολή: sudo apt-get install qt5-doc. (1) (3) (4)



Εικόνα 3.2: Εμφάνιση μεταγλωττιστών στο περιβάλλον Qt.

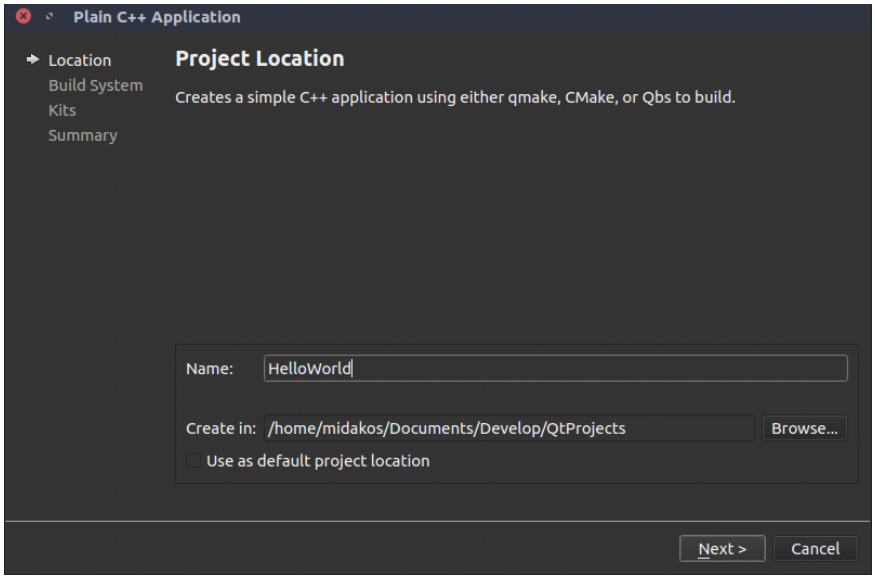
3.2 Παραδείγματα εφαρμογών κονσόλας σε Qt

Εφόσον η εγκατάσταση του Qt Creator έχει ολοκληρωθεί με επιτυχία, θα ξεκινήσουμε δίνοντας κάποια απλά παραδείγματα γραμμένα στην γλώσσα C++. Για την δημιουργία νέου Project επιλέγουμε από το οριζόντιο μενού File > New File or Project όπου εμφανίζεται το παράθυρο της εικόνας 3.3. Στη φάση αυτή, από τη δεξιά στήλη επιλέγουμε το Non-Qt Project, και από τη μεσαία στήλη επιλέγουμε το Plain C++ Application και τέλος κάτω δεξιά δίνουμε Choose.



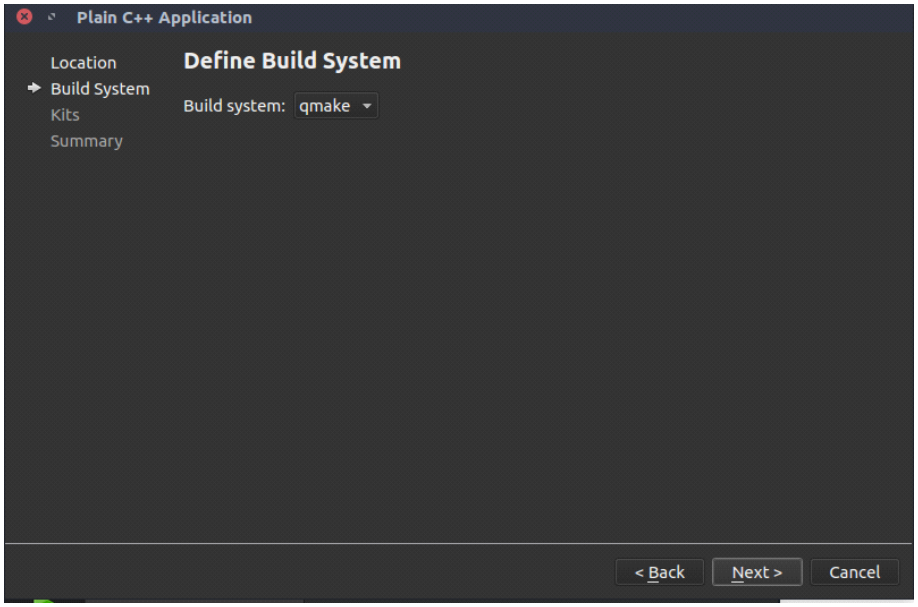
Εικόνα 3.3: Επιλογή είδος του έργου.

Στη συνέχεια στο παράθυρο, αντίστοιχο με αυτό της εικόνας 3.4, που ακολουθεί, δίνουμε το όνομα που θέλουμε να έχει το έργο μας, όπως για παράδειγμα “HelloWorld”, καθώς και το μονοπάτι (path name) που θέλουμε να αποθηκευτεί το έργο μέσα στον υπολογιστή.

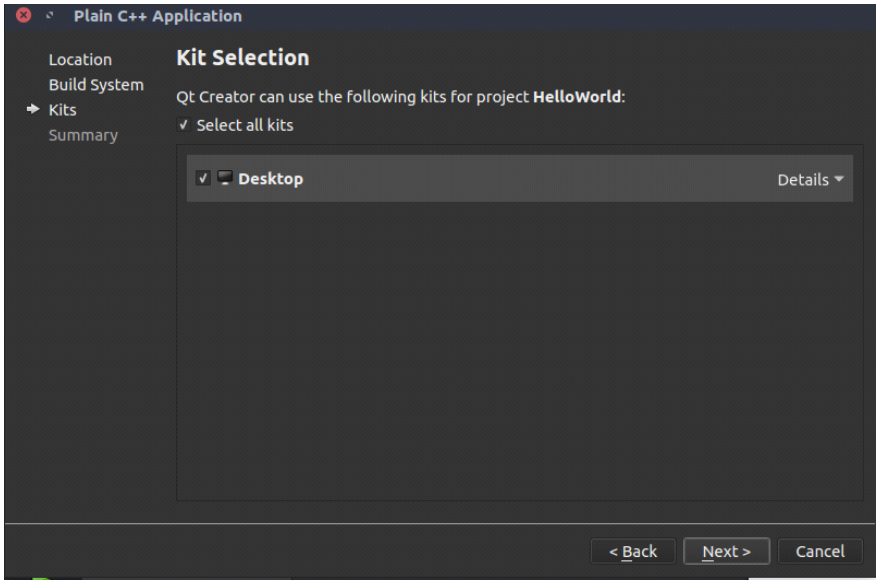


Εικόνα 3.4: Ονομασία και καταχώρηση του έργου στον υπολογιστή.

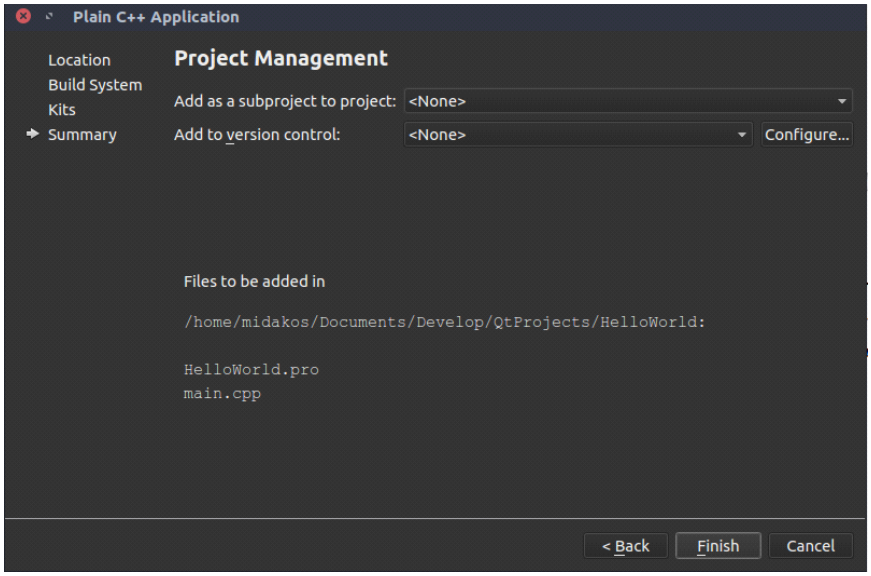
Έπειτα, πατάμε Next στα διαδοχικά εμφανιζόμενα παράθυρα (βλ εικόνες 3.5 και 3.6) έως ότου φτάσουμε στο παράθυρο της εικόνας 3.7 όπου εκεί βλέπουμε συνοπτικά όλες τις προηγούμενες ρυθμίσεις που έχουμε κάνει, και πατάμε Finish.



Εικόνα 3.5: Στιγμιότυπο δημιουργίας έργου.

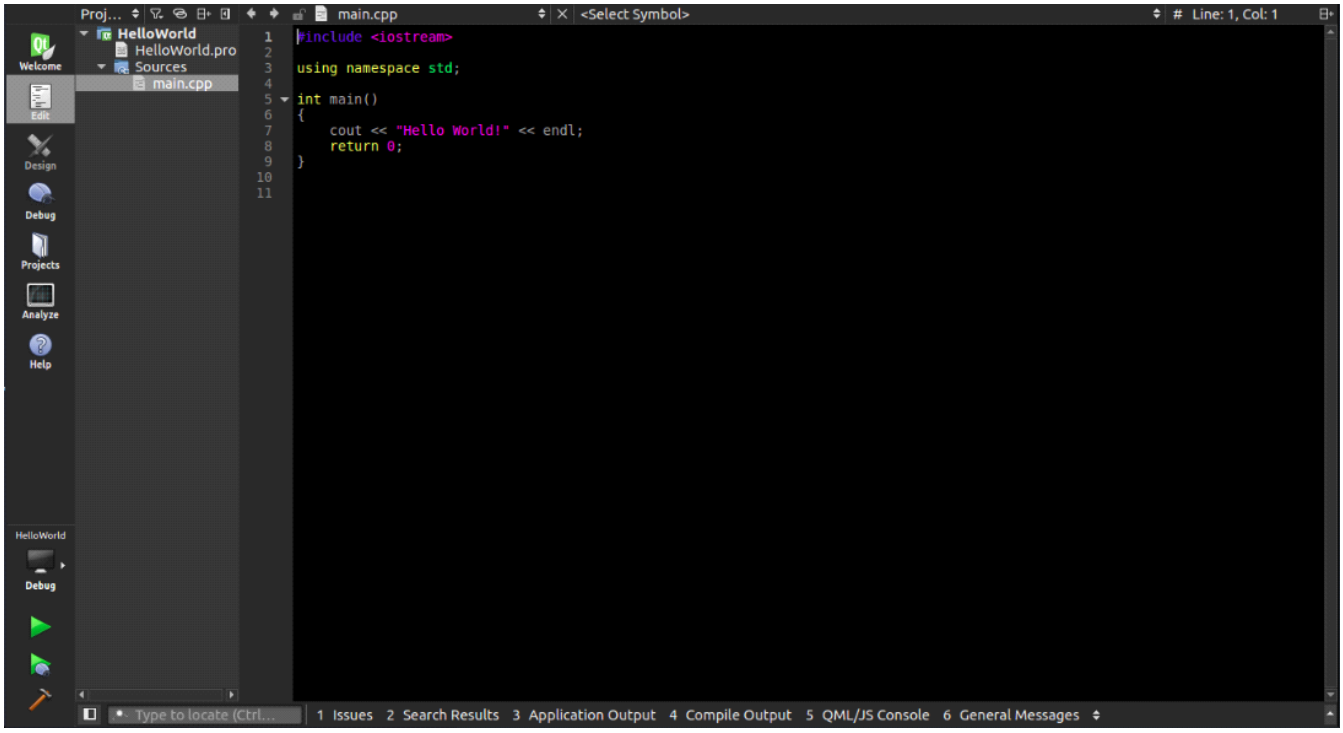


Εικόνα 3.6: Στιγμιότυπο δημιουργίας έργου.

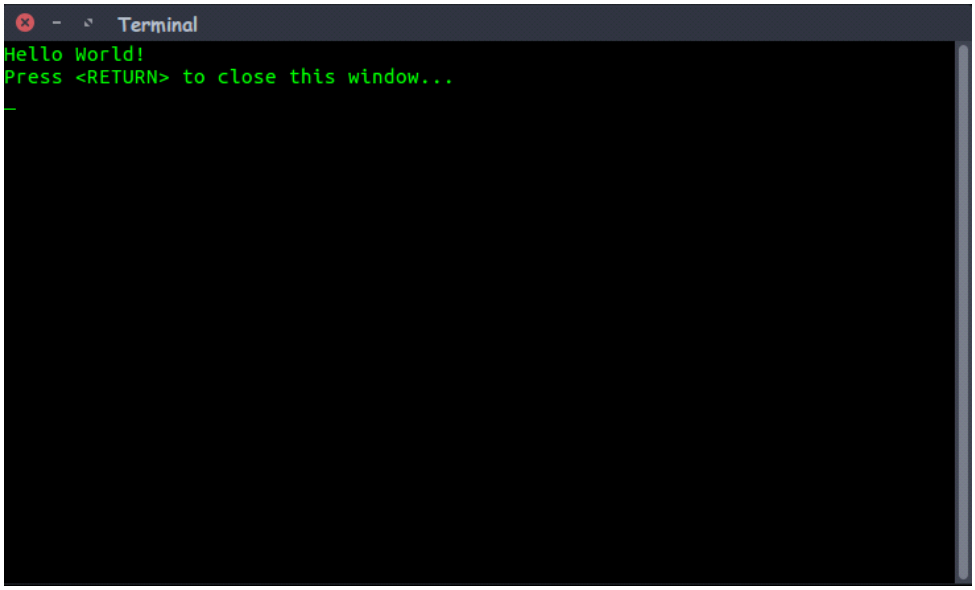


Εικόνα 3.7: Σύνοψη ρυθμίσεων του έργου.

Παρακάτω, στην εικόνα 3.8 βλέπουμε το περιβάλλον για την ανάπτυξη κώδικα σε C++ μαζί με ένα έτοιμο μικρό κομμάτι κώδικα όπου εμφανίζει το μήνυμα “Hello World!” στην κονσόλα. Ο κώδικας εκτελείται πατώντας το πράσινο τρίγωνο κάτω αριστερά στην οθόνη και το αποτέλεσμα της εφαρμογής αυτής φαίνεται στην εικόνα 3.9.

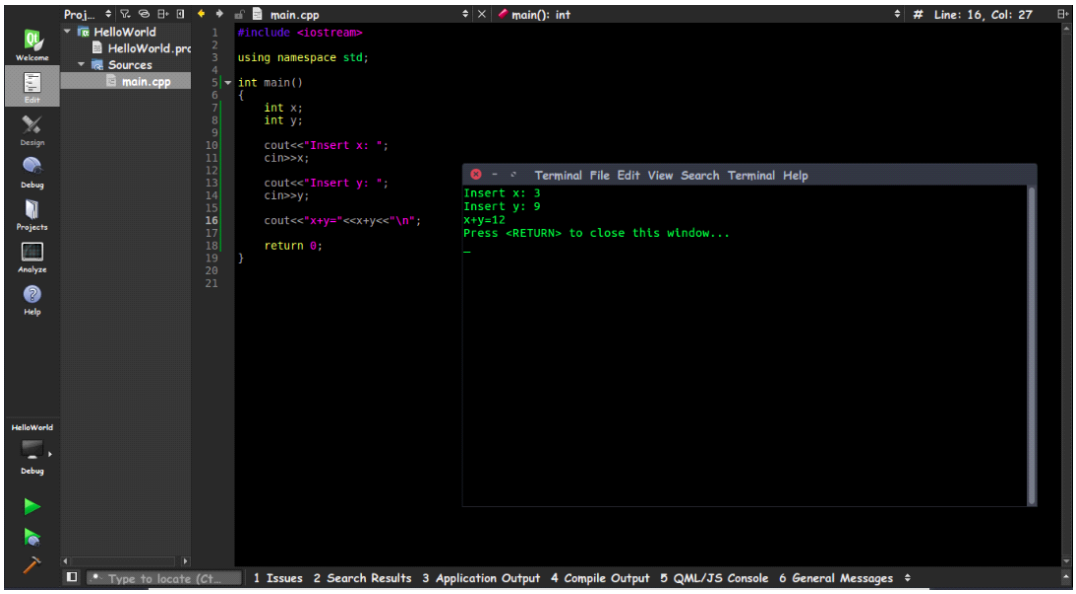


Εικόνα 3.8: Περιβάλλον ανάπτυξης κώδικα Qt Creator.



Εικόνα 3.9: Εμφάνιση μηνύματος “Hello World!” στην κονσόλα.

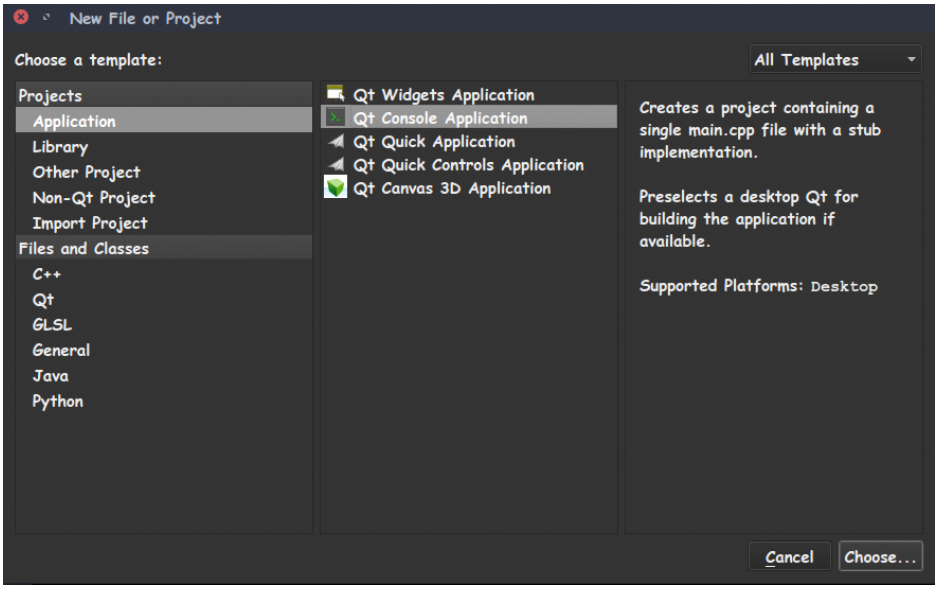
Ένα άλλο μικρό παράδειγμα κώδικα, είναι η υλοποίηση μιας εφαρμογής, όπου δέχεται σαν είσοδο δύο ακέραια νούμερα και εμφανίζει στην κονσόλα το άθροισμά τους (βλ. Εικόνα 3.10). Ξεκινάμε με τη δήλωση της βιβλιοθήκης iostream πληκτρολογώντας την εντολή #include <iostream> (γραμμή 1), στη γραμμή 3 χρησιμοποιούμε την εντολή using namespace std;, όπου αυτό μας δίνει τη δυνατότητα να χρησιμοποιούμε τις εντολές cin και cout. Χωρίς την εντολή της γραμμής 3, θα πρέπει αντί για cout και cin να γράψουμε std::cout και std::cin για να μπορέσουν να εκτελεστούν οι εντολές. Στη συνέχεια, δηλώνονται οι ακέραιοι x, y (γραμμές 7 και 8), έπειτα με την εντολή cout εμφανίζουμε τα μηνύματα στην κονσόλα που βρίσκονται μέσα στα διπλά εισαγωγικά (“), καθώς και το άθροισμα των δύο ακεραίων (γραμμές 10, 13 και 16). Τέλος, με την εντολή cin εισάγουμε τα νούμερα που έδωσε ο χρήστης, δηλαδή, το 3 και το 9 στις μεταβλητές x και y αντίστοιχα.



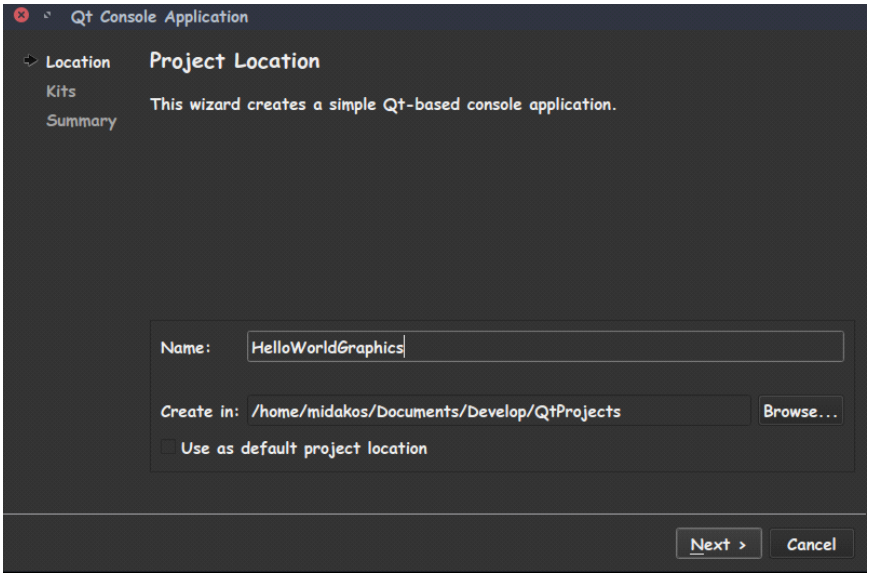
Εικόνα 3.10: Παράδειγμα εφαρμογής κονσόλας σε Qt.

3.3 Παραδείγματα γραφικών εφαρμογών σε Qt

Ξεκινάμε επιλέγοντας από το οριζόντιο μενού File > New File or Project, από την αριστερή στήλη επιλέγουμε το Application και από τη μεσαία το Qt Console Application (βλ. Εικόνα 3.11) και δίνουμε Choose. Δίνουμε ένα όνομα στο έργο μας, καθώς και την τοποθεσία που πρόκειται να αποθηκευτεί στον υπολογιστή μας (βλ. Εικόνα 3.12) και επιλέγουμε Next στις διαδοχικά εμφανιζόμενες εικόνες, παρόμοια διαδικασία με αυτή του κεφαλαίου 3.2, (βλ. Εικόνες 3.4, 3.5, 3.6, 3.7) και τέλος επιλέγουμε Finish.

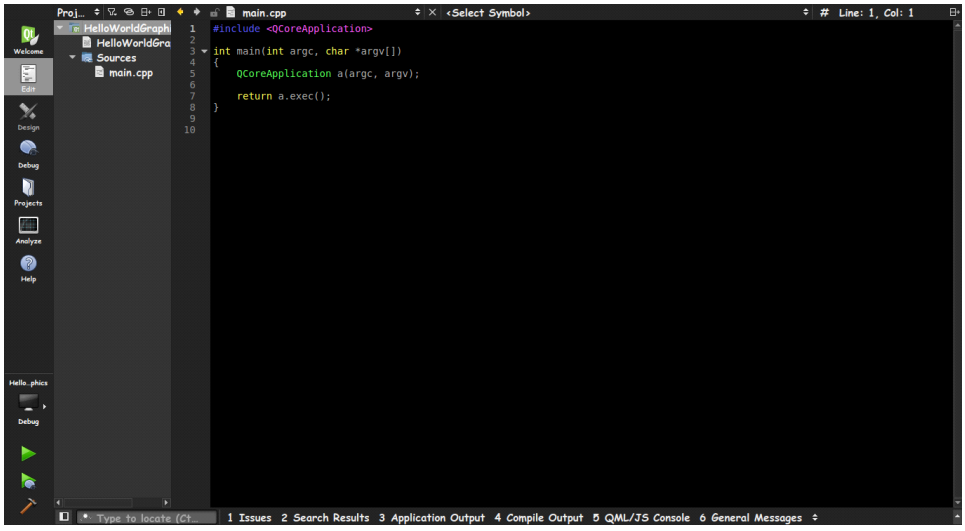


Εικόνα 3.11: Επιλογή είδους του έργου.



Εικόνα 3.12: Ονομασία και καταχώρηση του έργου στον υπολογιστή.

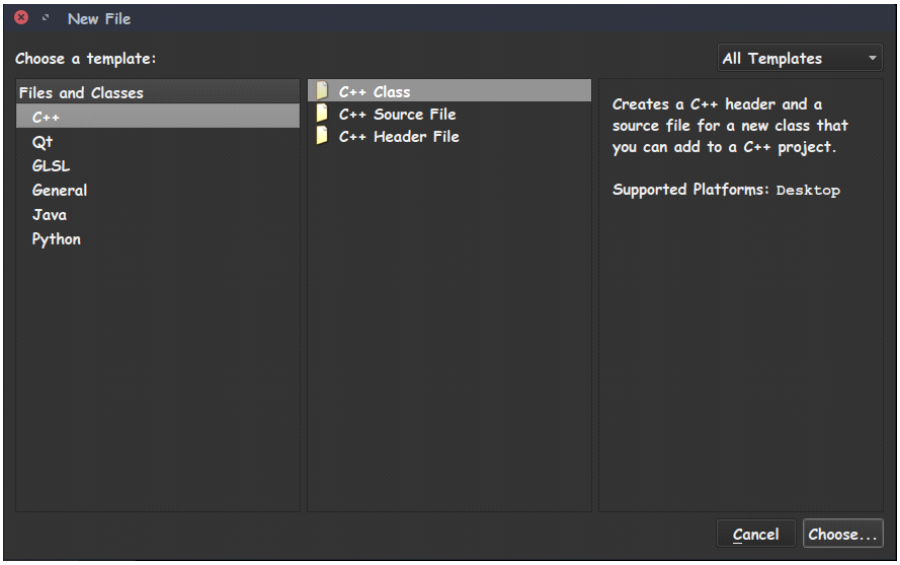
Πλέον, έχουμε έτοιμο το περιβάλλον για ανάπτυξη κώδικα με εισαγωγή γραφικών (Εικόνα 3.13). Στη παράγραφο αυτή θα δουλέψουμε το παράδειγμα του κεφαλαίου 3.2 (βλ. εικόνα 3.10). Θα μπορούσαμε να υλοποιήσουμε τα εξής: ένα παράθυρο, δύο πεδία όπου εκεί ο χρήστης θα εισάγει τους δύο ακεραίους, ένα επιπλέον πεδίο όπου εκεί θα εμφανίζεται το αποτέλεσμα της πρόσθεσης και ίσως κάποιο κουμπί για να εκτελείται η πρόσθεση.



Εικόνα 3.13: Περιβάλλον ανάπτυξης για δημιουργία κώδικα με γραφικά

Κάνουμε δεξί κλικ πάνω στο φάκελο source και επιλέγουμε Add New, στο παράθυρο που εμφανίζεται, στη δεξιά στήλη επιλέγουμε το C++ και στη μεσαία στήλη το C++ Source File

(βλ. Εικόνα 3.14) και δίνουμε ένα όνομα π.χ mainWindow και αμέσως έχουμε τη δημιουργία του αρχείου mainwindow.cpp , επαναλαμβάνουμε τη παραπάνω διαδικασία με τη διαφορά ότι στη μεσαία στήλη επιλέγουμε το C++ Header File, δίνουμε ένα όνομα (κατά προτίμηση το ίδιο με αυτό που δημιουργήσαμε και πριν) και έχουμε τη δημιουργία φακέλου Headers με το αρχείο mainwindow.h μέσα σε αυτό.



Εικόνα 3.14: Δημιουργία source file και header file.

Στο αρχείο mainwindow.h, δηλώνουμε τις μεταβλητές και τις αντίστοιχες βιβλιοθήκες που πρόκειται να χρησιμοποιήσουμε, (εδώ κάποιοι ίσως αντιμετωπίσουν το εξής πρόβλημα), κάποιες βιβλιοθήκες ίσως να μην αναγνωρίζονται. Η λύση σε αυτό είναι να πάμε στο αρχείο με κατάληξη .pro (στο παράδειγμά μας είναι το HelloWorldGraphics.pro) και να γράψουμε την εξής εντολή: qt += widgets, με τον τρόπο αυτό το σύστημα αναγνωρίζει όλα τα πιθανά οπτικά συστατικά που πρόκειται να χρησιμοποιήσουμε καθώς και τις αντίστοιχες βιβλιοθήκες τους. Στην εικόνα 3.15 φαίνονται οι δηλώσεις μεταβλητών και οι βιβλιοθήκες τους. Στη γραμμή 13 έχουμε φτιάξει μια κλάση με το όνομα MainWindow που κληρονομεί στοιχεία απο την κλάση QMainWindow (:public QMainWindow), στις γραμμές 16 έως 18 δηλώνουμε τον κατασκευαστή και μια μεταβλητή τύπου integer (ακέραιος), στις γραμμές 20 έως 25 είναι τα οπτικά συστατικά που έχουμε βάλει και στην γραμμή 28 είναι η συνάρτηση όπου εκτελεί την πράξη της πρόσθεσης και εμφανίζει το αποτέλεσμα. Δεν θα μπω σε βαθύτερη ανάλυση του τι σημαίνει το καθένα από τα οπτικά συστατικά που έχουν δηλωθεί γιατί υπάρχει ο κίνδυνος να κουράσω τον αναγνώστη· το μόνο που πιστεύω ότι αξίζει να σημειωθεί εδώ είναι ότι τα οπτικά συστατικά (π.χ κουμπιά, πεδία εισαγωγής των ακεραίων από τον χρήστη κ.α) “πατάνε” πάνω σε μια διάταξη, το mainLayout, που είναι τύπου QVBoxLayout όπου αυτό σημαίνει ότι όλα τα οπτικά

συστατικά που “πατάνε” πάνω σε αυτό έχουν κάθετη διάταξη μεταξύ τους και η διάταξη (δηλ. το mainLayout) με τη σειρά της “πατάει” πάνω στην κλάση QWidget όπου εκεί “πατάνε” όλα τα υπόλοιπα οπτικά συστατικά.

```
1  #ifndef MAINWINDOW
2  #define MAINWINDOW
3
4  #include <QObject>
5  #include <QMainWindow>
6  #include <QVBoxLayout>
7  #include <QLineEdit>
8  #include <QWidget>
9  #include <QPushButton>
10 #include <QLineEdit>
11 #include <QLabel>
12
13 class MainWindow:public QMainWindow{
14     Q_OBJECT
15
16 public:
17     MainWindow(QWidget *parent = 0);
18     int result;
19
20 private:
21     QWidget *mainWidget;
22     QVBoxLayout *mainLayout;
23     QLineEdit *valueLineEdit1, *valueLineEdit2, *resultsLineEdit;
24     QPushButton *sumButton;
25     QLabel *value1Label, *value2Label, *resultLabel;
26
27 public slots:
28     void makeAddition();
29
30 };
31
32
33
34 #endif // MAINWINDOW
35
36
```

Εικόνα 3.15: Ο κώδικας από το αρχείο mainwindow.h .

Έχοντας τελειώσει με τις δηλώσεις μεταβλητών και οπτικών συστατικών πηγαίνουμε το αρχείο mainwindow.cpp και εκεί γράφουμε τον κώδικα που χρειάζεται για να στηθεί το γραφικό περιβάλλον και η λειτουργικότητα της εφαρμογής. Για να μπορέσουμε να χρησιμοποιήσουμε αυτά που δηλώσαμε στο αρχείο mainwindow.h, θα πρέπει να το εισάγουμε στο αρχείο mainwindow.cp· αυτό γίνεται με την εξής εντολή στη γραμμη 1: #include <mainwindow.h>. Από τη γραμμή 3 μέχρι τη γραμμή 41, είναι ο κώδικας του

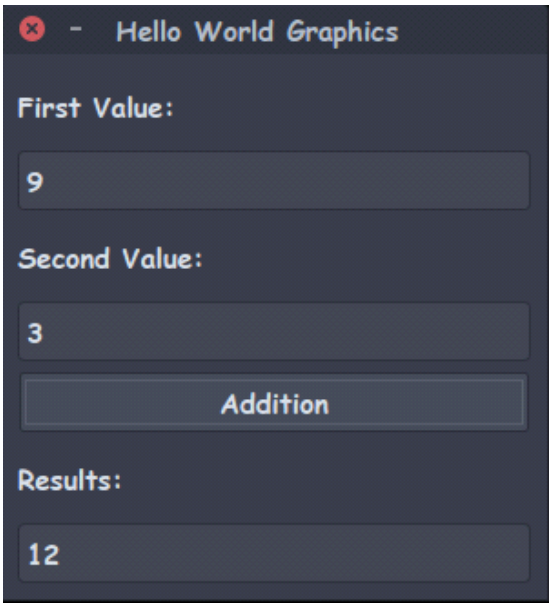
κατασκευαστή της κλάσης MainWindow (βλ. Εικόνα 3.16). Αναλυτικότερα, στις γραμμές 5 και 6 έχουμε τη δημιουργία παραθύρου με διαστάσεις 300x300 και τον τίτλο αυτού, έπειτα στις γραμμές 8 έως 10 και 12 έως 13 είναι η δημιουργία της διάταξης (Layout) και QWidget. Στη συνέχεια, γραμμές 15 έως 17, 21 έως 23 και 32 έως 34 είναι η δήλωση των ετικετών (QLabel), στις γραμμές 18 έως 19 και 24 έως 25, δημιουργούμε τις περιοχές που δίνει ο χρήστης τους δύο ακεραίους και τα τοποθετούμε στη διάταξή μας. Παρακάτω, γραμμές 28 έως 30 είναι η δημιουργία του κουμπιού και η τοποθέτησή του στη διάταξη, και καταλήγουμε στη δήλωση του τελευταίου οπτικού συστατικού, της περιοχής εμφάνισης του αποτελέσματος της πρόσθεσης (QLineEdit) γραμμές 35 έως 37. Τέλος, έχουμε τη γραμμή 39 όπου εκεί μόλις πατηθεί το κουμπί από τον χρήστη θα εκτελεστεί ο κώδικας που βρίσκεται στη συνάρτηση makeAddition (Εικόνα 3.17). Στη συνάρτηση αυτή "παίρνουμε" τα νούμερα που έχει δώσει στα πεδία ο χρήστης προηγουμένως (γραμμές 43 και 44), γίνεται η πρόσθεση ,όπου και το αποτέλεσμα της πράξης εμφανίζεται στο τρίτο πεδίο.

```cpp
#include <mainwindow.h>

MainWindow::MainWindow (QWidget *parent):QMainWindow(parent){

    setFixedSize(300,300);
    setWindowTitle("Hello World Graphics");

    mainWidget = new QWidget;
    setCentralWidget(mainWidget);
    mainWidget->setFixedSize(this->width(),this->height());

    mainLayout = new QVBoxLayout;
    mainWidget->setLayout(mainLayout);

    value1Label = new QLabel;
    value1Label->setText("First Value:");
    mainLayout->addWidget(value1Label);
    valueLineEdit1 = new QLineEdit;
    mainLayout->addWidget(valueLineEdit1);

    value2Label = new QLabel;
    value2Label->setText("Second Value:");
    mainLayout->addWidget(value2Label);
    valueLineEdit2 = new QLineEdit;
    mainLayout->addWidget(valueLineEdit2);


    sumButton = new QPushButton();
    sumButton->setText("Addition");
    mainLayout->addWidget(sumButton);

    resultLabel = new QLabel;
    resultLabel->setText("Results:");
    mainLayout->addWidget(resultLabel);
    resultsLineEdit = new QLineEdit;
    resultsLineEdit->setReadOnly(true);
    mainLayout->addWidget(resultsLineEdit);

    connect(sumButton,SIGNAL(clicked(bool)),this,SLOT(makeAddition()));
```

Εικόνα 3.16: Ο κώδικας από το αρχείο mainwindow.cpp.

```cpp
void MainWindow::makeAddition(){
    int x = valueLineEdit1->text().toInt();
    int y = valueLineEdit2->text().toInt();
    result = x + y;
    resultsLineEdit->setText(QString::number(result));
}
```

Εικόνα 3.17: Ο κώδικας από το αρχείο mainwindow.cpp.



Εικόνα 3.18: Παράδειγμα της εφαρμογής με γραφικό περιβάλλον.

3.4 Παραδείγματα εφαρμογών βάσεων δεδομένων σε QT

Οι βάσεις δεδομένων είναι μια οργανωμένη συλλογή διαφόρων τύπων δεδομένων όπου εξυπηρετούν ένα συγκεκριμένο σκοπό. Η αναζήτηση ενός στοιχείου ή αλλιώς ενός δεδομένου μπορεί να γίνει πολύ εύκολα και γρήγορα κάτω από συγκεκριμένες συνθήκες· για παράδειγμα ας υποθέσουμε ότι έχουμε καταγεγραμμένους από όλους τους Έλληνες πολίτες το όνομα, το επίθετο, τον αριθμό ταυτότητας και τον αριθμό τηλεφώνου τους και θέλουμε να βρούμε το ονοματεπώνυμο εκείνο που έχει αριθμό ταυτότητας “4”. Εδώ, το ονοματεπώνυμο αποτελεί το δεδομένο που ψάχνουμε και ο αριθμός ταυτότητας “4” συνιστά τη συνθήκη κάτω από την οποία ψάχνουμε το ονοματεπώνυμο αυτό.

Ας προσπαθήσουμε, λοιπόν, να υλοποιήσουμε στο Qt Creator το παραπάνω παράδειγμα δημιουργώντας μια βάση δεδομένων στην οποία ο χρήστης θα μπορεί να κάνει εισαγωγή,

διαγραφή, και εμφάνιση όλων των στοιχείων καθώς και αναζήτηση συγκεκριμένων στοιχείων κάτω από ορισμένες συνθήκες. Σκέφτομαι, να υπάρχει και ένα απλό γραφικό περιβάλλον όμοιο με αυτό του παραδείγματος στο κεφάλαιο 3.3, ώστε να είναι εύχρηστο για τον χρήστη. Δεν θα μπω σε πολύ βαθειά ανάλυση της δημιουργίας του γραφικού περιβάλλοντος, γιατί υπάρχει μια μικρή τέτοια ανάλυση στο κεφάλαιο 3.3. Επίσης, θεωρώ πως ξεφεύγει από τα όρια του κεφαλαίου 3.4. Τέλος, θα πρέπει να προσθέσω το εξής: κάποιες βασικές λειτουργίες, όπου σε μια πραγματική βάση δεδομένων η μη υλοποίηση τους θα ήταν αδιανόητη, λείπουν γιατί ο στόχος του κεφαλαίου αυτού είναι να δείξει στον αναγνώστη την αλληλεπίδραση του χρήστη με τις βάσεις δεδομένων καθώς και τη δημιουργία ενός πίνακα, τον τρόπο εισαγωγής στοιχείων κ.τ.λ. Βασικές λειτουργίες μπορούν να θεωρηθούν ο περιορισμός του χρήστη ώστε να μην μπορεί να εισάγει το ίδιο ονοματεπώνυμο και τηλέφωνο πάνω από μια φορά ή η βάση να μην δέχεται στοιχεία με κενό περιεχόμενο.

Αρχικά, ξεκινάμε τη δημιουργία της κλάσης person με σκοπό να καθορίσουμε τα ιδιωτικά πεδία και τις μεθόδους της κλάσης αυτής. Δημιουργούμε δύο αρχεία ένα τύπου header (με κατάληξη .h) και ένα τύπου source (με κατάληξη .cpp). Στο αρχείο header δηλώνω τα ιδιωτικά πεδία δηλαδή τον αριθμό ταυτότητας (ακέραιος), το όνομα (αλφαριθμητικό), το επίθετο (αλφαριθμητικό) και το τηλέφωνο(αλφαριθμητικό). Στη συνέχεια, δηλώνω την συνάρτηση δημιουργίας person, η οποία δέχεται σαν είσοδο τα ιδιωτικά πεδία που δηλώσαμε νωρίτερα, καθώς και τις συναρτήσεις όπου επιστρέφουν τις τιμές των ιδιωτικών πεδίων· τέλος δηλώνω τη συνάρτηση toString, όπου επιστρέφει συνολικά τα στοιχεία του ατόμου. Συνεχίζουμε, με τη δημιουργία της βάσης δεδομένων χρησιμοποιώντας την QSQLITE δηλώνοντας τις αντίστοιχες βιβλιοθήκες καθώς και το αρχείο person.h. Οι μέθοδοι που υλοποιούμε είναι για εισαγωγή στοιχείων, διαγραφή στοιχείων, εμφάνιση συγκεκριμένου στοιχείου με βάση τον αριθμό ταυτότητας και εμφάνιση όλων των ατόμων με όλα τους τα στοιχεία που υπάρχουν στη βάση δεδομένων. Στη συνάρτηση δημιουργίας που έχω ονομάσει database, δημιουργούμε τον πίνακα greekPeople και τις στήλες του, όπου κάθε στήλη είναι τα χαρακτηριστικά που προσδιορίσαμε ότι θα έχει ένα άτομο στη

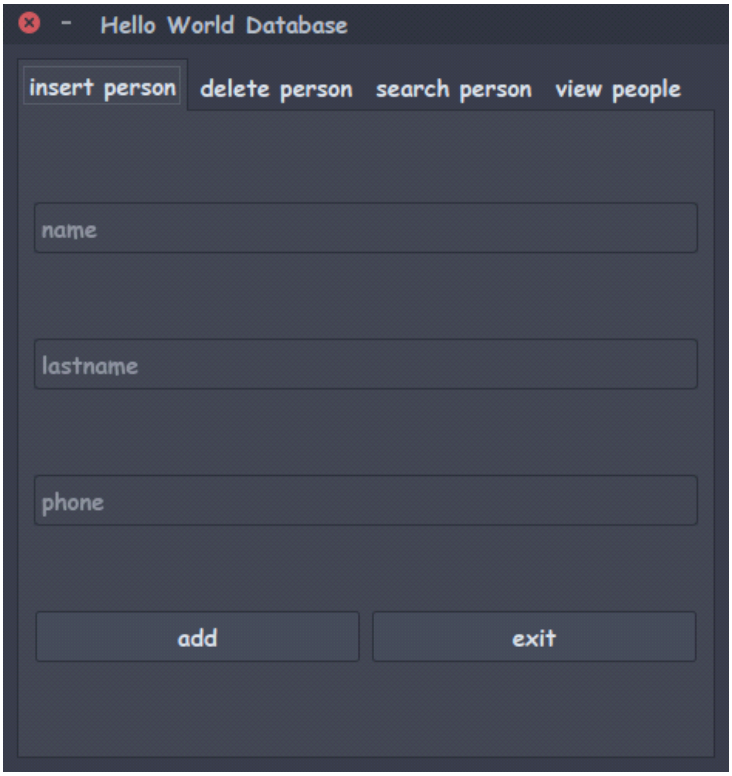
βάση μας, και ορίζουμε σαν πρωτεύον κλειδί τον αριθμό ταυτότητας ο οποίος θα αυξάνεται διαδοχικά σε κάθε εισαγωγή στοιχείου. Παρακάτω, προχωράμε σε μια μικρή ανάλυση της υλοποίησης των μεθόδων με τις οποίες αλληλεπιδρά ο χρήστης με τη βάση. Η υλοποίηση των μεθόδων αυτών (εισαγωγή, διαγραφή και εμφάνιση στοιχείων) φαίνεται στην εικόνα 3.19.

```
17  bool database::insertPerson(person &p)
18  {
19      bool success = false;
20      QSqlQuery query(db);
21      query.prepare("INSERT INTO greekPeople (name,lastname,phone)VALUES (:name,:lastname,:phone)");
22      query.bindValue(":name",p.getName());
23      query.bindValue(":lastname",p.getLastname());
24      query.bindValue(":phone",p.getPhone());
25      if(query.exec()) success = true;
26      else
27      {
28              qDebug() << "Database error:  "
29                       << query.lastError();
30      }
31      return success;
32  }
33
34  bool database::deletePerson(int id)
35  {
36      bool success = false;
37      QSqlQuery query(db);
38      query.prepare("DELETE FROM greekPeople WHERE id = :value");
39      query.bindValue(":value",id);
40      if(query.exec()) success = true;
41      else
42      {
43              qDebug() << "Database error:  "
44                       << query.lastError();
45      }
46      return success;
47  }
48
49  QVector<person*> database::getPeople()
50  {
51      QVector<person*> people;
52      QSqlQuery query(db);
53      query.exec("SELECT * FROM greekPeople");
54      while (query.next())
55      {
56          person *p = new person(query.value(0).toInt(),query.value(1).toString(),
57                                 query.value(2).toString(),
58                                 query.value(3).toString());
59          people.append(p);
60      }
61      return people;
62  }
```

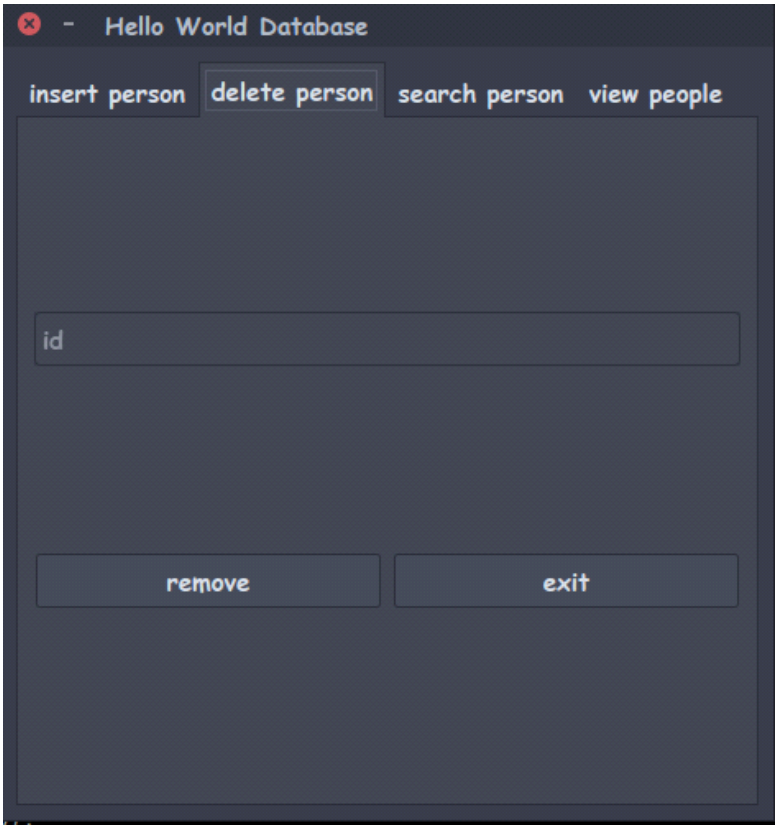
Εικόνα 3.19: Ο κώδικας των μεθόδων εισαγωγής, διαγραφής και εμφάνισης στοιχείων σε μια βάση δεδομένων.

Η μέθοδος insertPerson(person &p) δέχεται σαν είσοδο την συνάρτηση δημιουργίας person που έχουμε ορίσει πιο πάνω· στη συνάρτηση μέσα χρησιμοποιούμε την sqlite εντολή για εισαγωγή στοιχείων της οποίας η γενική μορφή είναι INSERT INTO <όνομα πίνακα> (<οι στήλες στις οποίες θέλουμε να κάνουμε εισαγωγή>) VALUES (<οι αντίστοιχες μεταβλητές>). Έπειτα, έχουμε τη συνάρτηση, για διαγραφή ολόκληρης της

γραμμής από τη βάση, την deletePerson(int id) η οποία δέχεται σαν είσοδο τον αριθμό ταυτότητας ο οποίος αντιστοιχίζεται με το στοιχείο προς διαγραφή. Γενική μορφή της εντολής για διαγραφή στοιχείων είναι DELETE FROM <όνομα πίνακα> WHERE <συνθήκη>. Τέλος, έχουμε την μεθόδο getPeople() για την εμφάνιση όλων των δεδομένων που περιέχονται στη βάση μας. Η εντολή αυτή έχει την μορφή SELECT <στοιχεία προς εμφάνιση> FROM <όνομα πίνακα>. Όταν όμως η εμφάνιση των στοιχείων που θέλουμε να κάνουμε είναι συγκεκριμένη, όπως για παράδειγμα όταν θέλω να εμφανίσω όλα τα άτομα εκείνα με αριθμό ταυτότητας "4", η εντολή αυτή παίρνει τη μορφή SELECT <στοιχεία προς εμφάνιση> FROM <όνομα πίνακα> WHERE <συνθήκη>, το WHERE μας δίνει τη δυνατότητα να εμφανίσουμε συγκεκριμένα στοιχεία του πίνακα, δηλαδή εκείνα που έχουν αριθμό ταυτότητας ίσο με 4. Αξίζει να σημειωθεί ότι στην εντολή DELETE αν δεν βάλουμε συνθήκη τότε απλώς διαγράφεται ολόκληρος ο πίνακας μαζί με όλα τα στοιχεία του. Παρακάτω, παραθέτω δυο εικόνες της εφαρμογής που υλοποιήθηκε στο κεφάλαιο αυτό. Ο πλήρης κώδικας του παραδείγματος δίνεται στο παράρτημα Α.



Εικόνα 3.20: Στιγμιότυπο εφαρμογής για εισαγωγή δεδομένων.



Εικόνα 3.21: Στιγμιότυπο εφαρμογής για διαγραφή δεδομένων.

4. Η εφαρμογή που υλοποιήθηκε

Η εφαρμογή που υλοποιήθηκε στην παρούσα εργασία είναι μια γραφική εφαρμογή γραμμένη σε qt c++, η οποία "πατάει" πάνω σε έναν αλγόριθμο κατηγοριοποίησης, με στόχο την εύκολη χρήση του από τον χρήστη. Ο κώδικας κατηγοριοποίησης βασίζεται στη μέθοδο της γραμματικής εξέλιξης και μπορείτε ελεύθερα να ανατρέξετε και να κατεβάσετε τον κώδικα στην ηλεκτρονική διεύθυνση https://github.com/itsoulos/GenClass. Θα προσπαθήσουμε να αναλύσουμε κάποια βασικά μέρη της εφαρμογής, θα δείξουμε τα βήματα του κώδικα κατηγοριοποίησης και τέλος θα εκτελέσουμε κάποια παραδείγματα όπου στα αποτελέσματα αυτών θα γίνει μια μικρή επεξήγηση. Στο παράρτημα Β, υπάρχει ολόκληρος ο κώδικας εφαρμογής.

4.1 Μέρη της εφαρμογής

4.1.1 Η γραφική εφαρμογή

Η γραφική εφαρμογή που έχει υλοποιηθεί στην παρούσα εργασία (εικόνα 4.1) αποτελείται από 6 κουμπιά (QPuhButton) και μια επιφάνεια στην οποία ο αλγόριθμος εμφανίζει τα αποτελέσματα. Τα κουμπιά είναι τα εξής:

- **Run**: εκτελεί τον αλγόριθμο αφού πρώτα προτρέψει τον χρήστη να επιλέξει την μέθοδο ή αλλιώς τον τρόπο με τον οποίο θα εμφανιστούν τα αποτελέσματα. Οι επιλογές του χρήστη είναι τρεις:
- ***Simple:*** Το πρόγραμμα τυπώνει κατευθείαν το αποτέλεσμα χωρίς περαιτέρω αναλύσεις.
- ***Comma separated value (csv):*** Το πρόγραμμα τυπώνει το αποτέλεσμα, καθώς και κάποιες επιπλέον πληροφορίες για κάθε γενιά. Οι επιπλέον πληροφορίες είναι ο αριθμός των γενεών, το train error και το test error (επεξήγηση των όρων train error και test error γίνονται στο κεφάλαιο 4.2).
- ***Full:*** Είναι η πλήρης μέθοδος όπου το πρόγραμμα σε κάθε γενιά (ή αλλιώς επανάληψη) τυπώνει λεπτομέρειες για τη διαδικασία βελτιστοποίησης, καθώς και το σφάλμα ταξινόμησης για κάθε διακριτή κλάση του προβλήματος.

- **Set Variables**: εμφανίζει μια φόρμα στην οποία ο χρήστης συμπληρώνει τις εξής μεταβλητές
- ***Αριθμός Γενεών (Number of Generations):*** ένας ακέραιος αριθμός ο οποίος καθορίζει το μέγιστο αριθμό γενεών για τον γενετικό αλγόριθμο. Κάθε γενιά είναι μια επανάληψη του αλγορίθμου.
- ***Αριθμός Χρωμοσωμάτων (Number of Chromosomes):*** ένας ακέραιος αριθμός που καθορίζει τον αριθμό χρωμοσωμάτων για τον γενετικό πληθυσμό.
- ***Ρυθμός Επιλογής (Selection Rate):*** ένας δεκαδικός αριθμός όπου καθορίζει τον ρυθμό επιλογής με τον οποίο επιλέγονται τα άτομα από τον τρέχων πληθυσμό που

χρησιμοποιείται στον γενετικό αλγόριθμο.

- ***Ρυθμός Μετάλλαξης (Mutation Rate):*** ο δεκαδικός αριθμός εκείνος ο οποίος καθορίζει τον ρυθμό μετάλλαξης που χρησιμοποιείται στον γενετικό αλγόριθμο. Επιλέγει τυχαία κάποια γονίδια από τα χρωμοσώματα των ατόμων του πληθυσμού και απλώς μεταβάλλει την τιμή τους
- ***Γεγονός Περιτυλίγματος (Wrapping Event):*** ο ακέραιος αριθμός "περιτυλίγματος" που καθορίζει τον μέγιστο αριθμό "περιτυλιγμάτων" που μπορούν να γίνουν. Η ανάγνωση των στοιχείων ενός χρωματοσώματος εκτελείται επαναληπτικά, μέχρις ότου να παραχθεί μια έγκυρη έκφραση ή να τελειώσει η ανάγνωση των στοιχείων του και να θεωρηθεί άκυρο το συγκεκριμένο χρωμόσωμα. Μόλις το χρωμόσωμα θεωρηθεί άκυρο, ξεκινά πάλι η ανάγνωση των στοιχείων του και εάν ο αριθμός περιτυλίγματος γίνει ίσος με τον αριθμό των αναγνώσεων των στοιχείων του χρωμοσώματος και δεν έχει παραχθεί μια έγκυρη έκφραση τότε το χρωμόσωμα θεωρείται άκυρο.
- ***Μέγεθος χρωμοσώματος (Genome Size):*** ένας ακέραιος αριθμός που καθορίζει το μέγεθος κάθε χρωμοσώματος στον πληθυσμό.

- **<u>Import Train File</u>:** Ο χρήστης εισάγει το αρχείο που χρειάζεται το πρόγραμμα, το οποίο περιέχει τα σημεία εκείνα που θα χρησιμοποιηθούν σαν δεδομένα εκπαίδευσης του αλγορίθμου. Το αρχείο θα πρέπει να είναι τύπου .txt ή .csv. Στο αρχείο αυτό θα πρέπει να υπάρχει ένας ακέραιος αριθμός που καθορίζει τη διάσταση του προβλήματος και ένας ακέραιος που καθορίζει τον αριθμό των σημείων στο αρχείο όπως φαίνεται παρακάτω. Όπου D ορίζεται η διάσταση του προβλήματος και M είναι ο αριθμός των σημείων που υπάρχουν στο αρχείο.

D
M

$$\begin{array}{ccccc} x_{11} & x_{12} & \dots & x_{1D} & y_1 \\ x_{21} & x_{22} & \dots & x_{2D} & y_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{M1} & x_{M2} & \dots & x_{MD} & y_M \end{array}$$

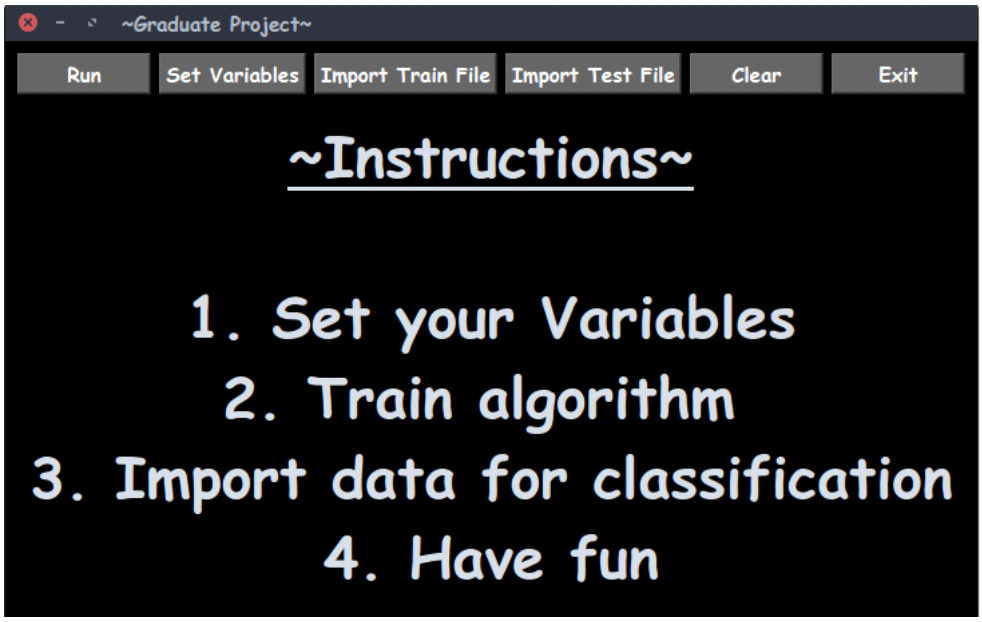
Εικόνα 4.1: Μορφή δεδομένων για εκπαίδευση και για κατηγοριοποίηση.

Κάθε γραμμή περιέχει ένα πρότυπο και η τελευταία στήλη είναι η πραγματική έξοδος του προτύπου αυτού. Το λογισμικό αναγνωρίζει τον αριθμό των κλάσεων του προβλήματος από το αρχείο εκπαίδευσης.

- **Import Test File:** Ο χρήστης εισάγει το αρχείο που περιέχει τα δεδομένα ελέγχου για το συγκεκριμένο πρόβλημα. Το αρχείο αυτό θα πρέπει να είναι της μορφής όπως φαίνεται στην εικόνα 4.1.

- **Clear:** Κάνει εκκαθάριση των αποτελεσμάτων από την οθόνη.

- **Exit:** Τερματίζει την εφαρμογή.



Εικόνα 4.2: Το κύριο μενού της γραφικής εφαρμογής.

4.1.2 Ο κώδικας κατηγοριοποίησης δεδομένων

Αρχικά, όπως αναφέρθηκε παραπάνω ο κώδικας κατηγοριοποίησης βασίζεται στην μέθοδο της γραμματικής εξέλιξης. Παρακάτω, δίνονται τα βήματα σε μια ψευδογλώσσα τα οποία εκτελεί ο αλγόριθμος για την κατηγοριοποίηση δεδομένων.

- Βήμα αρχικοποίησης.
- Διάβασε το αρχείο εκπαίδευσης.
- Όρισε τον αριθμό γενεών (Ng).
- Όρισε το πλήθος των χρωμοσωμάτων στον γενετικό πληθυσμό (Nc).
- Όρισε τον ρυθμό επιλογής (Ps).
- Όρισε τον ρυθμό μετάλλαξης (Pm).
- Αρχικοποίησε τα χρωματοσώματα στον γενετικό πληθυσμό. Τα χρωμοσώματα αρχικοποιούνται τυχαία σαν διανύσματα ακεραίων.

- Βήμα Γενετικής.
- Για i=1,...,Ng κάνε
- Δημιούργησε για κάθε χρωμόσωμα στον πληθυσμό ένα πρόγραμμα ταξινόμησης χρησιμοποιώντας την προηγούμενη διαδικασία της γραμματικής εξέλιξης. Δήλωσε το πρόγραμμα ταξινόμησης σαν Ci.
- Υπολόγισε τη συνάρτηση fi για κάθε χρωμόσωμα του γενετικού πληθυσμού. Η fi δίνεται από τη παρακάτω σχέση, όπου M είναι ο αριθμός εισροής προτύπων στα δεδομένα εκπαίδευσης, xi είναι το i-οστο πρότυπο εκπαίδευσης και ti είναι η επιθυμητή έξοδος:

$$f_i = \sum_{i=1}^{M} \left(C_i\left(x_i\right) - t_i\right)^2$$

- Εφαρμογή της διαδικασίας επιλογής: Τα χρωμοσώματα ταξινομούνται σε φθίνουσα σειρά με βάση την τιμή της συνάρτησής τους. Τα πρώτα (1-Ps) x Nc

χρωμοσώματα αντιγράφονται αυτούσια στις επόμενες γενεές. Τα υπόλοιπα χρωμοσώματα παράγονται χρησιμοποιώντας τη διαδικασία διασταύρωσης (crossover procedure). Για κάθε νέο χρωμόσωμα επιλέγονται άλλα δύο χρωματοσώματα σαν γονείς από τον παλιό πληθυσμό με τη μέθοδο της επιλογής διαγωνισμού (tournament selection). Η επιλογή διαγωνισμού έχει ως εξής: επιλέγεται τυχαία ένα σετ από χρωματοσώματα (τουλάχιστον δυο) και το χρωμόσωμα με τη καλύτερη τιμή συνάρτησης επιλέγεται. Τα επιλεγμένα ζευγάρια διασταυρώνονται σε ένα τυχαίο σημείο που ονομάζεται σημείο διασταύρωσης (crossover point). Κατά τη διαδικασία αυτή, στους "γονείς" ορίζεται ένα τυχαίο σημείο "κοπής" όπου από το σημείο αυτό και έπειτα το δεξί μέρος των χρωματοσωμάτων ανταλλάζεται το ένα με το άλλο στα "παιδιά" (βλ. Εικόνα 2.2) .

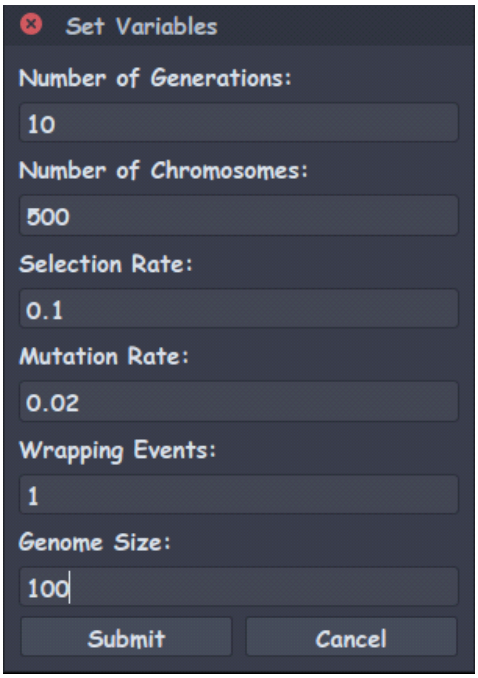
- Εφαρμογή της διαδικασίας μετάλλαξης: Για κάθε στοιχείο σε κάθε χρωμόσωμα παράγεται ένας τυχαίος αριθμός r σε διάστημα [0,1] . Εάν το r <= Pm τότε το αντίστοιχο στοιχείο αλλάζει τυχαία.
- Τέλος Για.

- Βήμα Αξιολόγησης.
- Δημιούργησε ένα πρόγραμμα ταξινόμησης για το καλύτερο χρωμόσωμα στον γενετικό πληθυσμό.
- Εφάρμοσε το προηγούμενο πρόγραμμα στο αρχείο δοκιμής και ανέφερε το σφάλμα που προκαλείται. (24)

4.2 Παραδείγματα χρήσεως της εφαρμογής

Στο κεφάλαιο αυτό θα πραγματοποιηθεί μια επίδειξη της εφαρμογής. Ξεκινάμε με το εξής παράδειγμα, έχουμε στη διάθεση μας δεδομένα τα οποία προκύπτουν σαν αποτέλεσμα μιας χημικής ανάλυσης κρασιών τα οποία προέρχονται από την ίδια περιοχή στην Ιταλία αλλά από τρεις διαφορετικές ποικιλίες το κάθε ένα. Τα έχουμε εισάγει σε ένα αρχείο .txt με τη μορφή που ορίζει η εικόνα 4.1. Έπειτα, αρχικοποιούμε τις μεταβλητές στη φόρμα που εμφανίζεται πατώντας το κουμπί set variables (εικόνα 4.4), εισάγουμε το αρχείο με τα

δεδομένα εκπαίδευσης του αλγορίθμου και το αρχείο με τα δεδομένα για την κατηγοριοποίηση αυτών.



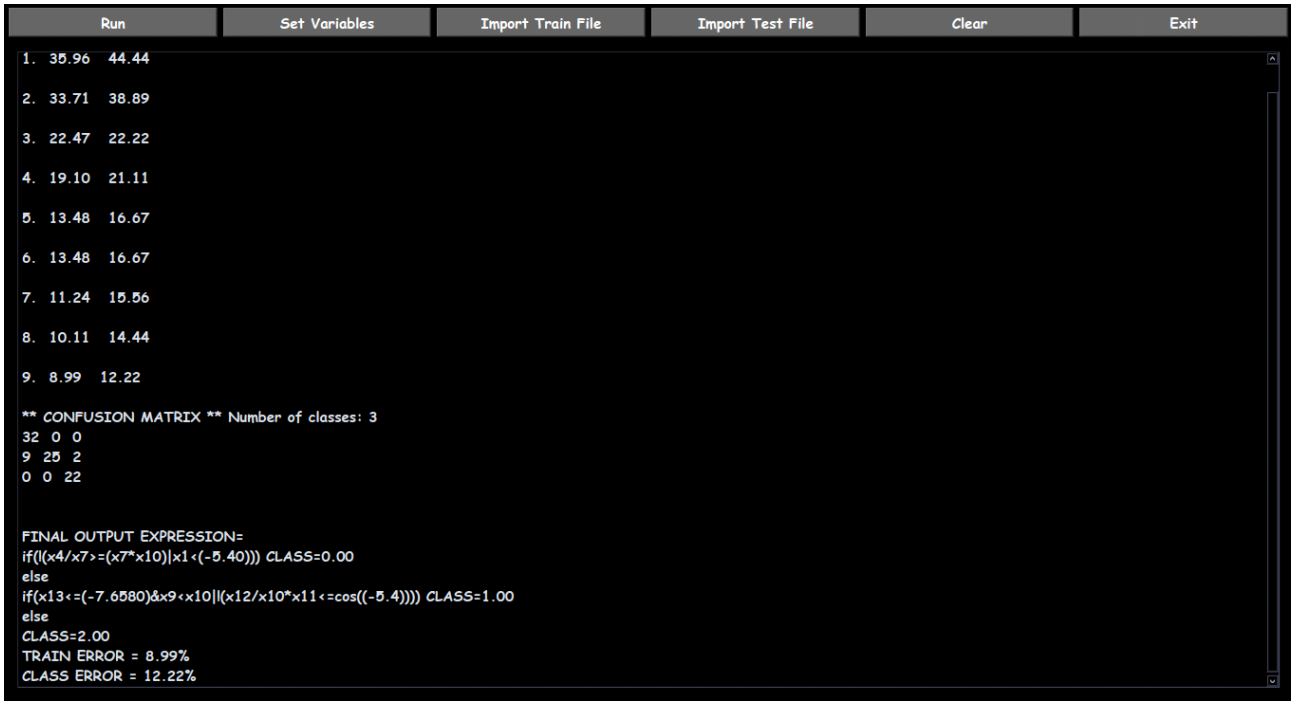
Εικόνα 4.3: Φόρμα αρχικοποίησης μεταβλητών.

Κατά την εκτέλεση της εφαρμογής επιλέγουμε την εμφάνιση των αποτελεσμάτων σε csv μορφή και τα αποτελέσματα του αλγορίθμου για κάθε επανάληψη (υπ' όψιν ότι κάθε επανάληψη εκπροσωπεί μια γενιά στον γενετικό πληθυσμό) φαίνονται στην εικόνα 4.5. Η πρώτη στήλη είναι ο αριθμός των επαναλήψεων ή αλλιώς ο αριθμός των γενεών, η δεύτερη στήλη είναι τα αποτελέσματα του train error για κάθε επανάληψη και η τρίτη είναι τα αποτελέσματα του class error για κάθε επανάληψη.

- **Train error (Σφάλμα εκπαίδευσης):** Σφάλμα εκπαίδευσης είναι το σφάλμα εκείνο που παίρνουμε όταν τρέχουμε το εκπαιδευμένο μοντέλο πίσω στα δεδομένα εκπαίδευσης. Υπ' όψιν ότι τα δεδομένα αυτά έχουν ήδη χρησιμοποιηθεί για την κατάρτιση του μοντέλου και αυτό κατ 'ανάγκην δεν σημαίνει ότι το μοντέλο που θα εκπαιδευτεί θα

εκτελεστεί με ακρίβεια όταν εφαρμοστεί πίσω στα ίδια δεδομένα εκπαίδευσης

- **Class error (Σφάλμα ταξινόμησης):** Σφάλμα ταξινόμησης είναι το σφάλμα εκείνο που παίρνουμε όταν εκτελείτε το εκπαιδευμένο μοντέλο σε ένα σύνολο δεδομένων που δεν είχε εκτεθεί ποτέ. Αυτά τα δεδομένα χρησιμοποιούνται συχνά για τη μέτρηση της ακρίβειας του μοντέλου προτού αποσταλεί στην παραγωγή.



Εικόνα 4.4: Αποτελέσματα κατηγοριοποίησης δεδομένων κρασιού.

Τέλος, βλέπουμε την αντίστοιχη έκφραση που παράγεται για κάθε μια από τις τρεις κλάσεις βασισμένο σε μια γραμματική που χρησιμοποιεί ο κώδικας κατηγοριοποίησης. Μια παρόμοια μορφή της γραμματικής που χρησιμοποιεί ο κώδικας είναι εκείνη της εικόνας 4.6, προσοχή όμως η γραμματική της εικόνας 4.6 αναφέρεται σε ταξινόμηση δυο κλάσεων, το παράδειγμα μας αφορά την ταξινόμηση δεδομένων σε τρεις κλάσεις.

```
<S>::=if(<BEXPR>) CLASS=0 else CLASS=1 (0)
<BEXPR>::=<XLIST><BOOLOP><EXPR>            (0)
         |!(<BEXPR>)                       (1)
         |<XLIST><BOOLOP><EXPR>&<BEXPR>    (2)
         |<XLIST><BOOLOP><EXPR>|<BEXPR>    (3)
<BOOLOP>::= >                              (0)
          | >=                             (1)
          | <                              (2)
          | <=                             (3)

<EXPR>::= (<EXPR><BINARYOP><EXPR>)         (0)
        | <FUNCTION>(<EXPR>)               (1)
        | <TERMINAL>                       (2)
<BINARYOP>::= +                            (0)
            | -                            (1)
            | *                            (2)
            | /                            (3)
<FUNCTION>::= sin | cos | exp | log        (0-3)
<TERMINAL>::= <XLIST>                      (0)
            | <DIGITLIST>.<DIGITLIST>      (1)
            | (-<DIGITLIST>.<DIGITLIST>)   (2)
 <XLIST>::= x1 | x2 | ...|xD               (0-D-1)
<DIGITLIST>::=<DIGIT>                      (0)
             | <DIGIT><DIGIT>              (1)
             | <DIGIT><DIGIT><DIGIT>       (2)
<DIGIT>::= 0 | 1 | 2 | 3 | 4 |5 |6 |7 |8 |9    (0-9)
```

Εικόνα 4.5: Γραμματική που χρησιμοποιεί ο κώδικας για κατηγοριοποίηση δεδομένων σε δυο κλάσεις.

ΠΑΡΑΡΤΗΜΑ Α: Ο κώδικας του παραδείγματος από το κεφάλαιο 3.4

database.h

```
#ifndef DATABASE
#define DATABASE

#include <QSqlDatabase>
#include <QSqlQuery>
#include <QSqlError>
#include <person.h>

class database{

private:
   QSqlDatabase db;

public:
   database();
   bool insertPerson(person &p);
   bool deletePerson(int id);
   QVector<person*> getPeople();
   QVector<person*> getPeople(int id);
   ~database();

};

#endif // DATABASE
```

mainWindow.h

```
#ifndef MAINWINDOW

#define MAINWINDOW

#include <QObject>
#include <QMainWindow>
#include <QHBoxLayout>
#include <QVBoxLayout>
#include <QLabel>
#include <QPushButton>
#include <QLineEdit>
#include <database.h>
#include <person.h>

class mainWindow:public QMainWindow{
   Q_OBJECT

public:
   mainWindow(QWidget *parent = 0);
   void insertTab();
   void deleteTab();
   void searchTab();
   void viewTab();

private:
   database *db;
   QWidget *mainWidget;
```

```
    QHBoxLayout *mainLayout;
    QTabWidget *tabWidget;

    QLineEdit *idEdit, *nameEdit, *lastnameEdit, *phoneEdit, *deleteIdEdit;

    QVBoxLayout *viewLayout;
    QVBoxLayout *searchLayout;

    QPushButton *viewButton;

public slots:
    void insertSlot();
    void deleteSlot();
    void viewSlot();
    void searchSlot();
};

#endif // MAINWINDOW
```

person.h

```
#ifndef PERSON

#define PERSON

#include <QString>

class person{

private:
    int id;
```

```
    QString name, lastname, phone;

public:
    person(int i,QString n, QString l, QString p);
    int getId();
    QString getName();
    QString getLastname();
    QString getPhone();
    QString toString();
};

#endif // PERSON
```

database.cpp

```
#include <database.h>

#include <QDebug>

database::database()
{
    db = QSqlDatabase::addDatabase("QSQLITE");
    db.setDatabaseName("example.db");
    db.open();
    QSqlQuery q(db);
    bool greekPeople = q.exec(QString("create table if not exists greekPeople(")+
                              QString("id integer primary key autoincrement,")+
                              QString("name varchar(50),lastname varchar(50),")+
                              QString("phone varchar(50))")
```

```
                        );
}

bool database::insertPerson(person &p)
{
   bool success = false;
   QSqlQuery query(db);
   query.prepare("INSERT INTO greekPeople (name,lastname,phone)VALUES
(:name,:lastname,:phone)");
   query.bindValue(":name",p.getName());
   query.bindValue(":lastname",p.getLastname());
   query.bindValue(":phone",p.getPhone());
   if(query.exec()) success = true;
   else
   {
         qDebug() << "Database error:  "
               << query.lastError();
   }
   return success;
}

bool database::deletePerson(int id)
{
   bool success = false;
   QSqlQuery query(db);
   query.prepare("DELETE FROM greekPeople WHERE id = :value");
   query.bindValue(":value",id);
   if(query.exec()) success = true;
   else
   {
         qDebug() << "Database error:  "
```

```
                << query.lastError();
    }
    return success;
}

QVector<person*> database::getPeople()
{
    QVector<person*> people;
    QSqlQuery query(db);
    query.exec("SELECT * FROM greekPeople");
    while (query.next())
    {
        person *p = new person(query.value(0).toInt(),query.value(1).toString(),
                               query.value(2).toString(),
                               query.value(3).toString());
        people.append(p);
    }
    return people;
}

QVector<person*> database::getPeople(int value)
{
    QVector<person*> people;
    QSqlQuery query(db);
    query.exec("SELECT * FROM greekPeople WHERE id ="+QString::number(value));
    query.bindValue(":value",value);
    while (query.next())
    {
        person *p = new person(query.value(0).toInt(),
                               query.value(1).toString(),
                               query.value(2).toString(),
```

```
                        query.value(3).toString());
        people.append(p);
    }
    return people;
}

database::~database()
{
    db.close();
}
```

mainWindow.cpp

```
#include <mainwindow.h>

#include<QDebug>

mainWindow::mainWindow (QWidget *parent):QMainWindow(parent)
{

    setFixedSize(470,470);
    setWindowTitle("Hello World Database");

    mainWidget = new QWidget;
    setCentralWidget(mainWidget);
    mainWidget->setFixedSize(this->width(),this->height());

    mainLayout = new QHBoxLayout;
    mainWidget->setLayout(mainLayout);
```

```
    tabWidget = new QTabWidget;
    mainLayout->addWidget(tabWidget);
    db = new database;
    insertTab();
    deleteTab();
    searchTab();
    viewTab();
}

void mainWindow::insertTab()
{
    QWidget *insertWidget = new QWidget;
    tabWidget->addTab(insertWidget,"insert person");

    QVBoxLayout *insertLayout = new QVBoxLayout;
    insertWidget->setLayout(insertLayout);

    nameEdit = new QLineEdit;
    nameEdit->setPlaceholderText("name");
    insertLayout->addWidget(nameEdit);

    lastnameEdit = new QLineEdit;
    lastnameEdit->setPlaceholderText("lastname");
    insertLayout->addWidget(lastnameEdit);

    phoneEdit = new QLineEdit;
    phoneEdit->setPlaceholderText("phone");
    insertLayout->addWidget(phoneEdit);

    QHBoxLayout *buttonLayout = new QHBoxLayout;
    insertLayout->addLayout(buttonLayout);

    QPushButton *insertButton = new QPushButton;
```

```
    insertButton->setText("add");
    buttonLayout->addWidget(insertButton);
    connect(insertButton,SIGNAL(clicked(bool)),this,SLOT(insertSlot()));

    QPushButton *exitButton = new QPushButton;
    exitButton->setText("exit");
    buttonLayout->addWidget(exitButton);
    connect(exitButton,SIGNAL(clicked(bool)),this,SLOT(close()));

}

void mainWindow::insertSlot()
{
    person p(NULL,nameEdit->text(),lastnameEdit->text(),
            phoneEdit->text());
    db->insertPerson(p);
}

void mainWindow::deleteTab()
{
    QWidget *deleteWidget = new QWidget;
    tabWidget->addTab(deleteWidget,"delete person");

    QVBoxLayout *deleteLayout = new QVBoxLayout;
    deleteWidget->setLayout(deleteLayout);

    deleteIdEdit = new QLineEdit;
    deleteIdEdit->setPlaceholderText("id");
    deleteLayout->addWidget(deleteIdEdit);

    QHBoxLayout *buttonLayout = new QHBoxLayout;
    deleteLayout->addLayout(buttonLayout);
```

```
    QPushButton *deleteButton = new QPushButton;
    deleteButton->setText("remove");
    buttonLayout->addWidget(deleteButton);
    connect(deleteButton,SIGNAL(clicked(bool)),this,SLOT(deleteSlot()));

    QPushButton *exitButton = new QPushButton;
    exitButton->setText("exit");
    buttonLayout->addWidget(exitButton);
    connect(exitButton,SIGNAL(clicked(bool)),this,SLOT(close()));
}

void mainWindow::deleteSlot()
{
    db->deletePerson(deleteIdEdit->text().toInt());
}

void mainWindow::searchTab()
{
    QWidget *searchWidget = new QWidget;
    tabWidget->addTab(searchWidget,"search person");

    searchLayout = new QVBoxLayout;
    searchWidget->setLayout(searchLayout);

    idEdit = new QLineEdit;
    idEdit->setPlaceholderText("id");
    searchLayout->addWidget(idEdit);

    QHBoxLayout *buttonLayout = new QHBoxLayout;
    searchLayout->addLayout(buttonLayout);

    QPushButton *searchButton = new QPushButton;
    searchButton->setText("search");
```

```
    buttonLayout->addWidget(searchButton);
    connect(searchButton,SIGNAL(clicked(bool)),this,SLOT(searchSlot()));

    QPushButton *exitButton = new QPushButton;
    exitButton->setText("exit");
   buttonLayout->addWidget(exitButton);
    connect(exitButton,SIGNAL(clicked(bool)),this,SLOT(close()));

}

void mainWindow::searchSlot()
{
    QLabel *result = new QLabel;
    searchLayout->addWidget(result);
    QVector<person*> people=db->getPeople(idEdit->text().toInt());
    QString htmlText="<h3 align=center>Your Results</h3>";

    htmlText+="<table border=1 cellpadding=10 cellspacing=3> <tr>";
    htmlText+= "<th> ID </th>";
    htmlText+= "<th> NAME </th>";
    htmlText+= "<th> LASTNAME </th>";
    htmlText+= "<th> PHONE </th> </tr>";

    for(int i=0;i<people.size();i++)
    {
        htmlText+="<tr><td>"+QString::number(people[i]->getId())+"</td>";
        htmlText+="<td>"+people[i]->getName()+"</td>";
        htmlText+="<td>"+people[i]->getLastname()+"</td>";
        htmlText+="<td>"+people[i]->getPhone()+"</td></tr>";
        delete people[i];
    }
    htmlText+="</table>";
```

```
    result->setText(htmlText);
}

void mainWindow::viewTab()
{
    QWidget *viewWidget = new QWidget;
    tabWidget->addTab(viewWidget,"view people");

    viewLayout = new QVBoxLayout;
    viewWidget->setLayout(viewLayout);

    QHBoxLayout *buttonLayout = new QHBoxLayout;
    viewLayout->addLayout(buttonLayout);

    viewButton = new QPushButton;
    viewButton->setText("view all people");
    buttonLayout->addWidget(viewButton);
    connect(viewButton,SIGNAL(clicked(bool)),this,SLOT(viewSlot()));

    QPushButton *exit = new QPushButton;
    exit->setText("exit");
    buttonLayout->addWidget(exit);
    connect(exit,SIGNAL(clicked(bool)),this,SLOT(close()));
}

void mainWindow::viewSlot()
{
    QLabel *result = new QLabel;
    viewLayout->addWidget(result);
    QVector<person*> people=db->getPeople();
    QString htmlText= "<h3 align=center>People from Greece</h3>";
    htmlText+= "<table border=1 cellpadding=10 cellspacing=3> <tr>";
```

```
    htmlText+= "<th> ID </th>";
    htmlText+= "<th> NAME </th>";
    htmlText+= "<th> LASTNAME </th>";
    htmlText+= "<th> PHONE </th> </tr>";
    for(int i=0;i<people.size();i++)
    {
        htmlText+="<tr><td>"+QString::number(people[i]->getId())+"</td>";
        htmlText+="<td>"+people[i]->getName()+"</td>";
        htmlText+="<td>"+people[i]->getLastname()+"</td>";
        htmlText+="<td>"+people[i]->getPhone()+"</td></tr>";
        delete people[i];
    }
    htmlText+="</table>";
    result->setText(htmlText);
    viewButton->setEnabled(false);

}
```

person.cpp

```
#include <person.h>

person::person(int i, QString n, QString l, QString p)
{
    id = i;
    name = n;
    lastname = l;
    phone = p;
```

```
}

int person::getId() {return id;}

QString person::getName() {return name;}

QString person::getLastname() {return lastname;}

QString person::getPhone() {return phone;}

QString person::toString(){
   return QString::number(id)+" "+name+" "+lastname+" "+phone;
}
```

main.cpp

```
#include <QApplication>

#include "mainwindow.h"

int main(int argc, char *argv[])
{
   QApplication a(argc, argv);

   mainWindow w;
   w.show();

   return a.exec();

}
```

ΠΑΡΑΡΤΗΜΑ Β: Ο κώδικας της εφαρμογής

classprogram.h

```
#ifndef CLASSPROGRAM

#define CLASSPROGRAM
# include <program.h>
# include <cprogram.h>
# include <vector>
using namespace std;

typedef vector<double> Data;
class ClassProgram      :public Program
{
   private:
      vector<double> vclass;
      vector<string> pstring;
      vector<int>    pgenome;
      vector<double*> trainx;
      vector<double>  trainy;
```

```
    vector<double*> testx;
    vector<double>  testy;
    vector<double>  outy;
    Cprogram *program;
    vector<double> mapper;
    int dimension,nclass;
  public:
    ClassProgram(char *filename);
    string  printF(vector<int> &genome);
    int     findMapper(double x);
    virtual double  fitness(vector<int> &genome);
    double  getClassError(vector<int> &genome,char *filename);
    void    getOutputs(vector<double> &real,vector<double> &est);
    int     getClass() const;
    void    getTrainData(vector<Data> &tx,Data &ty);
    void    setTrainData(vector<Data> &tx,Data &ty);
    double  getClassError(vector<int> &genome,vector<Data> &tx,Data &ty);
    ~ClassProgram();
};
#endif // CLASSPROGRAM
```

cprogram.h

```
#ifndef CPROGRAM

#define CPROGRAM
# include <symbol.h>
# include <rule.h>
```

```
# include <fparser.h>
# define SCALE  1
extern int mperror;
class Cprogram
{
   protected:
      string         vars;
      FunctionParser  parser;
      int            dimension,pdimension;
      vector<Rule*> rule;
      Symbol         Start, Expr, Near,NotNear,nearfunction,Number,
                     addN,subN,multN,divN,
                     function, binaryop, terminal,
                     XXlist,DigitList, Digit0, Digit1,
                  MinMax,Sin, Cos, Exp, Log, Abs, Sqrt,Avg,
                  Min, PI, Max,Delim,BooleanExpr,In,NotIn;
      Symbol         andor,Not,boolExpr,inexpr,infunction,
                     boolop,Gt,Lt,Ge,Le,Eq,Neq,And,Or;
      Symbol         Plus, Minus, Mult, Div, Pow;
      Symbol         Lpar, Rpar, Dot, Comma;
      Symbol         Tan, Int, Log10,Kernel,xlist;
      vector<Symbol>  Digit;
      vector<Symbol>  XX;
      int                 newRule();
      void                 makeTerminals();
      void                 makeNonTerminals();
      void                 makeRules();
   public:
      Cprogram(int dim,int pdim);
      int     Parse(string expr);
```

```
        double  Eval(const double *X);
        int     EvalError();
        Symbol  *getStartSymbol();
        ~Cprogram();
};
#endif // CPROGRAM
```

doublestack.h

```
#ifndef DOUBLESTACK

#define DOUBLESTACK
# include <stdlib.h>
# include <stdio.h>
# include <math.h>
# include <string.h>

class DoubleStack
{
   private:
      double *data;
      int counter;
   public:
      DoubleStack();
      int  size() const;
      void push(double x);
      double  top() const;
      double pop();
```

```
    void    clear();
    ~DoubleStack();
};

#endif // DOUBLESTACK
```

fparser.h

```
#ifndef FPARSER

#define FPARSER

#include <string>
#include <map>
#include <vector>

#ifdef FUNCTIONPARSER_SUPPORT_DEBUG_OUTPUT
#include <iostream>
#endif

extern int lastVariable;

class FunctionParser
{
public:
  enum ParseErrorType
  {
    SYNTAX_ERROR=0, MISM_PARENTH, MISSING_PARENTH, EMPTY_PARENTH,
    EXPECT_OPERATOR, OUT_OF_MEMORY, UNEXPECTED_ERROR,
INVALID_VARS,
```

```
    ILL_PARAMS_AMOUNT, PREMATURE_EOS, EXPECT_PARENTH_FUNC,
    FP_NO_ERROR
  };

  int Parse(const std::string& Function, const std::string& Vars,
            bool useDegrees = false);
  const char* ErrorMsg() const;
  inline ParseErrorType GetParseErrorType() const { return parseErrorType; }

  double Eval(const double* Vars);
  inline int EvalError() const { return evalErrorType; }

  bool AddConstant(const std::string& name, double value);

  typedef double (*FunctionPtr)(const double*);

  bool AddFunction(const std::string& name,
                   FunctionPtr, unsigned paramsAmount);
  bool AddFunction(const std::string& name, FunctionParser&);

  void Optimize();

  FunctionParser();
  ~FunctionParser();

  // Copy constructor and assignment operator (implemented using the
  // copy-on-write technique for efficiency):
  FunctionParser(const FunctionParser&);
  FunctionParser& operator=(const FunctionParser&);

#ifdef FUNCTIONPARSER_SUPPORT_DEBUG_OUTPUT
  // For debugging purposes only:
  void PrintByteCode(std::ostream& dest) const;
```

```
#endif

//===========================================================================
==========
private:
//===========================================================================
==========

// Private data:
// ------------
   ParseErrorType parseErrorType;
   int evalErrorType;

   struct Data
   {
      unsigned referenceCounter;

      int varAmount;
      bool useDegreeConversion;

      typedef std::map<std::string, unsigned> VarMap_t;
      VarMap_t Variables;

      typedef std::map<std::string, double> ConstMap_t;
      ConstMap_t Constants;

      VarMap_t FuncPtrNames;
      struct FuncPtrData
      {
         FunctionPtr ptr; unsigned params;
         FuncPtrData(FunctionPtr p, unsigned par): ptr(p), params(par) {}
      };
      std::vector<FuncPtrData> FuncPtrs;
```

```
        VarMap_t FuncParserNames;
        std::vector<FunctionParser*> FuncParsers;

        unsigned* ByteCode;
        unsigned ByteCodeSize;
        double* Immed;
        unsigned ImmedSize;
        double* Stack;
        unsigned StackSize;

        Data();
       ~Data();
        Data(const Data&);

        Data& operator=(const Data&); // not implemented on purpose
    };

    Data* data;
    unsigned evalRecursionLevel;

    // Temp data needed in Compile():
    unsigned StackPtr;
    std::vector<unsigned>* tempByteCode;
    std::vector<double>* tempImmed;

// Private methods:
// ---------------
    void copyOnWrite();

    bool checkRecursiveLinking(const FunctionParser*) const;

    bool isValidName(const std::string&) const;
    Data::VarMap_t::const_iterator FindVariable(const char*,
```

```
                                    const Data::VarMap_t&) const;
    Data::ConstMap_t::const_iterator FindConstant(const char*) const;
    int CheckSyntax(const char*);
    bool Compile(const char*);
    bool IsVariable(int);
    void AddCompiledByte(unsigned);
    void AddImmediate(double);
    void AddFunctionOpcode(unsigned);
    inline void incStackPtr();
    int CompileIf(const char*, int);
    int CompileFunctionParams(const char*, int, unsigned);
    int CompileElement(const char*, int);
    int CompilePow(const char*, int);
    int CompileUnaryMinus(const char*, int);
    int CompileMult(const char*, int);
    int CompileAddition(const char*, int);
    int CompileComparison(const char*, int);
    int CompileAnd(const char*, int);
    int CompileOr(const char*, int);
    int CompileExpression(const char*, int, bool=false);

    void MakeTree(void*) const;
};
#endif // FPARSER
```

fpconfig.h

```
//============================
```

```
// Function parser v2.8 by Warp
//=============================

// Configuration file
// ------------------

// NOTE:
// This file is for the internal use of the function parser only.
// You don't need to include this file in your source files, just
// include "fparser.hh".

/*
 Comment out the following line if your compiler supports the (non-standard)
 asinh, acosh and atanh functions and you want them to be supported. If
 you are not sure, just leave it (those function will then not be supported).
*/
#define NO_ASINH

/*
 Uncomment the following line to disable the eval() function if it could
 be too dangerous in the target application.
 Note that even though the maximum recursion level of eval() is limited,
 it is still possible to write functions using it which take enormous
 amounts of time to evaluate even though the maximum recursion is never
 reached. This may be undesirable in some applications.
*/
//#define DISABLE_EVAL

/*
 Maximum recursion level for eval() calls:
*/
#define EVAL_MAX_REC_LEVEL 1000
```

```
/*
 Comment out the following lines out if you are not going to use the
 optimizer and want a slightly smaller library. The Optimize() method
 can still be called, but it will not do anything.
 If you are unsure, just leave it. It won't slow down the other parts of
 the library.
*/
#ifndef NO_SUPPORT_OPTIMIZER
#define SUPPORT_OPTIMIZER
#endif

/*
 Epsilon value used with the comparison operators (must be non-negative):
 (Comment it out if you don't want to use epsilon in comparisons. Might
 lead to marginally faster evaluation of the comparison operators, but
 can introduce inaccuracies in comparisons.)
*/
#define FP_EPSILON 1e-14
```

fptypes.h

```
//=============================

// Function parser v2.8 by Warp
//=============================

// NOTE:
// This file contains only internal types for the function parser library.
// You don't need to include this file in your code. Include "fparser.hh"
```

```
// only.

namespace FUNCTIONPARSERTYPES
{
// The functions must be in alphabetical order:
   enum OPCODE
   {
      cAbs, cAcos,
#ifndef NO_ASINH
      cAcosh,
#endif
      cAsin,
#ifndef NO_ASINH
      cAsinh,
#endif
      cAtan,
      cAtan2,
#ifndef NO_ASINH
      cAtanh,
#endif
      cCeil, cCos, cCosh, cCot, cCsc,
#ifndef DISABLE_EVAL
      cEval,
#endif
      cExp, cFloor, cIf, cInt, cLog, cLog10, cMax, cMin,
      cSec, cSin, cSinh, cSqrt, cTan, cTanh,

// These do not need any ordering:
      cImmed, cJump,
      cNeg, cAdd, cSub, cMul, cDiv, cMod, cPow,
      cEqual, cNEqual, cLess, cLessOrEq, cGreater, cGreaterOrEq,
```

```
        cNot, cAnd, cOr,

        cDeg, cRad,

        cFCall, cPCall,

#ifdef SUPPORT_OPTIMIZER
        cVar, cDup, cInv,
#endif

        VarBegin
    };

    struct FuncDefinition
    {
        const char* name;
        unsigned nameLength;
        unsigned opcode;
        unsigned params;

        // This is basically strcmp(), but taking 'nameLength' as string
        // length (not ending '\0'):
        bool operator<(const FuncDefinition& rhs) const
        {
            for(unsigned i = 0; i < nameLength; ++i)
            {
                if(i == rhs.nameLength) return false;
                const char c1 = name[i], c2 = rhs.name[i];
                if(c1 < c2) return true;
                if(c2 < c1) return false;
            }
            return nameLength < rhs.nameLength;
```

```
    }
  };

// This list must be in alphabetical order:
  const FuncDefinition Functions[]=
  {
    { "abs", 3, cAbs, 1 },
    { "acos", 4, cAcos, 1 },
#ifndef NO_ASINH
    { "acosh", 5, cAcosh, 1 },
#endif
    { "asin", 4, cAsin, 1 },
#ifndef NO_ASINH
    { "asinh", 5, cAsinh, 1 },
#endif
    { "atan", 4, cAtan, 1 },
    { "atan2", 5, cAtan2, 2 },
#ifndef NO_ASINH
    { "atanh", 5, cAtanh, 1 },
#endif
    { "ceil", 4, cCeil, 1 },
    { "cos", 3, cCos, 1 },
    { "cosh", 4, cCosh, 1 },
    { "cot", 3, cCot, 1 },
    { "csc", 3, cCsc, 1 },
#ifndef DISABLE_EVAL
    { "eval", 4, cEval, 0 },
#endif
    { "exp", 3, cExp, 1 },
    { "floor", 5, cFloor, 1 },
    { "if", 2, cIf, 0 },
```

```
        { "int", 3, cInt, 1 },
        { "log", 3, cLog, 1 },
        { "log10", 5, cLog10, 1 },
        { "max", 3, cMax, 2 },
        { "min", 3, cMin, 2 },
        { "sec", 3, cSec, 1 },
        { "sin", 3, cSin, 1 },
        { "sinh", 4, cSinh, 1 },
        { "sqrt", 4, cSqrt, 1 },
        { "tan", 3, cTan, 1 },
        { "tanh", 4, cTanh, 1 }
    };
}
```

mainwindow.h

```
#ifndef MAINWINDOW_H

#define MAINWINDOW_H

#include <QHBoxLayout>
#include <QMainWindow>
#include <QWidget>
#include <QPushButton>
#include <QFileDialog>
#include <QFile>
#include <QTextStream>
#include <QLabel>
```

```
#include <QStackedLayout>
#include <QTextEdit>
#include <population.h>
#include <classprogram.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>

extern char     output_method[100];
extern int      random_seed;
extern int      wrapping;
extern int      foldcount;
extern int numberGenerations;
extern int numberChromosomes;
extern double selectionRate;
extern double mutationRate;
extern int genomeSize;
extern char *myTrainFile;
extern char *myTestFile;
extern vector<int> genome;
extern int pattern_dimension;
extern vector<double> lmargin,rmargin;

/** Output methods **/
# define CSV_METHOD     "csv"
# define SIMPLE_METHOD  "simple"
# define FULL_METHOD    "full"

class MainWindow:public QMainWindow{
   Q_OBJECT

public:
```

```
    MainWindow(QWidget *parent = 0);
     void createButtons();
     void displayInstructions();
     void displayResult();
     void clearResults();
     void printConfusionMatrix();
     void initProgram();
     void doneProgram();
     void runGenetic();
     void foldValidation();
     void runOneFold();
     int myabs(int x);

private:
    QWidget *mainWidget;
    QWidget *widgetRe;

    QVBoxLayout *resultLayout;
    QVBoxLayout *mainLayout;

    QPushButton *run, *sets, *importTrain, *importTest, *clear, *exit;
    QHBoxLayout *buttonLayout;

    QStackedLayout *panel;

    QLabel *instructions;

    Population *pop;
    ClassProgram *p;

    int choice;

    int flagVariables = 0;
```

```
    int flagTrainFile = 0;
    int flagTestFile = 0;
    int flagClear = 0;

    vector<int> genome;
    vector<double> lmargin,rmargin;
    string s;
    int ok_to_print=1;

    QTextEdit *messageArea;

public slots:
    void variablesFormSlot();
    void runButtonSlot();
    void clearButtonSlot();
    void importTrainSlot();
    void importTestSlot();
    void outputMethodSlot();

};

#endif // MAINWINDOW_H
```

outputmethoddialog.h

```
#ifndef OUTPUTMETHODDIALOG

#define OUTPUTMETHODDIALOG

#include <QWidget>
```

```
#include <QObject>
#include <QDialog>
#include <QVBoxLayout>
#include <QRadioButton>
#include <QButtonGroup>
#include <QPushButton>

class outputMethodDialog: public QDialog{
  Q_OBJECT

public:
   outputMethodDialog(QWidget *parent = 0);
   int choice;
   int getChoice();

private:
   QVBoxLayout *mainLayout;
   QWidget *mainWidget;
   QRadioButton *csvButton, *fullButton, *simpleButton;
   QPushButton *next;
   QButtonGroup *group;

public slots:
   void csvSlot(bool selected);
   void fullSlot(bool selected);
   void simpleSlot(bool selected);
};

#endif // OUTPUTMETHODDIALOG
```

population.h

```
# ifndef __POPULATION__H

# include <program.h>

/* The Population class holds the current population. */
/* The mutation, selection and crossover operators are defined here */
class Population
{
   private:
      int     **children;
      int     **trialx;
      int     **genome;
      int     *valid;
      double *fitness_array;
      double  mutation_rate,selection_rate;
      int     genome_count;
      int     genome_size;
      int     generation;
      Program *program;

      double  fitness(vector<int> &g);
      void    select();
      void    crossover();
      void    mutate();
      void    calcFitnessArray();
      int     elitism;
      int     tournament();
   public:
      Population(int gcount,int gsize,Program *p);
      void    setElitism(int s);
```

```
        int     getGeneration() const;
        int     getCount() const;
        int     getSize() const;
        void    nextGeneration();
        void    setMutationRate(double r);
        void    setSelectionRate(double r);
        double  getSelectionRate() const;
        double  getMutationRate() const;
        double  getBestFitness() const;
        double  evaluateBestFitness();
        vector<int> getBestGenome() const;
        void    reset();
        ~Population();

};
# define __POPULATION__H
# endif
```

program.h

```
# ifndef __PROGRAM__H

# include <symbol.h>
# include <rule.h>
# include <doublestack.h>
# include <vector>
using namespace std;

class Program
```

```
{
   protected:
      Symbol *start_symbol;
   public:
      Program();
      void    setStartSymbol(Symbol *s);
      Symbol  *getStartSymbol() const;
      string  printRandomProgram(vector<int> &genome,int &redo);
      int     hasFailed();
      int     parse(vector<Symbol*> slist);
      virtual double  fitness(vector<int> &genome);
      ~Program();
};

# define __PROGRAM__H
# endif
```

rule.h

```
# ifndef __RULE__H

# include <symbol.h>
# include <doublestack.h>
# include <vector>
using namespace std;

/* Creates the rules based on the grammar file and adds the symbols */

class Rule
```

```
{
   private:
      vector<Symbol*> data;
      int     length;
   public:
      Rule();
      void    addSymbol(Symbol *s);
      int     getSymbolPos(string s);
      Symbol  *getSymbol(int pos) const;
      void    setSymbol(int pos,Symbol *s);
      int     getLength() const;
      string  printRule(vector<int> genome,int &pos,int &redo);
      double  getValue(vector<int> genome,int &pos,int &redo,DoubleStack
&stack,double *X);
      ~Rule();
};
# define __RULE__H
# endif
```

setdialogvariables.h

```
#ifndef SETDIALOGVARIABLES

#define SETDIALOGVARIABLES

#include <QDialog>
#include <QVBoxLayout>
#include <QHBoxLayout>
```

```
#include <QWidget>
#include <QObject>
#include <QLineEdit>
#include <QPushButton>

class setDialogVariables: public QDialog{
  Q_OBJECT

public:
   setDialogVariables(QWidget *parent = 0);
   int getNumberGenerations();
   int getNumberChromosomes();
   double getSelectionRate();
   double getMutationRate();
   int getWrappingEvent();
   int getGenomeSize();

   int numberGenerations, numberChromosomes,wrappingEvent, genomeSize;
   double selectionRate, mutationRate;

private:
   QPushButton *submit,*cancel;
   QWidget *mainWidget;
   QVBoxLayout *mainLayout;
   QHBoxLayout *buttonLayout;
   QLineEdit *generationsLineEdit, *chromosomesLineEdit, *selectionLineEdit,
            *mutationLineEdit,*wrappingEdit, *genomeSizeEdit;

};
#endif // SETDIALOGVARIABLES
```

symbol.h

```
# ifndef __SYMBOL__H

# include <string>
# include <vector>
using namespace std;

/* Creates the symbols based on the grammar file and adds them to the rules */

class Rule;

class Symbol
{
   private:
      string name;
      vector<Rule*> rule;
      int    count_rules;
      int    is_terminal;
   public:
      Symbol();

      void   set(string s,int status);
      void   setName(string s);
      string  getName() const;

      void   setTerminalStatus(int status);
      int    getTerminalStatus() const;

      void   addRule(Rule *r);
      void   cut(int N);
```

```
        Rule    *getRule(int pos) const;
        int     getCountRules() const;
        ~Symbol();

};

# define __SYMBOL__H
# endif
```

classprogram.cpp

```
# include <classprogram.h>

# include <math.h>
# define NAN_CLASS      1e+10

static double dmax(double a,double b) {return a>b?a:b;}

int     problem_dimension;

ClassProgram::ClassProgram(char *filename)
{
   FILE *fp=fopen(filename,"r");
   if(!fp) return;
   int valid_count;
   int d,c;
   fscanf(fp,"%d",&d);
   dimension = d;
   fscanf(fp,"%d",&c);
```

```
trainy.resize(c);
trainx.resize(c);
vclass.resize(0);
problem_dimension = d;
for(int i=0;i<c;i++) trainx[i]=new double[d];
for(int i=0;i<c;i++)
{
   for(int j=0;j<d;j++)
   {
      fscanf(fp,"%lf",&trainx[i][j]);
   }
   fscanf(fp,"%lf",&trainy[i]);
   int flag=0;
   for(int j=0;j<vclass.size();j++)
   {
      if(fabs(vclass[j]-trainy[i])<1e-5)
      {
         flag=1;
         break;
      }
   }
   if(!flag)
   {
      int s=vclass.size();
      vclass.resize(s+1);
      vclass[s]=trainy[i];
   }
}
fclose(fp);
//sort vclass
```

```
    for(int i=0;i<vclass.size();i++)
    {
       for(int j=0;j<vclass.size()-1;j++)
       {
          if(vclass[j+1]<vclass[j])
          //if(vclass[j+1]>vclass[j])
          {
             double d=vclass[j];
             vclass[j]=vclass[j+1];
             vclass[j+1]=d;
          }
       }
    }

    mapper.resize(vclass.size());
    for(int i=0;i<vclass.size();i++)
    {
       mapper[i]=vclass[i];
       vclass[i]=i;
    }

    program = new Cprogram(d,vclass.size()-1);
    setStartSymbol(program->getStartSymbol());
    pstring.resize(vclass.size());
    for(int i=0;i<pstring.size();i++) pstring[i]=" ";
    pgenome.resize(0);
    nclass = vclass.size();
    outy.resize(trainy.size());
}

string  ClassProgram::printF(vector<int> &genome)
```

```
{
   string ret="";
   if(pgenome.size()!=genome.size()/(nclass-1))
      pgenome.resize(genome.size()/(nclass-1));
   char str[100];
   extern int wrapping;
   for(int i=0;i<nclass-1;i++)
   {
      for(int j=0;j<genome.size()/(nclass-1);j++)
         pgenome[j]=genome[i*genome.size()/(nclass-1)+j];
      int redo=0;
      pstring[i]=printRandomProgram(pgenome,redo);
      if(redo>=wrapping) return "";
      ret+="if(";
      ret+=pstring[i];
      ret+=") CLASS=";
      sprintf(str,"%.2lf",vclass[i]);
      ret+=str;
      ret+="\nelse \n";
   }
   sprintf(str,"%.2lf",vclass[nclass-1]);
   ret+="CLASS=";
   ret+=str;
   ret+="\n";
   return ret;
}

int    ClassProgram::findMapper(double y)
{
   for(int i=0;i<vclass.size();i++)
      if(fabs(mapper[i]-y)<1e-7) return i;
```

```
    return 0;
}

double  ClassProgram::getClassError(vector<int> &genome,vector<Data> &tx,Data &ty)
{
    if(testx.size()!=tx.size())
     for(int i=0;i<testx.size();i++)
        delete[] testx[i];
    testy.resize(tx.size());
    testx.resize(tx.size());
    extern int wrapping;
    int d=tx[0].size();
    int c=tx.size();
    for(int i=0;i<c;i++) testx[i]=new double[d];
    for(int i=0;i<c;i++)
    {
        for(int j=0;j<d;j++)
            testx[i][j]=tx[i][j];
        testy[i]=ty[i];
    }
    double value=0.0;
    if(pgenome.size()!=genome.size()/(nclass-1))
        pgenome.resize(genome.size()/(nclass-1));
    if(outy.size()!=testy.size()) outy.resize(testy.size());
    for(int i=0;i<outy.size();i++) outy[i]=NAN_CLASS;
    for(int i=0;i<nclass-1;i++)
    {
        for(int j=0;j<pgenome.size();j++)
            pgenome[j]=genome[i*genome.size()/(nclass-1)+j];
        int redo=0;
        string s = printRandomProgram(pgenome,redo);
```

```
        if(redo>=wrapping) return -1e+8;
        pstring[i]=s;
    }
    for(int j=0;j<nclass-1;j++)
    {
        program->Parse(pstring[j]);
        double dclass;
        for(int i=0;i<testy.size();i++)
        {
            if(fabs(outy[i]-NAN_CLASS)>1e-5) continue;
            double v=program->Eval(testx[i]);
            if(isnan(v) || isinf(v)) return -1e+8;
            if(fabs(v-1.0)<1e-5) outy[i]=vclass[j];
        }
    }

    for(int i=0;i<testy.size();i++)
    {
        if(fabs(outy[i]-NAN_CLASS)<1e-5) outy[i]=vclass[nclass-1];
        value=value+(fabs(findMapper(testy[i])-outy[i])>1e-5);
    }

    if(isnan(value) || isinf(value)) return -1e+8;
    return -value*100.0/testy.size();
}

double  ClassProgram::getClassError(vector<int> &genome,char *filename)
{
    FILE *fp=fopen(filename,"r");
    if(!fp) return -1.0;
    if(testx.size())
```

```
 for(int i=0;i<testx.size();i++)
    delete[] testx[i];
int d,c;
fscanf(fp,"%d",&d);
fscanf(fp,"%d",&c);
testy.resize(c);
testx.resize(c);
extern int wrapping;
for(int i=0;i<c;i++) testx[i]=new double[d];
for(int i=0;i<c;i++)
{
    for(int j=0;j<d;j++)
        fscanf(fp,"%lf",&testx[i][j]);
    fscanf(fp,"%lf",&testy[i]);
}
fclose(fp);
double value=0.0;
if(pgenome.size()!=genome.size()/(nclass-1))
    pgenome.resize(genome.size()/(nclass-1));
if(outy.size()!=testy.size()) outy.resize(testy.size());
for(int i=0;i<outy.size();i++) outy[i]=NAN_CLASS;
for(int i=0;i<nclass-1;i++)
{
    for(int j=0;j<pgenome.size();j++)
        pgenome[j]=genome[i*genome.size()/(nclass-1)+j];
    int redo=0;
    string s = printRandomProgram(pgenome,redo);
    if(redo>=wrapping) return -1e+8;
    pstring[i]=s;
}
```

```
   for(int j=0;j<nclass-1;j++)
   {
      program->Parse(pstring[j]);
      double dclass;
      for(int i=0;i<testy.size();i++)
      {
         if(fabs(outy[i]-NAN_CLASS)>1e-5) continue;
         double v=program->Eval(testx[i]);
         if(isnan(v) || isinf(v)) return -1e+8;
         if(fabs(v-1.0)<1e-5) outy[i]=vclass[j];
      }
   }

   for(int i=0;i<testy.size();i++)
   {
      if(fabs(outy[i]-NAN_CLASS)<1e-5) outy[i]=vclass[nclass-1];
      value=value+(fabs(findMapper(testy[i])-outy[i])>1e-5);
   }

   if(isnan(value) || isinf(value)) return -1e+8;
   return -value*100.0/testy.size();
}

int     ClassProgram::getClass() const
{
   return nclass;
}

double  ClassProgram::fitness(vector<int> &genome)
{
   double value=0.0;
```

```
//printf("nclass = %d \n",nclass);

 if(pgenome.size()!=genome.size()/(nclass-1))
    pgenome.resize(genome.size()/(nclass-1));
 if(outy.size()!=trainy.size()) outy.resize(trainy.size());
 for(int i=0;i<outy.size();i++) outy[i]=NAN_CLASS;
 extern int wrapping;
 for(int i=0;i<nclass-1;i++)
 {
    for(int j=0;j<pgenome.size();j++)
       pgenome[j]=genome[i*genome.size()/(nclass-1)+j];
    int redo=0;
    string s = printRandomProgram(pgenome,redo);
    if(redo>=wrapping) return -1e+8;
    pstring[i]=s;
 }

 for(int j=0;j<nclass-1;j++)
 {
    program->Parse(pstring[j]);
    double dclass;
    for(int i=0;i<trainy.size();i++)
    {
       if(fabs(outy[i]-NAN_CLASS)>1e-5) continue;
       double v=program->Eval(trainx[i]);
       if(isnan(v) || isinf(v) ) return -1e+8;
       if(fabs(v-1.0)<1e-5) outy[i]=vclass[j];
    }
 }

 vector<int> fail;
```

```
    vector<int> belong;
    fail.resize(nclass);
    belong.resize(nclass);
    for(int i=0;i<nclass;i++)
       fail[i]=belong[i]=0;
    for(int i=0;i<trainy.size();i++)
    {
       if(fabs(outy[i]-NAN_CLASS)<1e-5)        outy[i]=vclass[nclass-1];
       int pos=findMapper(trainy[i]);
       value=value+((fabs(findMapper(trainy[i])-outy[i]))>1e-5);
       belong[pos]++;
       if(fabs(findMapper(trainy[i])-outy[i])>1e-5)
       {
          fail[pos]++;
       }
    }
   /* extern*/ int ok_to_print=0; //with 1 print class, with 0 not
    if(ok_to_print)
    for(int i=0;i<nclass;i++)
    {
       printf("CLASS[%3d (%3d)] FAIL=%5.2lf%%
\n",i,belong[i],fail[i]*100.0/belong[i]);
    }

    if(isnan(value) || isinf(value)) return -1e+8;
    return -value*100.0/trainy.size();
}

void    ClassProgram::getTrainData(vector<Data> &tx,Data &ty)
{
    tx.resize(trainx.size());
```

```
    ty.resize(trainy.size());
    for(int i=0;i<trainx.size();i++)
    {
        tx[i].resize(dimension);
        for(int j=0;j<dimension;j++) tx[i][j]=trainx[i][j];
        ty[i]=trainy[i];
    }
}

void    ClassProgram::getOutputs(vector<double> &real,vector<double> &est)
{
    real.resize(testy.size());
    est.resize(testy.size());
    for(int i=0;i<testy.size();i++)
    {
        real[i]=findMapper(testy[i]);
        est[i]=outy[i];
    }
}

void    ClassProgram::setTrainData(vector<Data> &tx,Data &ty)
{
    if(trainx.size() != tx.size())
    {
        for(int i=0;i<trainy.size();i++) delete[] trainx[i];
        trainx.resize(tx.size());
        trainy.resize(ty.size());
        for(int i=0;i<tx.size();i++)
        {
            trainx[i]=new double[tx[0].size()];
            for(int j=0;j<tx[0].size();j++) trainx[i][j]=tx[i][j];
```

```
            trainy[i]=ty[i];
        }
    }
}

ClassProgram::~ClassProgram()
{
    for(int i=0;i<trainy.size();i++) delete[] trainx[i];
    for(int i=0;i<testy.size();i++)  delete[] testx[i];
    delete program;
}
```

cprogram.cpp

```
# include <cprogram.h>

# include <stdio.h>
# include <math.h>

extern vector<double> lmargin;
extern vector<double> rmargin;

static double dmax(double a,double b)
{
    return a>b?a:b;
}

static double dmin(double a,double b)
{
```

```
   return a<b?a:b;
}

double between(const double *X)
{
   return X[0]>=dmin(X[1],X[2]) && X[0]<=dmax(X[1],X[2]);
}

double not_between(const double *X)
{
   return !(X[0]>=dmin(X[1],X[2]) && X[0]<=dmax(X[1],X[2]));
}

double near(const double *X)
{
   double d=dmax(X[1],X[2])-dmin(X[1],X[2]);
   return fabs(X[0]-dmin(X[1],X[2]))<d/10
     ||
        fabs(X[0]-dmax(X[1],X[2])<d/10);
}
double notnear(const double *X)
{
   double d=dmax(X[1],X[2])-dmin(X[1],X[2]);
   return !(fabs(X[0]-dmin(X[1],X[2]))<d/10
     ||
        fabs(X[0]-dmax(X[1],X[2])<d/10));
}

double near2(const double *X)
{
   return fabs(X[0]-dmin(X[1],X[2]))<X[3]
     ||
```

```
        fabs(X[0]-dmax(X[1],X[2])<X[3]);
}

double notnear2(const double *X)
{
   return !(fabs(X[0]-dmin(X[1],X[2]))<X[3]
     ||
        fabs(X[0]-dmax(X[1],X[2])<X[3]));
}

double  kernel(const double *X)
{
   return exp((X[0]-X[1])/(2.0 * X[2]));
}

double  sig(const double *x)
{
   return 1.0/(1.0+exp(-x[0]));
}

Cprogram::Cprogram(int dim,int pdim)
{
   parser.AddFunction("in",between,3);
   parser.AddFunction("near",near,3);
   parser.AddFunction("notnear",notnear,3);

   parser.AddFunction("near2",near2,4);
   parser.AddFunction("notnear2",notnear2,4);
   parser.AddFunction("sig",sig,1);

   dimension = dim;
   pdimension = pdim;
```

```
    makeTerminals();
    makeNonTerminals();
    makeRules();
}

int     Cprogram::newRule()
{
    int r;
    r=rule.size();
    rule.resize(r+1);
    rule[r]=new Rule();
    return r;
}

void    Cprogram::makeTerminals()
{
    Number.set("Number",1);
    In.set("in",1);
    Delim.set("#",1);
    PI.set("pi",1);
    Plus.set("+",1);
    Minus.set("-",1);
    Mult.set("*",1);
    Div.set("/",1);
    Pow.set("^",1);
    Comma.set(",",1);
    Dot.set(".",1);
    Lpar.set("(",1);
    Rpar.set(")",1);

    Sin.set("sin",1);
```

```
Cos.set("cos",1);
Exp.set("exp",1);
Log.set("log",1);
Not.set("!",1);
addN.set("Add",1);
subN.set("Sub",1);
multN.set("Mult",1);
divN.set("Div",1);

Abs.set("abs",1);
Sqrt.set("sqrt",1);
Tan.set("tan",1);
Int.set("int",1);
Log10.set("log10",1);
Min.set("min",1);
Max.set("max",1);
Near.set("near",1);
NotNear.set("notnear",1);
Kernel.set("kernel",1);
Gt.set(">",1);
Ge.set(">=",1);
Lt.set("<",1);
Le.set("<=",1);
Eq.set("=",1);
Neq.set("!=",1);
And.set("&",1);
Or.set("|",1);
boolExpr.set("jump",1);
XX.resize(dimension);
Avg.set("sig",1);
vars="";
```

```
    for(int i=0;i<dimension;i++)
    {
       char str[100];
       sprintf(str,"x%d",i+1);
       XX[i].set(str,1);
       vars=vars+str;
       if(i<dimension-1) vars=vars+",";
    }
    Digit.resize(10);
    for(int i=0;i<10;i++)
    {
       char str[100];
       sprintf(str,"%d",i);
       Digit[i].set(str,1);
    }
}

void    Cprogram::makeNonTerminals()
{
    Start.set("START",0);
    xlist.set("XLIST",0);
    DigitList.set("DIGITLIST",0);
    Digit0.set("DIGIT0",0);
    Digit1.set("DIGIT1",0);
    XXlist.set("XXLIST",0);
    Expr.set("EXPR",0);
    function.set("FUNCTION",0);
    binaryop.set("BINARYOP",0);
    terminal.set("TERMINAL",0);
    MinMax.set("MINMAX",0);
    BooleanExpr.set("BOOLEXPR",0);
```

```
    boolop.set("BOOLOP",0);
    inexpr.set("INEXPR",0);
    infunction.set("INFUNCTION",0);
    andor.set("ANDOR",0);
    nearfunction.set("NEARFUNCTION",0);
}

void    Cprogram::makeRules()
{
    int r;
        r=newRule();
        rule[r]->addSymbol(&BooleanExpr);
        Start.addRule(rule[r]);

    r=newRule();
    rule[r]->addSymbol(&XXlist);
    rule[r]->addSymbol(&boolop);
    rule[r]->addSymbol(&Expr);
    BooleanExpr.addRule(rule[r]);

    r=newRule();
    rule[r]->addSymbol(&Not);
    rule[r]->addSymbol(&Lpar);
    rule[r]->addSymbol(&BooleanExpr);
    rule[r]->addSymbol(&Rpar);
    BooleanExpr.addRule(rule[r]);

    r=newRule();
    rule[r]->addSymbol(&XXlist);
    rule[r]->addSymbol(&boolop);
    rule[r]->addSymbol(&Expr);
    rule[r]->addSymbol(&And);
```

```
rule[r]->addSymbol(&BooleanExpr);
BooleanExpr.addRule(rule[r]);

r=newRule();
rule[r]->addSymbol(&XXlist);
rule[r]->addSymbol(&boolop);
rule[r]->addSymbol(&Expr);
rule[r]->addSymbol(&Or);
rule[r]->addSymbol(&BooleanExpr);
BooleanExpr.addRule(rule[r]);

r=newRule();
rule[r]->addSymbol(&Gt);
boolop.addRule(rule[r]);

r=newRule();
rule[r]->addSymbol(&Ge);
boolop.addRule(rule[r]);

r=newRule();
rule[r]->addSymbol(&Lt);
boolop.addRule(rule[r]);

r=newRule();
rule[r]->addSymbol(&Le);
boolop.addRule(rule[r]);

r=newRule();
rule[r]->addSymbol(&Lpar);
rule[r]->addSymbol(&Expr);
rule[r]->addSymbol(&binaryop);
rule[r]->addSymbol(&Expr);
```

```
rule[r]->addSymbol(&Rpar);
Expr.addRule(rule[r]);

r=newRule();
rule[r]->addSymbol(&function);
   rule[r]->addSymbol(&Lpar);
rule[r]->addSymbol(&Expr);
rule[r]->addSymbol(&Rpar);
Expr.addRule(rule[r]);

r=newRule();
rule[r]->addSymbol(&Log);
function.addRule(rule[r]);

r=newRule();
rule[r]->addSymbol(&Exp);
function.addRule(rule[r]);

r=newRule();
rule[r]->addSymbol(&Sin);
function.addRule(rule[r]);

r=newRule();
rule[r]->addSymbol(&Cos);
function.addRule(rule[r]);

r=newRule();
rule[r]->addSymbol(&Plus);
binaryop.addRule(rule[r]);

r=newRule();
rule[r]->addSymbol(&Minus);
binaryop.addRule(rule[r]);
```

```
r=newRule();
rule[r]->addSymbol(&Mult);
binaryop.addRule(rule[r]);

r=newRule();
rule[r]->addSymbol(&Div);
binaryop.addRule(rule[r]);

r=newRule();
rule[r]->addSymbol(&Digit0);
DigitList.addRule(rule[r]);

r=newRule();
rule[r]->addSymbol(&Digit0);
rule[r]->addSymbol(&DigitList);
DigitList.addRule(rule[r]);

r=newRule();
rule[r]->addSymbol(&Digit0);
rule[r]->addSymbol(&Digit0);
rule[r]->addSymbol(&Digit0);
//DigitList.addRule(rule[r]);

r=newRule();
rule[r]->addSymbol(&DigitList);
rule[r]->addSymbol(&Dot);
rule[r]->addSymbol(&DigitList);
terminal.addRule(rule[r]);

r=newRule();
rule[r]->addSymbol(&Lpar);
rule[r]->addSymbol(&Minus);
```

```
rule[r]->addSymbol(&DigitList);
rule[r]->addSymbol(&Dot);
rule[r]->addSymbol(&DigitList);
rule[r]->addSymbol(&Rpar);
terminal.addRule(rule[r]);
Expr.addRule(rule[r]);

r=newRule();
rule[r]->addSymbol(&XXlist);
Expr.addRule(rule[r]);

for(int i=0;i<dimension;i++)
{
   r=newRule();
   rule[r]->addSymbol(&XX[i]);
   XXlist.addRule(rule[r]);
}

r=newRule();
rule[r]->addSymbol(&Expr);
rule[r]->addSymbol(&Mult);
rule[r]->addSymbol(&XXlist);
XXlist.addRule(rule[r]);

r=newRule();
rule[r]->addSymbol(&XXlist);
rule[r]->addSymbol(&binaryop);
rule[r]->addSymbol(&XXlist);
XXlist.addRule(rule[r]);

for(int i=0;i<10;i++)
{
```

```
        r=newRule();
        rule[r]->addSymbol(&Digit[i]);
        Digit0.addRule(rule[r]);
    }

}

int             Cprogram::Parse(string expr)
{
    return (parser.Parse(expr,vars)==-1);
}

int             Cprogram::EvalError()
{
    return      parser.EvalError();
}

double  Cprogram::Eval( const double *X)
{
        return parser.Eval(X);
}

Symbol  *Cprogram::getStartSymbol()
{
    return &Start;
}

Cprogram::~Cprogram()
{
    for(int i=0;i<rule.size();i++)
        delete rule[i];
}
```

doublestack.cpp

```
# include <doublestack.h>

# include <iostream>
# include <stdio.h>
# include <stdlib.h>
# include <math.h>
using namespace std;

DoubleStack::DoubleStack()
{
   data = (double*)malloc(512*sizeof(double));//NULL;
   counter =0;
}

void    DoubleStack::clear()
{
   counter = 0;
}

int     DoubleStack::size() const
{
   return counter;
}

void DoubleStack::push(double x)
{
   if(isnan(x) || isinf(x)) {x=1e+8; }
```

```
    if(counter>=512)
    {
       data=(double*)realloc(data,(counter+1)*sizeof(double));
    }
    data[counter]=x;
    counter++;
}

double  DoubleStack::top() const
{
    return (counter!=0)?data[counter-1]:-1;
}

double  DoubleStack::pop()
{
    if(!counter) return -1;
    double t=data[counter-1];
    counter--;
    return t;
}

DoubleStack::~DoubleStack()
{
    free(data);
}
```

fparser.cpp

```
//==============================
```

```
// Function parser v2.8 by Warp
//=============================

#include "fparser.h"
#include "fpconfig.h"
#include "fptypes.h"
using namespace FUNCTIONPARSERTYPES;

#include <cstdlib>
#include <cstring>
#include <cctype>
#include <cmath>

using namespace std;

int lastVariable;
#ifndef M_PI
#define M_PI 3.14159265358979323846264338327 95
#endif

namespace
{
   const unsigned FUNC_AMOUNT = sizeof(Functions)/sizeof(Functions[0]);

   // BCB4 does not implement the standard lower_bound function.
   // This is used instead:
   const FuncDefinition* fp_lower_bound(const FuncDefinition* first,
                                        const FuncDefinition* last,
                                        const FuncDefinition& value)
   {
      while(first < last)
      {
```

```
            const FuncDefinition* middle = first+(last-first)/2;
            if(*middle < value) first = middle+1;
            else last = middle;
        }
        return last;
    }

    // Returns a pointer to the FuncDefinition instance which 'name' is
    // the same as the one given by 'F'. If no such function name exists,
    // returns 0.
    inline const FuncDefinition* FindFunction(const char* F)
    {
        FuncDefinition func = { F, 0, 0, 0 };
        while(isalnum(F[func.nameLength])) ++func.nameLength;
        if(func.nameLength)
        {
            const FuncDefinition* found =
                fp_lower_bound(Functions, Functions+FUNC_AMOUNT, func);
            if(found == Functions+FUNC_AMOUNT || func < *found)
                return 0;
            return found;
        }
        return 0;
    }
}

//---------------------------------------------------------------------------
// Copy-on-write method
//---------------------------------------------------------------------------
void FunctionParser::copyOnWrite()
{
```

```
    if(data->referenceCounter > 1)
    {
        Data* oldData = data;
        data = new Data(*oldData);
        --(oldData->referenceCounter);
        data->referenceCounter = 1;
    }
}

//---------------------------------------------------------------------------
// Constructors and destructors
//---------------------------------------------------------------------------
//===========================================================================
=============
FunctionParser::FunctionParser():
    parseErrorType(FP_NO_ERROR), evalErrorType(0),
    data(new Data),
    evalRecursionLevel(0)
{
    data->referenceCounter = 1;
}

FunctionParser::~FunctionParser()
{
    if(--(data->referenceCounter) == 0)
    {
        delete data;
    }
}

FunctionParser::FunctionParser(const FunctionParser& cpy):
```

```
    parseErrorType(cpy.parseErrorType),
    evalErrorType(cpy.evalErrorType),
    data(cpy.data),
    evalRecursionLevel(0)
{
    ++(data->referenceCounter);
}

FunctionParser& FunctionParser::operator=(const FunctionParser& cpy)
{
    if(data != cpy.data)
    {
        if(--(data->referenceCounter) == 0) delete data;

        parseErrorType = cpy.parseErrorType;
        evalErrorType = cpy.evalErrorType;
        data = cpy.data;
        evalRecursionLevel = cpy.evalRecursionLevel;

        ++(data->referenceCounter);
    }

    return *this;
}

FunctionParser::Data::Data():
    useDegreeConversion(false),
    ByteCode(0), ByteCodeSize(0),
    Immed(0), ImmedSize(0),
    Stack(0), StackSize(0)
{}
```

```
FunctionParser::Data::~Data()
{
  if(ByteCode) { delete[] ByteCode; ByteCode=0; }
  if(Immed) { delete[] Immed; Immed=0; }
  if(Stack) { delete[] Stack; Stack=0; }
}

// Makes a deep-copy of Data:
FunctionParser::Data::Data(const Data& cpy):
  varAmount(cpy.varAmount), useDegreeConversion(cpy.useDegreeConversion),
  Variables(cpy.Variables), Constants(cpy.Constants),
  FuncPtrNames(cpy.FuncPtrNames), FuncPtrs(cpy.FuncPtrs),
  FuncParserNames(cpy.FuncParserNames), FuncParsers(cpy.FuncParsers),
  ByteCode(0), ByteCodeSize(cpy.ByteCodeSize),
  Immed(0), ImmedSize(cpy.ImmedSize),
  Stack(0), StackSize(cpy.StackSize)
{
  if(ByteCodeSize) ByteCode = new unsigned[ByteCodeSize];
  if(ImmedSize) Immed = new double[ImmedSize];
  if(StackSize) Stack = new double[StackSize];

  for(unsigned i=0; i<ByteCodeSize; ++i) ByteCode[i] = cpy.ByteCode[i];
  for(unsigned i=0; i<ImmedSize; ++i) Immed[i] = cpy.Immed[i];

  // No need to copy the stack contents because it's obsolete outside Eval()
}

//---------------------------------------------------------------------------
// Function parsing
//---------------------------------------------------------------------------
//===========================================================================
=============
```

```
namespace
{
  // Error messages returned by ErrorMsg():
  const char* ParseErrorMessage[]=
  {
    "Syntax error",                         // 0
    "Mismatched parenthesis",               // 1
    "Missing ')'",                          // 2
    "Empty parentheses",                    // 3
    "Syntax error: Operator expected",      // 4
    "Not enough memory",                    // 5
    "An unexpected error occurred. Please make a full bug report "
    "to the author",                        // 6
    "Syntax error in parameter 'Vars' given to "
    "FunctionParser::Parse()",              // 7
    "Illegal number of parameters to function", // 8
    "Syntax error: Premature end of string",    // 9
    "Syntax error: Expecting ( after function", // 10
    ""
  };

  // Parse variables
  bool ParseVars(const string& Vars, map<string, unsigned>& dest)
  {
    unsigned varNumber = VarBegin;
    unsigned ind1 = 0, ind2;

    while(ind1 < Vars.size())
    {
      if(!isalpha(Vars[ind1]) && Vars[ind1]!='_') return false;
      for(ind2=ind1+1; ind2<Vars.size() && Vars[ind2]!=','; ++ind2)
```

```
            if(!isalnum(Vars[ind2]) && Vars[ind2]!='_') return false;
          const string varName = Vars.substr(ind1, ind2-ind1);

          if(dest.insert(make_pair(varName, varNumber++)).second == false)
            return false;

          ind1 = ind2+1;
        }
        return true;
    }
}

bool FunctionParser::isValidName(const std::string& name) const
{
    if(name.empty() || (!isalpha(name[0]) && name[0] != '_')) return false;
    for(unsigned i=0; i<name.size(); ++i)
        if(!isalnum(name[i]) && name[i] != '_') return false;

    if(FindFunction(name.c_str())) return false;

    return true;
}

// Constants:
bool FunctionParser::AddConstant(const string& name, double value)
{
    if(isValidName(name))
    {
        const char* n = name.c_str();
        if(FindVariable(n, data->FuncParserNames) !=
           data->FuncParserNames.end() ||
           FindVariable(n, data->FuncPtrNames) !=
```

```
        data->FuncPtrNames.end())
         return false;

      copyOnWrite();

      data->Constants[name] = value;
      return true;
   }
   return false;
}

// Function pointers
bool FunctionParser::AddFunction(const std::string& name,
                                 FunctionPtr func, unsigned paramsAmount)
{
   if(isValidName(name))
   {
      const char* n = name.c_str();
      if(FindVariable(n, data->FuncParserNames) !=
        data->FuncParserNames.end() ||
        FindConstant(n) != data->Constants.end())
         return false;

      copyOnWrite();

      data->FuncPtrNames[name] = data->FuncPtrs.size();
      data->FuncPtrs.push_back(Data::FuncPtrData(func, paramsAmount));
      return true;
   }
   return false;
}
```

```
bool FunctionParser::checkRecursiveLinking(const FunctionParser* fp) const
{
   if(fp == this) return true;
   for(unsigned i=0; i<fp->data->FuncParsers.size(); ++i)
      if(checkRecursiveLinking(fp->data->FuncParsers[i])) return true;
   return false;
}

bool FunctionParser::AddFunction(const std::string& name,
                                 FunctionParser& parser)
{
   if(isValidName(name))
   {
      const char* n = name.c_str();
      if(FindVariable(n, data->FuncPtrNames) != data->FuncPtrNames.end() ||
         FindConstant(n) != data->Constants.end())
          return false;

      if(checkRecursiveLinking(&parser)) return false;

      copyOnWrite();

      data->FuncParserNames[name] = data->FuncParsers.size();
      data->FuncParsers.push_back(&parser);
      return true;
   }
   return false;
}

// Main parsing function
// ---------------------
int FunctionParser::Parse(const std::string& Function,
```

```
                    const std::string& Vars,
                    bool useDegrees)
{
  copyOnWrite();

  data->Variables.clear();

  if(!ParseVars(Vars, data->Variables))
  {
    parseErrorType = FP_INVALID_VARS;
    return Function.size();
  }
  data->varAmount = data->Variables.size(); // this is for Eval()

  const char* Func = Function.c_str();

  parseErrorType = FP_NO_ERROR;

  int Result = CheckSyntax(Func);
  if(Result>=0) return Result;

  data->useDegreeConversion = useDegrees;
  if(!Compile(Func)) return Function.size();

  data->Variables.clear();

  parseErrorType = FP_NO_ERROR;
  return -1;
}

namespace
{
  const char* const fpOperators[] =
```

```
    {
        "+", "-", "*", "/", "%", "^",
        "=", "!=", "<=", "<", ">=", ">", "&", "|",
        0
    };

    // Is given char an operator?
    // (Returns 0 if not, else the size of the operator)
    inline int IsOperator(const char* F)
    {
        for(unsigned opInd = 0; fpOperators[opInd]; ++opInd)
        {
            const char* op = fpOperators[opInd];
            for(unsigned n = 0; F[n] == *op; ++n)
            {
                ++op;
                if(*op == 0) return op-fpOperators[opInd];
            }
        }
        return 0;
    }

    // skip whitespace
    inline void sws(const char* F, int& Ind)
    {
        while(F[Ind] && isspace(F[Ind])) ++Ind;
    }
}

// Returns an iterator to the variable with the same name as 'F', or to
// Variables.end() if no such variable exists:
```

```
inline FunctionParser::Data::VarMap_t::const_iterator
FunctionParser::FindVariable(const char* F, const Data::VarMap_t& vars) const
{
   if(vars.size())
   {
      unsigned ind = 0;
      while(isalnum(F[ind]) || F[ind] == '_') ++ind;
      if(ind)
      {
         string name(F, ind);
      /**GIANNIS **/
      if(name[0]=='x' && isdigit(name[1])) lastVariable = atoi(name.substr(1).c_str());
      /**END OF GIANNIS **/
         return vars.find(name);
      }
   }
   return vars.end();
}

inline FunctionParser::Data::ConstMap_t::const_iterator
FunctionParser::FindConstant(const char* F) const
{
   if(data->Constants.size())
   {
      unsigned ind = 0;
      while(isalnum(F[ind]) || F[ind] == '_') ++ind;
      if(ind)
      {
         string name(F, ind);
         return data->Constants.find(name);
      }
```

```
    }
    return data->Constants.end();
}

//---------------------------------------------------------------------------
// Check function string syntax
// ---------------------------
int FunctionParser::CheckSyntax(const char* Function)
{
    const Data::VarMap_t& Variables = data->Variables;
    const Data::ConstMap_t& Constants = data->Constants;
    const Data::VarMap_t& FuncPtrNames = data->FuncPtrNames;
    const Data::VarMap_t& FuncParserNames = data->FuncParserNames;

    vector<int> functionParenthDepth;

    int Ind=0, ParenthCnt=0, c;
    char* Ptr;

    while(true)
    {
        sws(Function, Ind);
        c=Function[Ind];

// Check for valid operand (must appear)

        // Check for leading - or !
        if(c=='-' || c=='!') { sws(Function, ++Ind); c=Function[Ind]; }
        if(c==0) { parseErrorType=PREMATURE_EOS; return Ind; }

        // Check for math function
        bool foundFunc = false;
        const FuncDefinition* fptr = FindFunction(&Function[Ind]);
```

```
if(fptr)
{
   Ind += fptr->nameLength;
   foundFunc = true;
}
else
{
   // Check for user-defined function
   Data::VarMap_t::const_iterator fIter =
      FindVariable(&Function[Ind], FuncPtrNames);
   if(fIter != FuncPtrNames.end())
   {
      Ind += fIter->first.size();
      foundFunc = true;
   }
   else
   {
      Data::VarMap_t::const_iterator pIter =
         FindVariable(&Function[Ind], FuncParserNames);
      if(pIter != FuncParserNames.end())
      {
         Ind += pIter->first.size();
         foundFunc = true;
      }
   }
}

if(foundFunc)
{
   sws(Function, Ind);
   c = Function[Ind];
```

```
    if(c!='(') { parseErrorType=EXPECT_PARENTH_FUNC; return Ind; }

    int Ind2 = Ind+1;
    sws(Function, Ind2);
    if(Function[Ind2] == ')')
    {
        Ind = Ind2+1;
        sws(Function, Ind);
        c = Function[Ind];
        // Ugly, but other methods would just be uglier...
        goto CheckOperator;
    }

    functionParenthDepth.push_back(ParenthCnt+1);
}

// Check for opening parenthesis
if(c=='(')
{
    ++ParenthCnt;
    sws(Function, ++Ind);
    if(Function[Ind]==')') { parseErrorType=EMPTY_PARENTH; return Ind;}
    continue;
}

// Check for number
if(isdigit(c) || (c=='.' && isdigit(Function[Ind+1])))
{
    strtod(&Function[Ind], &Ptr);
    Ind += int(Ptr-&Function[Ind]);
    sws(Function, Ind);
    c = Function[Ind];
```

```
}
else
{ // Check for variable
   Data::VarMap_t::const_iterator vIter =
      FindVariable(&Function[Ind], Variables);
   if(vIter != Variables.end())
      Ind += vIter->first.size();
   else
   {
      // Check for constant
      Data::ConstMap_t::const_iterator cIter =
         FindConstant(&Function[Ind]);
      if(cIter != Constants.end())
         Ind += cIter->first.size();
      else
      { parseErrorType=SYNTAX_ERROR; return Ind; }
   }
   sws(Function, Ind);
   c = Function[Ind];
}

// Check for closing parenthesis
while(c==')')
{
   if(functionParenthDepth.size() &&
     functionParenthDepth.back() == ParenthCnt)
      functionParenthDepth.pop_back();
   if((--ParenthCnt)<0) { parseErrorType=MISM_PARENTH; return Ind; }
   sws(Function, ++Ind);
   c=Function[Ind];
}
```

```
// If we get here, we have a legal operand and now a legal operator or
// end of string must follow

   CheckOperator:
      // Check for EOS
      if(c==0) break; // The only way to end the checking loop without error

      // Check for operator
      int opSize = 0;
      if(c == ',' && !functionParenthDepth.empty() &&
        functionParenthDepth.back() == ParenthCnt)
         opSize = 1;
      else
         opSize = IsOperator(Function+Ind);
      if(opSize == 0)
      { parseErrorType=EXPECT_OPERATOR; return Ind; }

// If we get here, we have an operand and an operator; the next loop will
// check for another operand (must appear)
      Ind += opSize;
   } // while

   // Check that all opened parentheses are also closed
   if(ParenthCnt>0) { parseErrorType=MISSING_PARENTH; return Ind; }

// The string is ok
   parseErrorType=FP_NO_ERROR;
   return -1;
}

// Compile function string to bytecode
// ----------------------------------
```

```
bool FunctionParser::Compile(const char* Function)
{
  if(data->ByteCode) { delete[] data->ByteCode; data->ByteCode=0; }
  if(data->Immed) { delete[] data->Immed; data->Immed=0; }
  if(data->Stack) { delete[] data->Stack; data->Stack=0; }

  vector<unsigned> byteCode; byteCode.reserve(1024);
  tempByteCode = &byteCode;

  vector<double> immed; immed.reserve(1024);
  tempImmed = &immed;

  data->StackSize = StackPtr = 0;

  CompileExpression(Function, 0);
  if(parseErrorType != FP_NO_ERROR) return false;

  data->ByteCodeSize = byteCode.size();
  data->ImmedSize = immed.size();

  if(data->ByteCodeSize)
  {
    data->ByteCode = new unsigned[data->ByteCodeSize];
    memcpy(data->ByteCode, &byteCode[0],
        sizeof(unsigned)*data->ByteCodeSize);
  }
  if(data->ImmedSize)
  {
    data->Immed = new double[data->ImmedSize];
    memcpy(data->Immed, &immed[0],
        sizeof(double)*data->ImmedSize);
  }
```

```
  if(data->StackSize)
    data->Stack = new double[data->StackSize];

  return true;
}

inline void FunctionParser::AddCompiledByte(unsigned c)
{
  tempByteCode->push_back(c);
}

inline void FunctionParser::AddImmediate(double i)
{
  tempImmed->push_back(i);
}

inline void FunctionParser::AddFunctionOpcode(unsigned opcode)
{
  if(data->useDegreeConversion)
    switch(opcode)
    {
      case cCos:
      case cCosh:
      case cCot:
      case cCsc:
      case cSec:
      case cSin:
      case cSinh:
      case cTan:
      case cTanh:
        AddCompiledByte(cRad);
    }
```

```
   AddCompiledByte(opcode);

   if(data->useDegreeConversion)
      switch(opcode)
      {
        case cAcos:
#ifndef NO_ASINH
        case cAcosh:
        case cAsinh:
        case cAtanh:
#endif
        case cAsin:
        case cAtan:
        case cAtan2:
          AddCompiledByte(cDeg);
      }
}

inline void FunctionParser::incStackPtr()
{
   if(++StackPtr > data->StackSize) ++(data->StackSize);
}

// Compile if()
int FunctionParser::CompileIf(const char* F, int ind)
{
   int ind2 = CompileExpression(F, ind, true); // condition
   sws(F, ind2);
   if(F[ind2] != ',') { parseErrorType=ILL_PARAMS_AMOUNT; return ind2; }
   AddCompiledByte(cIf);
   unsigned curByteCodeSize = tempByteCode->size();
```

```
    AddCompiledByte(0); // Jump index; to be set later
    AddCompiledByte(0); // Immed jump index; to be set later

    --StackPtr;

    ind2 = CompileExpression(F, ind2+1, true); // then
    sws(F, ind2);
    if(F[ind2] != ',') { parseErrorType=ILL_PARAMS_AMOUNT; return ind2; }
    AddCompiledByte(cJump);
    unsigned curByteCodeSize2 = tempByteCode->size();
    unsigned curImmedSize2 = tempImmed->size();
    AddCompiledByte(0); // Jump index; to be set later
    AddCompiledByte(0); // Immed jump index; to be set later

    --StackPtr;

    ind2 = CompileExpression(F, ind2+1, true); // else
    sws(F, ind2);
    if(F[ind2] != ')') { parseErrorType=ILL_PARAMS_AMOUNT; return ind2; }

    // Set jump indices
    (*tempByteCode)[curByteCodeSize] = curByteCodeSize2+1;
    (*tempByteCode)[curByteCodeSize+1] = curImmedSize2;
    (*tempByteCode)[curByteCodeSize2] = tempByteCode->size()-1;
    (*tempByteCode)[curByteCodeSize2+1] = tempImmed->size();

    return ind2+1;
}

int FunctionParser::CompileFunctionParams(const char* F, int ind,
                                          unsigned requiredParams)
{
    int ind2 = ind;
```

```
    if(requiredParams > 0)
    {
        unsigned curStackPtr = StackPtr;
        ind2 = CompileExpression(F, ind);

        if(StackPtr != curStackPtr+requiredParams)
        { parseErrorType=ILL_PARAMS_AMOUNT; return ind; }

        StackPtr -= requiredParams - 1;
    }
    else
    {
        incStackPtr();
    }

    sws(F, ind2);
    return ind2+1; // F[ind2] is ')'
}

// Compiles element
int FunctionParser::CompileElement(const char* F, int ind)
{
    sws(F, ind);
    char c = F[ind];

    if(c == '(')
    {
        ind = CompileExpression(F, ind+1);
        sws(F, ind);
        return ind+1; // F[ind] is ')'
    }
```

```
    if(isdigit(c) || c=='.' /*|| c=='-'*/) // Number
    {
      const char* startPtr = &F[ind];
      char* endPtr;
      double val = strtod(startPtr, &endPtr);
      AddImmediate(val);
      AddCompiledByte(cImmed);
      incStackPtr();
      return ind+(endPtr-startPtr);
    }

    if(isalpha(c) || c == '_') // Function, variable or constant
    {
      const FuncDefinition* func = FindFunction(F+ind);
      if(func) // is function
      {
        int ind2 = ind + func->nameLength;
        sws(F, ind2); // F[ind2] is '('
        if(strcmp(func->name, "if") == 0) // "if" is a special case
        {
          return CompileIf(F, ind2+1);
        }

#ifndef DISABLE_EVAL
        unsigned requiredParams =
          strcmp(func->name, "eval") == 0 ?
          data->Variables.size() : func->params;
#else
        unsigned requiredParams = func->params;
#endif
        ind2 = CompileFunctionParams(F, ind2+1, requiredParams);
```

```
    AddFunctionOpcode(func->opcode);
    return ind2; // F[ind2-1] is ')'
}

Data::VarMap_t::const_iterator vIter =
    FindVariable(F+ind, data->Variables);
if(vIter != data->Variables.end()) // is variable
{
    AddCompiledByte(vIter->second);
    incStackPtr();
    return ind + vIter->first.size();
}

Data::ConstMap_t::const_iterator cIter = FindConstant(F+ind);
if(cIter != data->Constants.end()) // is constant
{
    AddImmediate(cIter->second);
    AddCompiledByte(cImmed);
    incStackPtr();
    return ind + cIter->first.size();
}

Data::VarMap_t::const_iterator fIter =
    FindVariable(F+ind, data->FuncPtrNames);
if(fIter != data->FuncPtrNames.end()) // is user-defined func pointer
{
    unsigned index = fIter->second;

    int ind2 = ind + fIter->first.length();
    sws(F, ind2); // F[ind2] is '('

    ind2 = CompileFunctionParams(F, ind2+1,
```

```
                                    data->FuncPtrs[index].params);

            AddCompiledByte(cFCall);
            AddCompiledByte(index);
            return ind2;
        }

        Data::VarMap_t::const_iterator pIter =
            FindVariable(F+ind, data->FuncParserNames);
        if(pIter != data->FuncParserNames.end()) // is user-defined func parser
        {
            unsigned index = pIter->second;

            int ind2 = ind + pIter->first.length();
            sws(F, ind2); // F[ind2] is '('

            ind2 = CompileFunctionParams
                (F, ind2+1, data->FuncParsers[index]->data->varAmount);

            AddCompiledByte(cPCall);
            AddCompiledByte(index);
            return ind2;
        }
    }

    parseErrorType = UNEXPECTED_ERROR;
    return ind;
}

// Compiles '^'
int FunctionParser::CompilePow(const char* F, int ind)
{
    int ind2 = CompileElement(F, ind);
```

```
    sws(F, ind2);

    while(F[ind2] == '^')
    {
        ind2 = CompileUnaryMinus(F, ind2+1);
        sws(F, ind2);
        AddCompiledByte(cPow);
        --StackPtr;
    }

    return ind2;
}

// Compiles unary '-'
int FunctionParser::CompileUnaryMinus(const char* F, int ind)
{
    sws(F, ind);
    if(F[ind] == '-' || F[ind] == '!')
    {
        int ind2 = ind+1;
        sws(F, ind2);
        ind2 = CompilePow(F, ind2);
        sws(F, ind2);

        // if we are negating a constant, negate the constant itself:
        if(F[ind] == '-' && tempByteCode->back() == cImmed)
            tempImmed->back() = -tempImmed->back();

        // if we are negating a negation, we can remove both:
        else if((F[ind] == '-' && tempByteCode->back() == cNeg))
            tempByteCode->pop_back();
```

```
        else
            AddCompiledByte(F[ind] == '-' ? cNeg : cNot);

        return ind2;
    }

    int ind2 = CompilePow(F, ind);
    sws(F, ind2);
    return ind2;
}

// Compiles '*', '/' and '%'
int FunctionParser::CompileMult(const char* F, int ind)
{
    int ind2 = CompileUnaryMinus(F, ind);
    sws(F, ind2);
    char op;

    while((op = F[ind2]) == '*' || op == '/' || op == '%')
    {
        ind2 = CompileUnaryMinus(F, ind2+1);
        sws(F, ind2);
        switch(op)
        {
          case '*': AddCompiledByte(cMul); break;
          case '/': AddCompiledByte(cDiv); break;
          case '%': AddCompiledByte(cMod); break;
        }
        --StackPtr;
    }

    return ind2;
```

```
}

// Compiles '+' and '-'
int FunctionParser::CompileAddition(const char* F, int ind)
{
   int ind2 = CompileMult(F, ind);
   sws(F, ind2);
   char op;

   while((op = F[ind2]) == '+' || op == '-')
   {
      ind2 = CompileMult(F, ind2+1);
      sws(F, ind2);
      AddCompiledByte(op=='+' ? cAdd : cSub);
      --StackPtr;
   }

   return ind2;
}

// Compiles '=', '<' and '>'
int FunctionParser::CompileComparison(const char* F, int ind)
{
   int ind2 = CompileAddition(F, ind);
   sws(F, ind2);
   char op;

   while((op = F[ind2]) == '=' || op == '<' || op == '>' || op == '!')
   {
      int opSize = (F[ind2+1] == '=' ? 2 : 1);
      ind2 = CompileAddition(F, ind2+opSize);
      sws(F, ind2);
```

```
        switch(op)
        {
          case '=':
            AddCompiledByte(cEqual); break;
          case '<':
            AddCompiledByte(opSize == 1 ? cLess : cLessOrEq); break;
          case '>':
            AddCompiledByte(opSize == 1 ? cGreater : cGreaterOrEq); break;
          case '!':
            AddCompiledByte(cNEqual); break;
        }
        --StackPtr;
    }

    return ind2;
}

// Compiles '&'
int FunctionParser::CompileAnd(const char* F, int ind)
{
    int ind2 = CompileComparison(F, ind);
    sws(F, ind2);

    while(F[ind2] == '&')
    {
        ind2 = CompileComparison(F, ind2+1);
        sws(F, ind2);
        AddCompiledByte(cAnd);
        --StackPtr;
    }

    return ind2;
```

```
}

// Compiles '|'
int FunctionParser::CompileOr(const char* F, int ind)
{
   int ind2 = CompileAnd(F, ind);
   sws(F, ind2);

   while(F[ind2] == '|')
   {
      ind2 = CompileAnd(F, ind2+1);
      sws(F, ind2);
      AddCompiledByte(cOr);
      --StackPtr;
   }

   return ind2;
}

// Compiles ','
int FunctionParser::CompileExpression(const char* F, int ind, bool stopAtComma)
{
   int ind2 = CompileOr(F, ind);
   sws(F, ind2);

   if(stopAtComma) return ind2;

   while(F[ind2] == ',')
   {
      ind2 = CompileOr(F, ind2+1);
      sws(F, ind2);
   }
```

```
  return ind2;
}

// Return parse error message
// -------------------------
const char* FunctionParser::ErrorMsg() const
{
  if(parseErrorType != FP_NO_ERROR) return ParseErrorMessage[parseErrorType];
  return 0;
}

//---------------------------------------------------------------------------
// Function evaluation
//---------------------------------------------------------------------------
//===========================================================================
=============
namespace
{
  inline int doubleToInt(double d)
  {
    return d<0 ? -int((-d)+.5) : int(d+.5);
  }

  inline double Min(double d1, double d2)
  {
    return d1<d2 ? d1 : d2;
  }
  inline double Max(double d1, double d2)
  {
    return d1>d2 ? d1 : d2;
  }
```

```
    inline double DegreesToRadians(double degrees)
    {
      return degrees*(M_PI/180.0);
    }
    inline double RadiansToDegrees(double radians)
    {
      return radians*(180.0/M_PI);
    }
}

double FunctionParser::Eval(const double* Vars)
{
    const unsigned* const ByteCode = data->ByteCode;
    const double* const Immed = data->Immed;
    double* const Stack = data->Stack;
    const unsigned ByteCodeSize = data->ByteCodeSize;
    unsigned IP, DP=0;
    int SP=-1;

    for(IP=0; IP<ByteCodeSize; ++IP)
    {
      switch(ByteCode[IP])
      {
// Functions:
        case  cAbs: Stack[SP] = fabs(Stack[SP]); break;
        case  cAcos: if(Stack[SP] < -1 || Stack[SP] > 1)
                  { evalErrorType=4; return 0; }
                  Stack[SP] = acos(Stack[SP]); break;
#ifndef NO_ASINH
        case cAcosh: Stack[SP] = acosh(Stack[SP]); break;
#endif
```

```
        case  cAsin: if(Stack[SP] < -1 || Stack[SP] > 1)
                    { evalErrorType=4; return 0; }
                    Stack[SP] = asin(Stack[SP]); break;
#ifndef NO_ASINH
        case cAsinh: Stack[SP] = asinh(Stack[SP]); break;
#endif
        case  cAtan: Stack[SP] = atan(Stack[SP]); break;
        case cAtan2: Stack[SP-1] = atan2(Stack[SP-1], Stack[SP]);
                    --SP; break;
#ifndef NO_ASINH
        case cAtanh: Stack[SP] = atanh(Stack[SP]); break;
#endif
        case  cCeil: Stack[SP] = ceil(Stack[SP]); break;
        case   cCos: Stack[SP] = cos(Stack[SP]); break;
        case  cCosh: Stack[SP] = cosh(Stack[SP]); break;

        case   cCot:
           {
              double t = tan(Stack[SP]);
              if(t == 0) { evalErrorType=1; return 0; }
              Stack[SP] = 1/t; break;
           }
        case   cCsc:
           {
              double s = sin(Stack[SP]);
              if(s == 0) { evalErrorType=1; return 0; }
              Stack[SP] = 1/s; break;
           }

#ifndef DISABLE_EVAL
        case  cEval:
```

```
            {
               double retVal = 0;
               if(evalRecursionLevel == EVAL_MAX_REC_LEVEL)
               {
                  evalErrorType = 5;
               }
               else
               {
                  data->Stack = new double[data->StackSize];
                  ++evalRecursionLevel;
                  retVal = Eval(&Stack[SP-data->varAmount+1]);
                  --evalRecursionLevel;
                  delete[] data->Stack;
                  data->Stack = Stack;
               }
               SP -= data->varAmount-1;
               Stack[SP] = retVal;
               break;
            }
#endif

         case   cExp:
           {
              Stack[SP] = exp(Stack[SP]); break;
           }
         case cFloor: Stack[SP] = floor(Stack[SP]); break;

         case    cIf:
            {
               unsigned jumpAddr = ByteCode[++IP];
               unsigned immedAddr = ByteCode[++IP];
```

```
      if(doubleToInt(Stack[SP]) == 0)
      {
         IP = jumpAddr;
         DP = immedAddr;
      }
      --SP; break;
   }

case   cInt: Stack[SP] = floor(Stack[SP]+.5); break;
case   cLog:
  if(Stack[SP] <= 0) { evalErrorType=3; return 0; }
           Stack[SP] = log(Stack[SP]); break;
case cLog10: if(Stack[SP] <= 0) { evalErrorType=3; return 0; }
           Stack[SP] = log10(Stack[SP]); break;
case   cMax: Stack[SP-1] = Max(Stack[SP-1], Stack[SP]);
           --SP; break;
case   cMin: Stack[SP-1] = Min(Stack[SP-1], Stack[SP]);
           --SP; break;
case   cSec:
   {
      double c = cos(Stack[SP]);
      if(c == 0) { evalErrorType=1; return 0; }
      Stack[SP] = 1/c; break;
   }
case   cSin: Stack[SP] = sin(Stack[SP]); break;
case  cSinh: Stack[SP] = sinh(Stack[SP]); break;
case  cSqrt: if(Stack[SP] < 0) { evalErrorType=2; return 0; }
           Stack[SP] = sqrt(Stack[SP]); break;
case   cTan: Stack[SP] = tan(Stack[SP]); break;
case  cTanh: Stack[SP] = tanh(Stack[SP]); break;
```

```
// Misc:
        case cImmed: Stack[++SP] = Immed[DP++]; break;
        case  cJump: DP = ByteCode[IP+2];
                IP = ByteCode[IP+1];
                break;

// Operators:
        case   cNeg: Stack[SP] = -Stack[SP]; break;
        case   cAdd: Stack[SP-1] += Stack[SP]; --SP; break;
        case   cSub: Stack[SP-1] -= Stack[SP]; --SP; break;
        case   cMul: Stack[SP-1] *= Stack[SP]; --SP; break;
        case   cDiv: if(Stack[SP] == 0) { evalErrorType=1; return 0; }
                Stack[SP-1] /= Stack[SP]; --SP; break;
        case   cMod: if(Stack[SP] == 0) { evalErrorType=1; return 0; }
                Stack[SP-1] = fmod(Stack[SP-1], Stack[SP]);
                --SP; break;
        case   cPow: Stack[SP-1] = pow(Stack[SP-1], Stack[SP]);
                --SP; break;

#ifdef FP_EPSILON
        case cEqual: Stack[SP-1] =
                   (fabs(Stack[SP-1]-Stack[SP]) <= FP_EPSILON);
                --SP; break;
        case cNEqual: Stack[SP-1] =
                    (fabs(Stack[SP-1] - Stack[SP]) >= FP_EPSILON);
                --SP; break;
        case  cLess: Stack[SP-1] = (Stack[SP-1] < Stack[SP]-FP_EPSILON);
                --SP; break;
        case  cLessOrEq: Stack[SP-1] = (Stack[SP-1] <= Stack[SP]+FP_EPSILON);
                --SP; break;
        case cGreater: Stack[SP-1] = (Stack[SP-1]-FP_EPSILON > Stack[SP]);
```

```
                --SP; break;
        case cGreaterOrEq: Stack[SP-1] =
                        (Stack[SP-1]+FP_EPSILON >= Stack[SP]);
                --SP; break;
#else
        case cEqual: Stack[SP-1] = (Stack[SP-1] == Stack[SP]);
                --SP; break;
        case cNEqual: Stack[SP-1] = (Stack[SP-1] != Stack[SP]);
                --SP; break;
        case  cLess: Stack[SP-1] = (Stack[SP-1] < Stack[SP]);
                --SP; break;
        case  cLessOrEq: Stack[SP-1] = (Stack[SP-1] <= Stack[SP]);
                --SP; break;
        case cGreater: Stack[SP-1] = (Stack[SP-1] > Stack[SP]);
                --SP; break;
        case cGreaterOrEq: Stack[SP-1] = (Stack[SP-1] >= Stack[SP]);
                --SP; break;
#endif

        case   cAnd: Stack[SP-1] =
                    (doubleToInt(Stack[SP-1]) &&
                     doubleToInt(Stack[SP]));
                --SP; break;
        case    cOr: Stack[SP-1] =
                    (doubleToInt(Stack[SP-1]) ||
                     doubleToInt(Stack[SP]));
                --SP; break;
        case   cNot: Stack[SP] = !doubleToInt(Stack[SP]); break;

// Degrees-radians conversion:
        case   cDeg: Stack[SP] = RadiansToDegrees(Stack[SP]); break;
```

```
        case   cRad: Stack[SP] = DegreesToRadians(Stack[SP]); break;

// User-defined function calls:
        case cFCall:
          {
            unsigned index = ByteCode[++IP];
            unsigned params = data->FuncPtrs[index].params;
            double retVal =
              data->FuncPtrs[index].ptr(&Stack[SP-params+1]);
            SP -= int(params)-1;
            Stack[SP] = retVal;
            break;
          }

        case cPCall:
          {
            unsigned index = ByteCode[++IP];
            unsigned params = data->FuncParsers[index]->data->varAmount;
            double retVal =
              data->FuncParsers[index]->Eval(&Stack[SP-params+1]);
            SP -= int(params)-1;
            Stack[SP] = retVal;
            break;
          }

#ifdef SUPPORT_OPTIMIZER
        case   cVar: break; // Paranoia. These should never exist
        case   cDup: Stack[SP+1] = Stack[SP]; ++SP; break;
        case   cInv:
          if(Stack[SP] == 0.0) { evalErrorType=1; return 0; }
          Stack[SP] = 1.0/Stack[SP];
```

```
            break;
#endif

// Variables:
          default:
            Stack[++SP] = Vars[ByteCode[IP]-VarBegin];
        }
    }

    evalErrorType=0;
    return Stack[SP];
}

#ifdef FUNCTIONPARSER_SUPPORT_DEBUG_OUTPUT
namespace
{
    inline void printHex(std::ostream& dest, unsigned n)
    {
        dest.width(8); dest.fill('0'); hex(dest); //uppercase(dest);
        dest << n;
    }
}

void FunctionParser::PrintByteCode(std::ostream& dest) const
{
    const unsigned* const ByteCode = data->ByteCode;
    const double* const Immed = data->Immed;

    for(unsigned IP=0, DP=0; IP<data->ByteCodeSize; ++IP)
    {
        printHex(dest, IP);
        dest << ": ";
```

```
unsigned opcode = ByteCode[IP];

switch(opcode)
{
  case cIf:
     dest << "jz\t";
     printHex(dest, ByteCode[IP+1]+1);
     dest << endl;
     IP += 2;
     break;

  case cJump:
     dest << "jump\t";
     printHex(dest, ByteCode[IP+1]+1);
     dest << endl;
     IP += 2;
     break;
  case cImmed:
     dest.precision(10);
     dest << "push\t" << Immed[DP++] << endl;
     break;

  case cFCall:
     {
        unsigned index = ByteCode[++IP];
        Data::VarMap_t::const_iterator iter =
           data->FuncPtrNames.begin();
        while(iter->second != index) ++iter;
        dest << "fcall\t" << iter->first
             << " (" << data->FuncPtrs[index].params << ")" << endl;
        break;
```

```
    }

  case cPCall:
    {
      unsigned index = ByteCode[++IP];
      Data::VarMap_t::const_iterator iter =
        data->FuncParserNames.begin();
      while(iter->second != index) ++iter;
      dest << "pcall\t" << iter->first
           << " (" << data->FuncParsers[index]->data->varAmount
           << ")" << endl;
      break;
    }

  default:
    if(opcode < VarBegin)
    {
      string n;
      unsigned params = 1;
      switch(opcode)
      {
        case cNeg: n = "neg"; break;
        case cAdd: n = "add"; break;
        case cSub: n = "sub"; break;
        case cMul: n = "mul"; break;
        case cDiv: n = "div"; break;
        case cMod: n = "mod"; break;
        case cPow: n = "pow"; break;
        case cEqual: n = "eq"; break;
        case cNEqual: n = "neq"; break;
        case cLess: n = "lt"; break;
```

```
                case cLessOrEq: n = "le"; break;
                case cGreater: n = "gt"; break;
                case cGreaterOrEq: n = "ge"; break;
                case cAnd: n = "and"; break;
                case cOr: n = "or"; break;
                case cNot: n = "not"; break;
                case cDeg: n = "deg"; break;
                case cRad: n = "rad"; break;

#ifndef DISABLE_EVAL
                case cEval: n = "call\t0"; break;
#endif

#ifdef SUPPORT_OPTIMIZER
                case cVar: n = "(var)"; break;
                case cDup: n = "dup"; break;
                case cInv: n = "inv"; break;
#endif

                default:
                    n = Functions[opcode-cAbs].name;
                    params = Functions[opcode-cAbs].params;
                }
                dest << n;
                if(params != 1) dest << " (" << params << ")";
                dest << endl;
            }
            else
            {
                dest << "push\tVar" << opcode-VarBegin << endl;
            }
```

```
      }
   }
}
#endif

#ifndef SUPPORT_OPTIMIZER
void FunctionParser::MakeTree(void *) const {}
void FunctionParser::Optimize()
{
   // Do nothing if no optimizations are supported.
}
#endif
```

fpoptimizer

```
//=========================================

// Function parser v2.8 optimizer by Bisqwit
//=========================================

/*
 NOTE!
 ----
 Everything that goes into the #ifndef SUPPORT_OPTIMIZER part
 (ie. when SUPPORT_OPTIMIZER is not defined) should be put in
 the end of fparser.cc file, not in this file.

 Everything in this file should be inside the #ifdef SUPPORT_OPTIMIZER
 block (except the #include "fpconfig.hh" line).
```

```
*/

#include "fpconfig.h"

#ifdef SUPPORT_OPTIMIZER

#include "fparser.h"
#include "fptypes.h"
using namespace FUNCTIONPARSERTYPES;

#include <cmath>
#include <list>
#include <utility>
#include <cassert>
using namespace std;

#ifndef M_PI
#define M_PI 3.14159265358979323846264338327 95
#endif

#define CONSTANT_E     2.71828182845904509080  // exp(1)
#define CONSTANT_PI    M_PI                    // atan2(0,-1)
#define CONSTANT_L10   2.30258509299404590109  // log(10)
#define CONSTANT_L10I  0.43429448190325176116  // 1/log(10)
#define CONSTANT_L10E  CONSTANT_L10I           // log10(e)
#define CONSTANT_L10EI CONSTANT_L10            // 1/log10(e)
#define CONSTANT_DR    (180.0 / M_PI)          // 180/pi
#define CONSTANT_RD    (M_PI / 180.0)          // pi/180

namespace {
inline double Min(double d1, double d2)
{
   return d1<d2 ? d1 : d2;
```

```
}
inline double Max(double d1, double d2)
{
   return d1>d2 ? d1 : d2;
}

class compres
{
   // states: 0=false, 1=true, 2=unknown
public:
   compres(bool b) : state(b) {}
   compres(char v) : state(v) {}
   // is it?
   operator bool() const { return state != 0; }
   // is it not?
   bool operator! () const { return state != 1; }
   bool operator==(bool b) const { return state != !b; }
   bool operator!=(bool b) const { return state != b; }
private:
   char state;
};

const compres maybe = (char)2;

struct CodeTree;

class SubTree
{
   CodeTree *tree;
   bool sign;  // Only possible when parent is cAdd or cMul

   inline void flipsign() { sign = !sign; }
```

```
public:
    SubTree();
    SubTree(double value);
    SubTree(const SubTree &b);
    SubTree(const CodeTree &b);

    ~SubTree();
    const SubTree &operator= (const SubTree &b);
    const SubTree &operator= (const CodeTree &b);

    bool getsign() const { return sign; }

    const CodeTree* operator-> () const { return tree; }
    const CodeTree& operator* () const { return *tree; }
    struct CodeTree* operator-> () { return tree; }
    struct CodeTree& operator* () { return *tree; }

    bool operator< (const SubTree& b) const;
    bool operator== (const SubTree& b) const;
    void Negate(); // Note: Parent must be cAdd
    void Invert(); // Note: Parent must be cMul

    void CheckConstNeg();
    void CheckConstInv();
};

bool IsNegate(const SubTree &p1, const SubTree &p2);
bool IsInverse(const SubTree &p1, const SubTree &p2);

typedef list<SubTree> paramlist;

struct CodeTreeData
{
```

```
    paramlist args;

private:
    unsigned op;       // Operation
    double value;      // In case of cImmed
    unsigned var;      // In case of cVar
    unsigned funcno;   // In case of cFCall, cPCall

public:
    CodeTreeData() : op(cAdd) {}
    ~CodeTreeData() {}

    void SetOp(unsigned newop)     { op=newop; }
    void SetFuncNo(unsigned newno) { funcno=newno; }
    unsigned GetFuncNo() const { return funcno; }

    bool IsFunc() const  { return op == cFCall || op == cPCall; }
    bool IsImmed() const { return op == cImmed; }
    bool IsVar() const   { return op == cVar; }
    inline unsigned GetOp() const { return op; }
    inline double GetImmed() const
    {
      return value;
    }
    inline unsigned GetVar() const
    {
      return var;
    }

    void AddParam(const SubTree &p)
    {
      args.push_back(p);
```

```
}
void SetVar(unsigned v)
{
   args.clear();
   op  = cVar;
   var = v;
}
void SetImmed(double v)
{
   args.clear();
   op       = cImmed;
   value    = orig = v;
   inverted = negated = false;
}
void NegateImmed()
{
   negated = !negated;
   UpdateValue();
}
void InvertImmed()
{
   inverted = !inverted;
   UpdateValue();
}

bool IsOriginal() const { return !(IsInverted() || IsNegated()); }
bool IsInverted() const { return inverted; }
bool IsNegated() const { return negated; }
bool IsInvertedOriginal() const { return IsInverted() && !IsNegated(); }
bool IsNegatedOriginal() const { return !IsInverted() && IsNegated(); }
```

```
private:
   void UpdateValue()
   {
      value = orig;
      if(IsInverted()) { value = 1.0 / value;
                         // FIXME: potential divide by zero.
                        }
      if(IsNegated()) value = -value;
   }

   double orig;
   bool inverted;
   bool negated;
protected:
   // Ensure we don't accidentally copy this
   void operator=(const CodeTreeData &b);
};

class CodeTreeDataPtr
{
   typedef pair<CodeTreeData, unsigned> p_t;
   typedef p_t* pp;
   mutable pp p;

   void Alloc()   const { ++p->second; }
   void Dealloc() const { if(!--p->second) delete p; p = 0; }

   void PrepareForWrite()
   {
      // We're ready if we're the only owner.
      if(p->second == 1) return;
```

```
        // Then make a clone.
        p_t *newtree = new p_t(p->first, 1);
        // Forget the old
        Dealloc();
        // Keep the new
        p = newtree;
    }

public:
    CodeTreeDataPtr() : p(new p_t) { p->second = 1; }
    CodeTreeDataPtr(const CodeTreeDataPtr &b): p(b.p) { Alloc(); }
    ~CodeTreeDataPtr() { Dealloc(); }
    const CodeTreeDataPtr &operator= (const CodeTreeDataPtr &b)
    {
        b.Alloc();
        Dealloc();
        p = b.p;
        return *this;
    }
    const CodeTreeData *operator-> () const { return &p->first; }
    const CodeTreeData &operator*  () const { return p->first; }
    CodeTreeData *operator-> () { PrepareForWrite(); return &p->first; }
    CodeTreeData &operator*  () { PrepareForWrite(); return p->first; }

    void Shock();
};

#define CHECKCONSTNEG(item, op) \
    ((op)==cMul) \
      ? (item).CheckConstInv() \
      : (item).CheckConstNeg()
```

```
struct CodeTree
{
   CodeTreeDataPtr data;

private:
   typedef paramlist::iterator pit;
   typedef paramlist::const_iterator pcit;

   /*
   template<unsigned v> inline void chk() const
   {
   }
   */

public:
   const pcit GetBegin() const { return data->args.begin(); }
   const pcit GetEnd()   const { return data->args.end(); }
   const pit GetBegin() { return data->args.begin(); }
   const pit GetEnd()   { return data->args.end(); }
   const SubTree& getp0() const { /*chk<1>();*/pcit tmp=GetBegin();              return
*tmp; }
   const SubTree& getp1() const { /*chk<2>();*/pcit tmp=GetBegin(); ++tmp;       return
*tmp; }
   const SubTree& getp2() const { /*chk<3>();*/pcit tmp=GetBegin(); ++tmp; ++tmp;
return *tmp; }
   unsigned GetArgCount() const { return data->args.size(); }
   void Erase(const pit p)      { data->args.erase(p); }

   SubTree& getp0() { /*chk<1>();*/pit tmp=GetBegin();              return *tmp; }
   SubTree& getp1() { /*chk<2>();*/pit tmp=GetBegin(); ++tmp;       return *tmp; }
   SubTree& getp2() { /*chk<3>();*/pit tmp=GetBegin(); ++tmp; ++tmp; return *tmp; }
```

```
    // set
    void SetImmed(double v) { data->SetImmed(v); }
    void SetOp(unsigned op) { data->SetOp(op); }
    void SetVar(unsigned v) { data->SetVar(v); }
    // get
    double GetImmed() const { return data->GetImmed(); }
    unsigned GetVar() const { return data->GetVar(); }
    unsigned GetOp() const  { return data->GetOp(); }
    // test
    bool IsImmed() const { return data->IsImmed(); }
    bool IsVar()   const { return data->IsVar(); }
    // act
    void AddParam(const SubTree &p) { data->AddParam(p); }
    void NegateImmed() { data->NegateImmed(); } // don't use when op!=cImmed
    void InvertImmed() { data->InvertImmed(); } // don't use when op!=cImmed

    compres NonZero() const { if(!IsImmed()) return maybe;
                              return GetImmed() != 0.0; }
    compres IsPositive() const { if(!IsImmed()) return maybe;
                                 return GetImmed() > 0.0; }

private:
    struct ConstList
    {
       double voidvalue;
       list<pit> cp;
       double value;
       unsigned size() const { return cp.size(); }
    };
    struct ConstList BuildConstList();
    void KillConst(const ConstList &cl)
```

```
    {
        for(list<pit>::const_iterator i=cl.cp.begin(); i!=cl.cp.end(); ++i)
            Erase(*i);
    }
    void FinishConst(const ConstList &cl)
    {
        if(cl.value != cl.voidvalue && cl.size() > 1) AddParam(cl.value);
        if(cl.value == cl.voidvalue || cl.size() > 1) KillConst(cl);
    }

public:
    CodeTree() {}
    CodeTree(double v) { SetImmed(v); }

    CodeTree(unsigned op, const SubTree &p)
    {
        SetOp(op);
        AddParam(p);
    }
    CodeTree(unsigned op, const SubTree &p1, const SubTree &p2)
    {
        SetOp(op);
        AddParam(p1);
        AddParam(p2);
    }

    bool operator== (const CodeTree& b) const;
    bool operator< (const CodeTree& b) const;

private:
    bool IsSortable() const
    {
```

```
    switch(GetOp())
    {
      case cAdd:  case cMul:
      case cEqual:
      case cAnd: case cOr:
      case cMax: case cMin:
        return true;
      default:
        return false;
    }
}
void SortIfPossible()
{
    if(IsSortable())
    {
      data->args.sort();
    }
}

void ReplaceWithConst(double value)
{
    SetImmed(value);

    /* REMEMBER TO CALL CheckConstInv / CheckConstNeg
     * FOR PARENT SubTree, OR MAYHEM HAPPENS
     */
}

void ReplaceWith(const CodeTree &b)
{
    // If b is child of *this, mayhem
```

```
    // happens. So we first make a clone
    // and then proceed with copy.
    CodeTreeDataPtr tmp = b.data;
    tmp.Shock();
    data = tmp;
}

void ReplaceWith(unsigned op, const SubTree &p)
{
    ReplaceWith(CodeTree(op, p));
}

void ReplaceWith(unsigned op, const SubTree &p1, const SubTree &p2)
{
    ReplaceWith(CodeTree(op, p1, p2));
}

void OptimizeConflict()
{
    // This optimization does this: x-x = 0, x/x = 1, a+b-a = b.

    if(GetOp() == cAdd || GetOp() == cMul)
    {
    Redo:
        pit a, b;
        for(a=GetBegin(); a!=GetEnd(); ++a)
        {
            for(b=GetBegin(); ++b != GetEnd(); )
            {
                const SubTree &p1 = *a;
                const SubTree &p2 = *b;
```

```
            if(GetOp() == cMul ? IsInverse(p1,p2)
                               : IsNegate(p1,p2))
            {
               // These parameters complement each others out
               Erase(b);
               Erase(a);
               goto Redo;
            }
         }
      }
   }
   OptimizeRedundant();
}

void OptimizeRedundant()
{
   // This optimization does this: min()=0, max()=0, add()=0, mul()=1

   if(!GetArgCount())
   {
      if(GetOp() == cAdd || GetOp() == cMin || GetOp() == cMax)
         ReplaceWithConst(0);
      else if(GetOp() == cMul)
         ReplaceWithConst(1);
      return;
   }

   // And this: mul(x) = x, min(x) = x, max(x) = x, add(x) = x

   if(GetArgCount() == 1)
   {
      if(GetOp() == cMul || GetOp() == cAdd || GetOp() == cMin || GetOp() == cMax)
```

```
            if(!getp0().getsign())
                ReplaceWith(*getp0());
        }

        OptimizeDoubleNegations();
    }

    void OptimizeDoubleNegations()
    {
        if(GetOp() == cAdd)
        {
            // Eschew double negations

            // If any of the elements is cMul
            // and has a numeric constant, negate
            // the constant and negate sign.

            for(pit a=GetBegin(); a!=GetEnd(); ++a)
            {
                SubTree &pa = *a;
                if(pa.getsign()
                && pa->GetOp() == cMul)
                {
                    CodeTree &p = *pa;
                    for(pit b=p.GetBegin();
                            b!=p.GetEnd(); ++b)
                    {
                        SubTree &pb = *b;
                        if(pb->IsImmed())
                        {
                            pb.Negate();
                            pa.Negate();
```

```
                    break;
                }
            }
        }
    }

    if(GetOp() == cMul)
    {
        // If any of the elements is cPow
        // and has a numeric exponent, negate
        // the exponent and negate sign.

        for(pit a=GetBegin(); a!=GetEnd(); ++a)
        {
            SubTree &pa = *a;
            if(pa.getsign() && pa->GetOp() == cPow)
            {
                CodeTree &p = *pa;
                if(p.getp1()->IsImmed())
                {
                    // negate ok for pow when op=cImmed
                    p.getp1().Negate();
                    pa.Negate();
                }
            }
        }
    }
}

void OptimizeConstantMath1()
```

```
{
  // This optimization does three things:
  //   - For adding groups:
  //       Constants are added together.
  //   - For multiplying groups:
  //       Constants are multiplied together.
  //   - For function calls:
  //       If all parameters are constants,
  //       the call is replaced with constant value.

  // First, do this:
  OptimizeAddMulFlat();

  switch(GetOp())
  {
    case cAdd:
    {
      ConstList cl = BuildConstList();
      FinishConst(cl);
      break;
    }
    case cMul:
    {
      ConstList cl = BuildConstList();

      if(cl.value == 0.0) ReplaceWithConst(0.0);
      else FinishConst(cl);

      break;
    }
    #define ConstantUnaryFun(token, fun) \
      case token: { const SubTree &p0 = getp0(); \
```

```
            if(p0->IsImmed()) ReplaceWithConst(fun(p0->GetImmed())); \
            break; }
        #define ConstantBinaryFun(token, fun) \
           case token: { const SubTree &p0 = getp0(); \
                   const SubTree &p1 = getp1(); \
            if(p0->IsImmed() && \
              p1->IsImmed()) ReplaceWithConst(fun(p0->GetImmed(), p1-
>GetImmed())); \
            break; }

        // FIXME: potential invalid parameters for functions
        //       can cause exceptions here

        ConstantUnaryFun(cAbs,   fabs);
        ConstantUnaryFun(cAcos,  acos);
        ConstantUnaryFun(cAsin,  asin);
        ConstantUnaryFun(cAtan,  atan);
        ConstantUnaryFun(cCeil,  ceil);
        ConstantUnaryFun(cCos,   cos);
        ConstantUnaryFun(cCosh,  cosh);
        ConstantUnaryFun(cFloor, floor);
        ConstantUnaryFun(cLog,   log);
        ConstantUnaryFun(cSin,   sin);
        ConstantUnaryFun(cSinh,  sinh);
        ConstantUnaryFun(cTan,   tan);
        ConstantUnaryFun(cTanh,  tanh);
        ConstantBinaryFun(cAtan2, atan2);
        ConstantBinaryFun(cMax,   Max);
        ConstantBinaryFun(cMin,   Min);
        ConstantBinaryFun(cMod,   fmod); // not a func, but belongs here too
        ConstantBinaryFun(cPow,   pow);
```

```
      case cNeg:
      case cSub:
      case cDiv:
        /* Unreached (nonexistent operator)
         * TODO: internal error here?
         */
        break;

      case cCot:
      case cCsc:
      case cSec:
      case cDeg:
      case cRad:
      case cLog10:
      case cSqrt:
      case cExp:
        /* Unreached (nonexistent function)
         * TODO: internal error here?
         */
         break;
    }

    OptimizeConflict();
}

void OptimizeAddMulFlat()
{
    // This optimization flattens the topography of the tree.
    //   Examples:
    //       x + (y+z) = x+y+z
    //       x * (y/z) = x*y/z
```

```
//       x / (y/z) = x/y*z

if(GetOp() == cAdd || GetOp() == cMul)
{
   // If children are same type as parent add them here
   for(pit b, a=GetBegin(); a!=GetEnd(); a=b)
   {
      const SubTree &pa = *a;  b=a; ++b;
      if(pa->GetOp() != GetOp()) continue;

      // Child is same type
      for(pcit c=pa->GetBegin();
            c!=pa->GetEnd();
            ++c)
      {
         const SubTree &pb = *c;
         if(pa.getsign())
         {
            // +a -(+b +c)
            // means b and c will be negated

            SubTree tmp = pb;
            if(GetOp() == cMul)
               tmp.Invert();
            else
               tmp.Negate();
            AddParam(tmp);
         }
         else
            AddParam(pb);
      }
```

```
            Erase(a);

            // Note: OptimizeConstantMath1() would be a good thing to call next.
        }
    }
}

void OptimizeLinearCombine()
{
    // This optimization does the following:
    //
    //   x*x*x*x -> x^4
    //   x+x+x+x -> x*4
    //   x*x     -> x^2
    //   x/z/z   ->
    //

    // Remove conflicts first, so we don't have to worry about signs.
    OptimizeConflict();

    bool didchanges = false;
    if(GetOp() == cAdd || GetOp() == cMul)
    {
    Redo:
        for(pit a=GetBegin(); a!=GetEnd(); ++a)
        {
            const SubTree &pa = *a;

            list<pit> poslist;

            for(pit b=a; ++b!=GetEnd(); )
            {
```

```
    const SubTree &pb = *b;
    if(*pa == *pb)
      poslist.push_back(b);
}

unsigned min = 2;
if(poslist.size() >= min)
{
    SubTree arvo = pa;
    bool negate = arvo.getsign();

    double factor = poslist.size() + 1;

    if(negate)
    {
      arvo.Negate();
      factor = -factor;
    }

    CodeTree tmp(GetOp()==cAdd ? cMul : cPow,
                 arvo,
                 factor);

    list<pit>::const_iterator j;
    for(j=poslist.begin(); j!=poslist.end(); ++j)
      Erase(*j);
    poslist.clear();

    *a = tmp;
    didchanges = true;
    goto Redo;
}
```

```
      }
    }
    if(didchanges)
    {
      // As a result, there might be need for this:
      OptimizeAddMulFlat();
      // And this:
      OptimizeRedundant();
    }
}

void OptimizeLogarithm()
{
  /*
    This is basic logarithm math:
      pow(X,Y)/log(Y) = X
      log(X)/log(Y) = logY(X)
      log(X^Y)      = log(X)*Y
      log(X*Y)      = log(X)+log(Y)
      exp(log(X)*Y) = X^Y

    This function does these optimizations:
       pow(const_E, log(x))   = x
       pow(const_E, log(x)*y) = x^y
       pow(10,      log(x)*const_L10I*y) = x^y
       pow(z,       log(x)/log(z)*y)     = x^y

    And this:
       log(x^z)            = z * log(x)
    Which automatically causes these too:
       log(pow(const_E, x))        = x
```

```
      log(pow(y,       x))        = x * log(y)
      log(pow(pow(const_E, y), x)) = x*y

   And it does this too:
      log(x) + log(y) + log(z) = log(x * y * z)
      log(x * exp(y)) = log(x) + y

*/

// Must be already in exponential form.

// Optimize exponents before doing something.
OptimizeExponents();

if(GetOp() == cLog)
{
   // We should have one parameter for log() function.
   // If we don't, we're screwed.

   const SubTree &p = getp0();

   if(p->GetOp() == cPow)
   {
      // Found log(x^y)
      SubTree p0 = p->getp0(); // x
      SubTree p1 = p->getp1(); // y

      // Build the new logarithm.
      CodeTree tmp(GetOp(), p0);  // log(x)

      // Become log(x) * y
      ReplaceWith(cMul, tmp, p1);
   }
```

```
else if(p->GetOp() == cMul)
{
  // Redefine &p nonconst
  SubTree &p = getp0();

  p->OptimizeAddMulFlat();
  p->OptimizeExponents();
  CHECKCONSTNEG(p, p->GetOp());

  list<SubTree> adds;

  for(pit b, a = p->GetBegin();
          a != p->GetEnd(); a=b)
  {
    SubTree &pa = *a;  b=a; ++b;
    if(pa->GetOp() == cPow
    && pa->getp0()->IsImmed()
    && pa->getp0()->GetImmed() == CONSTANT_E)
    {
      adds.push_back(pa->getp1());
      p->Erase(a);
      continue;
    }
  }
  if(adds.size())
  {
    CodeTree tmp(cAdd, *this);

    list<SubTree>::const_iterator i;
    for(i=adds.begin(); i!=adds.end(); ++i)
      tmp.AddParam(*i);
```

```
            ReplaceWith(tmp);
        }
    }
}
if(GetOp() == cAdd)
{
    // Check which ones are logs.
    list<pit> poslist;

    for(pit a=GetBegin(); a!=GetEnd(); ++a)
    {
        const SubTree &pa = *a;
        if(pa->GetOp() == cLog)
            poslist.push_back(a);
    }

    if(poslist.size() >= 2)
    {
        CodeTree tmp(cMul, 1.0); // eek

        list<pit>::const_iterator j;
        for(j=poslist.begin(); j!=poslist.end(); ++j)
        {
            const SubTree &pb = **j;
            // Take all of its children
            for(pcit b=pb->GetBegin();
                    b!=pb->GetEnd();
                    ++b)
            {
                SubTree tmp2 = *b;
                if(pb.getsign()) tmp2.Negate();
```

```
            tmp.AddParam(tmp2);
          }
          Erase(*j);
        }
        poslist.clear();

        AddParam(CodeTree(cLog, tmp));
    }
    // Done, hopefully
}
if(GetOp() == cPow)
{
    const SubTree &p0 = getp0();
    SubTree &p1 = getp1();

    if(p0->IsImmed() && p0->GetImmed() == CONSTANT_E
    && p1->GetOp() == cLog)
    {
        // pow(const_E, log(x)) = x
        ReplaceWith(*(p1->getp0()));
    }
    else if(p1->GetOp() == cMul)
    {
        //bool didsomething = true;

        pit poslogpos; bool foundposlog = false;
        pit neglogpos; bool foundneglog = false;

        ConstList cl = p1->BuildConstList();

        for(pit a=p1->GetBegin(); a!=p1->GetEnd(); ++a)
        {
```

```
    const SubTree &pa = *a;
    if(pa->GetOp() == cLog)
    {
      if(!pa.getsign())
      {
        foundposlog = true;
        poslogpos   = a;
      }
      else if(*p0 == *(pa->getp0()))
      {
        foundneglog = true;
        neglogpos   = a;
      }
    }
}

if(p0->IsImmed()
&& p0->GetImmed() == 10.0
&& cl.value == CONSTANT_L10I
&& foundposlog)
{
    SubTree base = (*poslogpos)->getp0();
    p1->KillConst(cl);
    p1->Erase(poslogpos);
    p1->OptimizeRedundant();
    SubTree mul = p1;

    ReplaceWith(cPow, base, mul);

    // FIXME: what optimizations should be done now?
    return;
```

```
}

// Put back the constant
FinishConst(cl);

if(p0->IsImmed()
&& p0->GetImmed() == CONSTANT_E
&& foundposlog)
{
   SubTree base = (*poslogpos)->getp0();
   p1->Erase(poslogpos);

   p1->OptimizeRedundant();
   SubTree mul = p1;

   ReplaceWith(cPow, base, mul);

   // FIXME: what optimizations should be done now?
   return;
}

if(foundposlog
&& foundneglog
&& *((*neglogpos)->getp0()) == *p0)
{
   SubTree base = (*poslogpos)->getp0();
   p1->Erase(poslogpos);
   p1->Erase(neglogpos);

   p1->OptimizeRedundant();
   SubTree mul = p1;

   ReplaceWith(cPow, base, mul);
```

```
            // FIXME: what optimizations should be done now?
            return;
          }
        }
    }
}

void OptimizeFunctionCalls()
{
    /* Goals: sin(asin(x)) = x
     *        cos(acos(x)) = x
     *        tan(atan(x)) = x
     * NOTE:
     *   Do NOT do these:
     *     asin(sin(x))
     *     acos(cos(x))
     *     atan(tan(x))
     *   Because someone might want to wrap the angle.
     */
    // FIXME: TODO
}

void OptimizePowMulAdd()
{
    // x^3 * x -> x^4
    // x*3 + x -> x*4
    // FIXME: Do those

    // x^1 -> x
    if(GetOp() == cPow)
    {
```

```
      const SubTree &base     = getp0();
      const SubTree &exponent = getp1();

      if(exponent->IsImmed())
      {
        if(exponent->GetImmed() == 1.0)
          ReplaceWith(*base);
        else if(exponent->GetImmed() == 0.0
           && base->NonZero())
          ReplaceWithConst(1.0);
      }
    }
}

void OptimizeExponents()
{
  /* Goals:
   *     (x^y)^z   -> x^(y*z)
   *     x^y * x^z -> x^(y+z)
   */
  // First move to exponential form.
  OptimizeLinearCombine();

  bool didchanges = false;

Redo:
  if(GetOp() == cPow)
  {
    // (x^y)^z   -> x^(y*z)

    const SubTree &p0 = getp0();
    const SubTree &p1 = getp1();
```

```
    if(p0->GetOp() == cPow)
    {
      CodeTree tmp(cMul, p0->getp1(), p1);
      tmp.Optimize();

      ReplaceWith(cPow, p0->getp0(), tmp);

      didchanges = true;
      goto Redo;
    }
  }
  if(GetOp() == cMul)
  {
    // x^y * x^z -> x^(y+z)

    for(pit a=GetBegin(); a!=GetEnd(); ++a)
    {
      const SubTree &pa = *a;

      if(pa->GetOp() != cPow) continue;

      list<pit> poslist;

      for(pit b=a; ++b != GetEnd(); )
      {
        const SubTree &pb = *b;
        if(pb->GetOp() == cPow
        && *(pa->getp0())
        == *(pb->getp0()))
        {
          poslist.push_back(b);
        }
```

```
}

if(poslist.size() >= 1)
{
   poslist.push_back(a);

   CodeTree base = *(pa->getp0());

   CodeTree exponent(cAdd, 0.0); //eek

   // Collect all exponents to cAdd
   list<pit>::const_iterator i;
   for(i=poslist.begin(); i!=poslist.end(); ++i)
   {
      const SubTree &pb = **i;

      SubTree tmp2 = pb->getp1();
      if(pb.getsign()) tmp2.Invert();

      exponent.AddParam(tmp2);
   }

   exponent.Optimize();

   CodeTree result(cPow, base, exponent);

   for(i=poslist.begin(); i!=poslist.end(); ++i)
      Erase(*i);
   poslist.clear();

   AddParam(result); // We're cMul, remember

   didchanges = true;
   goto Redo;
```

```
            }
          }
        }

        OptimizePowMulAdd();

        if(didchanges)
        {
          // As a result, there might be need for this:
          OptimizeConflict();
        }
      }

      void OptimizeLinearExplode()
      {
        // x^2 -> x*x
        // But only if x is just a simple thing

        // Won't work on anything else.
        if(GetOp() != cPow) return;

        // TODO TODO TODO
      }

      void OptimizePascal()
      {
#if 0    // Too big, too specific, etc

        // Won't work on anything else.
        if(GetOp() != cAdd) return;

        // Must be done after OptimizeLinearCombine();
```

```
        // Don't need pascal triangle
        // Coefficient for x^a * y^b * z^c = 3! / (a! * b! * c!)

        // We are greedy and want other than just binomials
        // FIXME

        // note: partial ones are also nice
        //     x*x + x*y + y*y
        //   = (x+y)^2 - x*y
        //
        //     x x * x y * + y y * +
        // ->  x y + dup * x y * -
#endif
    }

public:

    void Optimize();

    void Assemble(vector<unsigned> &byteCode,
                  vector<double>   &immed) const;

    void FinalOptimize()
    {
        // First optimize each parameter.
        for(pit a=GetBegin(); a!=GetEnd(); ++a)
            (*a)->FinalOptimize();

        /* These things are to be done:
         *
         * x * CONSTANT_DR       -> cDeg(x)
         * x * CONSTANT_RD       -> cRad(x)
         * pow(x, 0.5)           -> sqrt(x)
```

```
 * log(x) * CONSTANT_L10I -> log10(x)
 * pow(CONSTANT_E, x)     -> exp(x)
 * inv(sin(x))           -> csc(x)
 * inv(cos(x))           -> sec(x)
 * inv(tan(x))           -> cot(x)
 */

if(GetOp() == cPow)
{
   const SubTree &p0 = getp0();
   const SubTree &p1 = getp1();
   if(p0->GetOp()    == cImmed
   && p0->GetImmed() == CONSTANT_E)
   {
      ReplaceWith(cExp, p1);
   }
   else if(p1->GetOp()    == cImmed
        && p1->GetImmed() == 0.5)
   {
      ReplaceWith(cSqrt, p0);
   }
}
if(GetOp() == cMul)
{
   if(GetArgCount() == 1 && getp0().getsign())
   {
      /***/if(getp0()->GetOp() == cSin)ReplaceWith(cCsc, getp0()->getp0());
      else if(getp0()->GetOp() == cCos)ReplaceWith(cSec, getp0()->getp0());
      else if(getp0()->GetOp() == cTan)ReplaceWith(cCot, getp0()->getp0());
   }
}
```

```
// Separate "if", because op may have just changed
if(GetOp() == cMul)
{
  CodeTree *found_log = 0;

  ConstList cl = BuildConstList();

  for(pit a=GetBegin(); a!=GetEnd(); ++a)
  {
    SubTree &pa = *a;
    if(pa->GetOp() == cLog && !pa.getsign())
      found_log = &*pa;
  }
  if(cl.value == CONSTANT_L10I && found_log)
  {
    // Change the log() to log10()
    found_log->SetOp(cLog10);
    // And forget the constant
    KillConst(cl);
  }
  else if(cl.value == CONSTANT_DR)
  {
    OptimizeRedundant();
    ReplaceWith(cDeg, *this);
  }
  else if(cl.value == CONSTANT_RD)
  {
    OptimizeRedundant();
    ReplaceWith(cRad, *this);
  }
  else FinishConst(cl);
```

```
        }

        SortIfPossible();
    }
};

void CodeTreeDataPtr::Shock()
{
 /*
    PrepareForWrite();
    paramlist &p2 = (*this)->args;
    for(paramlist::iterator i=p2.begin(); i!=p2.end(); ++i)
    {
        (*i)->data.Shock();
    }
 */
}

CodeTree::ConstList CodeTree::BuildConstList()
{
    ConstList result;
    result.value     =
    result.voidvalue = GetOp()==cMul ? 1.0 : 0.0;

    list<pit> &cp = result.cp;
    for(pit b, a=GetBegin(); a!=GetEnd(); a=b)
    {
        SubTree &pa = *a;  b=a; ++b;
        if(!pa->IsImmed()) continue;

        double thisvalue = pa->GetImmed();
        if(thisvalue == result.voidvalue)
```

```
    {
      // This value is no good, forget it
      Erase(a);
      continue;
    }
    if(GetOp() == cMul)
      result.value *= thisvalue;
    else
      result.value += thisvalue;
    cp.push_back(a);
  }
  if(GetOp() == cMul)
  {
    /*
     Jos joku niistδ arvoista on -1 eikδ se ole ainoa arvo,
     niin joku muu niistδ arvoista negatoidaan.
    */
    for(bool done=false; cp.size() > 1 && !done; )
    {
      done = true;
      for(list<pit>::iterator b,a=cp.begin(); a!=cp.end(); a=b)
      {
        b=a; ++b;
        if((**a)->GetImmed() == -1.0)
        {
          Erase(*a);
          cp.erase(a);

          // take randomly something
          (**cp.begin())->data->NegateImmed();
          if(cp.size() < 2)break;
```

```
                done = false;
              }
            }
          }
    }
    return result;
}

void CodeTree::Assemble
   (vector<unsigned> &byteCode,
    vector<double>   &immed) const
{
    #define AddCmd(op) byteCode.push_back((op))
    #define AddConst(v) do { \
        byteCode.push_back(cImmed); \
        immed.push_back((v)); \
    } while(0)

    if(IsVar())
    {
        AddCmd(GetVar());
        return;
    }
    if(IsImmed())
    {
        AddConst(GetImmed());
        return;
    }

    switch(GetOp())
    {
```

```
case cAdd:
case cMul:
{
   unsigned opcount = 0;
   for(pcit a=GetBegin(); a!=GetEnd(); ++a)
   {
      const SubTree &pa = *a;

      if(opcount < 2) ++opcount;

      bool pnega = pa.getsign();

      bool done = false;
      if(pa->IsImmed())
      {
         if(GetOp() == cMul
         && pa->data->IsInverted()
         && (pnega || opcount==2)
          )
         {
            CodeTree tmp = *pa;
            tmp.data->InvertImmed();
            tmp.Assemble(byteCode, immed);
            pnega = !pnega;
            done = true;
         }
         else if(GetOp() == cAdd
         && (pa->data->IsNegatedOriginal()
      //     || pa->GetImmed() < 0
            )
         && (pnega || opcount==2)
```

```
            )
        {
            CodeTree tmp = *pa;
            tmp.data->NegateImmed();
            tmp.Assemble(byteCode, immed);
            pnega = !pnega;
            done = true;
        }
    }
    if(!done)
        pa->Assemble(byteCode, immed);

    if(opcount == 2)
    {
        unsigned tmpop = GetOp();
        if(pnega) // negate
        {
            tmpop = (tmpop == cMul) ? cDiv : cSub;
        }
        AddCmd(tmpop);
    }
    else if(pnega)
    {
        if(GetOp() == cMul) AddCmd(cInv);
        else AddCmd(cNeg);
    }
  }
  break;
}
case cIf:
{
```

```
    // If the parameter amount is != 3, we're screwed.
    getp0()->Assemble(byteCode, immed);

    unsigned ofs = byteCode.size();
    AddCmd(cIf);
    AddCmd(0); // code index
    AddCmd(0); // immed index

    getp1()->Assemble(byteCode, immed);

    byteCode[ofs+1] = byteCode.size()+2;
    byteCode[ofs+2] = immed.size();

    ofs = byteCode.size();
    AddCmd(cJump);
    AddCmd(0); // code index
    AddCmd(0); // immed index

    getp2()->Assemble(byteCode, immed);

    byteCode[ofs+1] = byteCode.size()-1;
    byteCode[ofs+2] = immed.size();

    break;
}
case cFCall:
{
    // If the parameter count is invalid, we're screwed.
    for(pcit a=GetBegin(); a!=GetEnd(); ++a)
    {
        const SubTree &pa = *a;
        pa->Assemble(byteCode, immed);
    }
```

```
            AddCmd(GetOp());
            AddCmd(data->GetFuncNo());
            break;
        }
        case cPCall:
        {
            // If the parameter count is invalid, we're screwed.
            for(pcit a=GetBegin(); a!=GetEnd(); ++a)
            {
                const SubTree &pa = *a;
                pa->Assemble(byteCode, immed);
            }
            AddCmd(GetOp());
            AddCmd(data->GetFuncNo());
            break;
        }
        default:
        {
            // If the parameter count is invalid, we're screwed.
            for(pcit a=GetBegin(); a!=GetEnd(); ++a)
            {
                const SubTree &pa = *a;
                pa->Assemble(byteCode, immed);
            }
            AddCmd(GetOp());
            break;
        }
    }
}

void CodeTree::Optimize()
```

```
{
   // Phase:
   //   Phase 0: Do local optimizations.
   //   Phase 1: Optimize each.
   //   Phase 2: Do local optimizations again.

   for(unsigned phase=0; phase<=2; ++phase)
   {
      if(phase == 1)
      {
         // Optimize each parameter.
         for(pit a=GetBegin(); a!=GetEnd(); ++a)
         {
            (*a)->Optimize();
            CHECKCONSTNEG(*a, GetOp());
         }
         continue;
      }
      if(phase == 0 || phase == 2)
      {
         // Do local optimizations.

         OptimizeConstantMath1();
         OptimizeLogarithm();
         OptimizeFunctionCalls();
         OptimizeExponents();
         OptimizeLinearExplode();
         OptimizePascal();

         /* Optimization paths:

            doublenegations=
```

```
            redundant= * doublenegations
            conflict= * redundant
            addmulflat=
            constantmath1= addmulflat * conflict
            linearcombine= conflict * addmulflatH redundantH
            powmuladd=
            exponents= linearcombine * powmuladd conflictH
            logarithm= exponents *
            functioncalls= IDLE
            linearexplode= IDLE
            pascal= IDLE

            * = actions here
            H = only if made changes
          */
        }
    }
}

bool CodeTree::operator== (const CodeTree& b) const
{
    if(GetOp() != b.GetOp()) return false;
    if(IsImmed()) if(GetImmed()  != b.GetImmed())  return false;
    if(IsVar())   if(GetVar()    != b.GetVar())    return false;
    if(data->IsFunc())
        if(data->GetFuncNo() != b.data->GetFuncNo()) return false;
    return data->args == b.data->args;
}

bool CodeTree::operator< (const CodeTree& b) const
{
```

```
if(GetArgCount() != b.GetArgCount())
   return GetArgCount() > b.GetArgCount();

if(GetOp() != b.GetOp())
{
   // sort immeds last
   if(IsImmed() != b.IsImmed()) return IsImmed() < b.IsImmed();

   return GetOp() < b.GetOp();
}

if(IsImmed())
{
   if(GetImmed() != b.GetImmed()) return GetImmed() < b.GetImmed();
}
if(IsVar() && GetVar() != b.GetVar())
{
   return GetVar() < b.GetVar();
}
if(data->IsFunc() && data->GetFuncNo() != b.data->GetFuncNo())
{
   return data->GetFuncNo() < b.data->GetFuncNo();
}

pcit i = GetBegin(), j = b.GetBegin();
for(; i != GetEnd(); ++i, ++j)
{
   const SubTree &pa = *i, &pb = *j;

   if(!(pa == pb))
      return pa < pb;
}
```

```
    return false;
}

bool IsNegate(const SubTree &p1, const SubTree &p2) /*const */
{
    if(p1->IsImmed() && p2->IsImmed())
    {
        return p1->GetImmed() == -p2->GetImmed();
    }
    if(p1.getsign() == p2.getsign()) return false;
    return *p1 == *p2;
}
bool IsInverse(const SubTree &p1, const SubTree &p2) /*const*/
{
    if(p1->IsImmed() && p2->IsImmed())
    {
        // FIXME: potential divide by zero.
        return p1->GetImmed() == 1.0 / p2->GetImmed();
    }
    if(p1.getsign() == p2.getsign()) return false;
    return *p1 == *p2;
}

SubTree::SubTree() : tree(new CodeTree), sign(false)
{
}

SubTree::SubTree(const SubTree &b) : tree(new CodeTree(*b.tree)), sign(b.sign)
{
}

#define SubTreeDecl(p1, p2) \
```

```
   SubTree::SubTree p1 : tree(new CodeTree p2), sign(false) { }

SubTreeDecl( (const CodeTree &b), (b) )
SubTreeDecl( (double value),          (value) )

#undef SubTreeDecl

SubTree::~SubTree()
{
   delete tree; tree=0;
}

const SubTree &SubTree::operator= (const SubTree &b)
{
   sign = b.sign;
   CodeTree *oldtree = tree;
   tree = new CodeTree(*b.tree);
   delete oldtree;
   return *this;
}
const SubTree &SubTree::operator= (const CodeTree &b)
{
   sign = false;
   CodeTree *oldtree = tree;
   tree = new CodeTree(b);
   delete oldtree;
   return *this;
}

bool SubTree::operator< (const SubTree& b) const
{
   if(getsign() != b.getsign()) return getsign() < b.getsign();
```

```
    return *tree < *b.tree;
}
bool SubTree::operator== (const SubTree& b) const
{
    return sign == b.sign && *tree == *b.tree;
}
void SubTree::Negate() // Note: Parent must be cAdd
{
    flipsign();
    CheckConstNeg();
}
void SubTree::CheckConstNeg()
{
    if(tree->IsImmed() && getsign())
    {
        tree->NegateImmed();
        sign = false;
    }
}
void SubTree::Invert() // Note: Parent must be cMul
{
    flipsign();
    CheckConstInv();
}
void SubTree::CheckConstInv()
{
    if(tree->IsImmed() && getsign())
    {
        tree->InvertImmed();
        sign = false;
```

```
    }
}

}//namespace

void FunctionParser::MakeTree(void *r) const
{
    // Dirty hack. Should be fixed.
    CodeTree* result = static_cast<CodeTree*>(r);

    vector<CodeTree> stack(1);

    #define GROW(n) do { \
        stacktop += n; \
        if(stack.size() <= stacktop) stack.resize(stacktop+1); \
    } while(0)

    #define EAT(n, opcode) do { \
        unsigned newstacktop = stacktop-n; \
        if((n) == 0) GROW(1); \
        stack[stacktop].SetOp((opcode)); \
        for(unsigned a=0, b=(n); a<b; ++a) \
            stack[stacktop].AddParam(stack[newstacktop+a]); \
        stack.erase(stack.begin() + newstacktop, \
                    stack.begin() + stacktop); \
        stacktop = newstacktop; GROW(1); \
    } while(0)

    #define ADDCONST(n) do { \
        stack[stacktop].SetImmed((n)); \
        GROW(1); \
    } while(0)
```

```
unsigned stacktop=0;

list<unsigned> labels;

const unsigned* const ByteCode = data->ByteCode;
const unsigned ByteCodeSize = data->ByteCodeSize;
const double* const Immed = data->Immed;

for(unsigned IP=0, DP=0; ; ++IP)
{
  while(labels.size() > 0
  && *labels.begin() == IP)
  {
    // The "else" of an "if" ends here
    EAT(3, cIf);
    labels.erase(labels.begin());
  }

  if(IP >= ByteCodeSize)
  {
    break;
  }

  unsigned opcode = ByteCode[IP];

  if(opcode == cIf)
  {
    IP += 2;
  }
  else if(opcode == cJump)
  {
    labels.push_front(ByteCode[IP+1]+1);
```

```
    IP += 2;
}
else if(opcode == cImmed)
{
    ADDCONST(Immed[DP++]);
}
else if(opcode < VarBegin)
{
    switch(opcode)
    {
        // Unary operators
        case cNeg:
        {
            EAT(1, cAdd); // Unary minus is negative adding.
            stack[stacktop-1].getp0().Negate();
            break;
        }
        // Binary operators
        case cSub:
        {
            EAT(2, cAdd); // Minus is negative adding
            stack[stacktop-1].getp1().Negate();
            break;
        }
        case cDiv:
        {
            EAT(2, cMul); // Divide is inverse multiply
            stack[stacktop-1].getp1().Invert();
            break;
        }
```

```
// ADD ALL TWO PARAMETER NON-FUNCTIONS HERE
case cAdd: case cMul:
case cMod: case cPow:
case cEqual: case cLess: case cGreater:
case cNEqual: case cLessOrEq: case cGreaterOrEq:
case cAnd: case cOr:
   EAT(2, opcode);
   break;

// ADD ALL UNARY NON-FUNCTIONS HERE
case cNot:
   EAT(1, opcode);
   break;

case cFCall:
{
   unsigned index = ByteCode[++IP];
   unsigned params = data->FuncPtrs[index].params;
   EAT(params, opcode);
   stack[stacktop-1].data->SetFuncNo(index);
   break;
}
case cPCall:
{
   unsigned index = ByteCode[++IP];
   unsigned params =
      data->FuncParsers[index]->data->varAmount;
   EAT(params, opcode);
   stack[stacktop-1].data->SetFuncNo(index);
   break;
}
```

```
            // Converted to cMul on fly
            case cDeg:
               ADDCONST(CONSTANT_DR);
               EAT(2, cMul);
               break;

            // Converted to cMul on fly
            case cRad:
               ADDCONST(CONSTANT_RD);
               EAT(2, cMul);
               break;

            // Functions
            default:
            {
               //assert(opcode >= cAbs);
               unsigned funcno = opcode-cAbs;
               assert(funcno < sizeof(Functions)/sizeof(Functions[0]));
               const FuncDefinition& func = Functions[funcno];

               //const FuncDefinition& func = Functions[opcode-cAbs];

               unsigned paramcount = func.params;
#ifndef DISABLE_EVAL
               if(opcode == cEval) paramcount = data->varAmount;
#endif
               if(opcode == cSqrt)
               {
                  // Converted on fly: sqrt(x) = x^0.5
                  opcode = cPow;
                  paramcount = 2;
                  ADDCONST(0.5);
```

```
}
if(opcode == cExp)
{
   // Converted on fly: exp(x) = CONSTANT_E^x

   opcode = cPow;
   paramcount = 2;
   // reverse the parameters... kludgey
   stack[stacktop] = stack[stacktop-1];
   stack[stacktop-1].SetImmed(CONSTANT_E);
   GROW(1);
}
bool do_inv = false;
if(opcode == cCot) { do_inv = true; opcode = cTan; }
if(opcode == cCsc) { do_inv = true; opcode = cSin; }
if(opcode == cSec) { do_inv = true; opcode = cCos; }

bool do_log10 = false;
if(opcode == cLog10)
{
   // Converted on fly: log10(x) = log(x) * CONSTANT_L10I
   opcode = cLog;
   do_log10 = true;
}
EAT(paramcount, opcode);
if(do_log10)
{
   ADDCONST(CONSTANT_L10I);
   EAT(2, cMul);
}
if(do_inv)
```

```
            {
               // Unary cMul, inverted. No need for "1.0"
               EAT(1, cMul);
               stack[stacktop-1].getp0().Invert();
            }
            break;
         }
      }
   }
   else
   {
      stack[stacktop].SetVar(opcode);
      GROW(1);
   }
}

if(!stacktop)
{
   // ERROR: Stack does not have any values!
   return;
}

--stacktop; // Ignore the last element, it is always nop (cAdd).

if(stacktop > 0)
{
   // ERROR: Stack has too many values!
   return;
}

// Okay, the tree is now stack[0]
*result = stack[0];
```

```
}

void FunctionParser::Optimize()
{
   copyOnWrite();

   CodeTree tree;
   MakeTree(&tree);

   // Do all sorts of optimizations
   tree.Optimize();
   // Last changes before assembly
   tree.FinalOptimize();

   // Now rebuild from the tree.

   vector<unsigned> byteCode;
   vector<double> immed;

#if 0
   byteCode.resize(Comp.ByteCodeSize);
   for(unsigned a=0; a<Comp.ByteCodeSize; ++a)byteCode[a] = Comp.ByteCode[a];

   immed.resize(Comp.ImmedSize);
   for(unsigned a=0; a<Comp.ImmedSize; ++a)immed[a] = Comp.Immed[a];
#else
   byteCode.clear(); immed.clear();
   tree.Assemble(byteCode, immed);
#endif

   delete[] data->ByteCode; data->ByteCode = 0;
   if((data->ByteCodeSize = byteCode.size()) > 0)
   {
```

```
        data->ByteCode = new unsigned[data->ByteCodeSize];
        for(unsigned a=0; a<byteCode.size(); ++a)
            data->ByteCode[a] = byteCode[a];
    }

    delete[] data->Immed; data->Immed = 0;
    if((data->ImmedSize = immed.size()) > 0)
    {
        data->Immed = new double[data->ImmedSize];
        for(unsigned a=0; a<immed.size(); ++a)
            data->Immed[a] = immed[a];
    }
}

#endif // #ifdef SUPPORT_OPTIMIZER
```

mainwindow.cpp

```
#include "mainwindow.h"

#include <setdialogvariables.h>
#include <outputmethoddialog.h>
#include <QMessageBox>
#include <QFont>
#include <iomanip>

int numberGenerations;
int numberChromosomes;
double selectionRate;
```

```
double mutationRate;
int wrapping;
int genomeSize;
char *myTrainFile;
char *myTestFile;
char output_method[100];
int random_seed;
int foldcount = 0;
int pattern_dimension;
vector<int> genome;

MainWindow::MainWindow (QWidget *parent):QMainWindow(parent)
{
   setWindowTitle("~Graduate Project~");
   mainWidget = new QWidget;
   mainWidget->setStyleSheet("background-color:black;");
   setCentralWidget(mainWidget);
   mainLayout = new QVBoxLayout;
   mainWidget->setLayout(mainLayout);
   createButtons();
   panel = new QStackedLayout;
   mainLayout->addLayout(panel);
   displayInstructions();
   displayResult();
}

void MainWindow::createButtons()
{
   buttonLayout = new QHBoxLayout;
   mainLayout->addLayout(buttonLayout);
```

```
    run = new QPushButton;
    run->setText("Run");
    run->setStyleSheet("background-color:rgb(100,100,100); color:white; outline: none;");
    buttonLayout->insertWidget(1, run, 0, Qt::AlignLeft);
    buttonLayout->insertWidget(1, run, 0, Qt::AlignTop);
    connect(run,SIGNAL(clicked(bool)),this,SLOT(runButtonSlot()));

    sets = new QPushButton;
    sets->setText("Set Variables");
    sets->setStyleSheet("background-color:rgb(100,100,100); color: white; outline:
none;");
    buttonLayout->insertWidget(2,sets,0,Qt::AlignLeft);
    buttonLayout->insertWidget(2,sets,0,Qt::AlignTop);
    connect(sets,SIGNAL(clicked(bool)),this,SLOT(variablesFormSlot()));

    importTrain = new QPushButton;
    importTrain->setText("Import Train File");
    importTrain->setStyleSheet("background-color:rgb(100,100,100); color: white;
outline: none;");
    buttonLayout->insertWidget(3, importTrain, 0, Qt::AlignLeft);
    buttonLayout->insertWidget(3, importTrain, 0, Qt::AlignTop);
    connect(importTrain,SIGNAL(clicked(bool)),this,SLOT(importTrainSlot()));

    importTest = new QPushButton;
    importTest->setText("Import Test File");
    importTest->setStyleSheet("background-color:rgb(100,100,100); color: white; outline:
none;");
    buttonLayout->insertWidget(4, importTest, 0, Qt::AlignLeft);
    buttonLayout->insertWidget(4, importTest, 0, Qt::AlignTop);
    connect(importTest,SIGNAL(clicked(bool)),this,SLOT(importTestSlot()));

    clear = new QPushButton;
```

```
    clear->setText("Clear");
    clear->setStyleSheet("background-color:rgb(100,100,100); color: white; outline:
none;");
    buttonLayout->insertWidget(5, clear, 0, Qt::AlignLeft);
    buttonLayout->insertWidget(5, clear, 0, Qt::AlignTop);
    connect(clear,SIGNAL(clicked(bool)),this,SLOT(clearButtonSlot()));

    exit = new QPushButton;
    exit->setText("Exit");
    exit->setStyleSheet("background-color:rgb(100,100,100); color: white;");
    buttonLayout->insertWidget(7, exit, 0, Qt::AlignLeft);
    buttonLayout->insertWidget(7, exit, 0, Qt::AlignTop);
    connect(exit,SIGNAL(clicked(bool)),this,SLOT(close()));

}

void MainWindow::variablesFormSlot()
{
    setDialogVariables *d = new setDialogVariables(this);
    int result = d->exec();
    if (result == QDialog::Accepted)
    {
        numberGenerations = d->getNumberGenerations();
        numberChromosomes = d->getNumberChromosomes();
        selectionRate = d->getSelectionRate();
        mutationRate = d->getMutationRate();
        wrapping = d->getWrappingEvent();
        genomeSize = d->getGenomeSize();

        if (numberGenerations == NULL || numberChromosomes == NULL || genomeSize
== NULL ||
                selectionRate == NULL || mutationRate == NULL || wrapping == NULL)
```

```
        {
            QMessageBox::warning(this,"Error","Please fill in all fields");
            variablesFormSlot();
        }

        flagVariables = 1;
    }

}

void MainWindow::runButtonSlot()
{
    if(flagVariables == 0 && flagTrainFile == 0 && flagTestFile == 0)
    {
        QMessageBox::warning(this,"Warning","Application can't run, not inputs");
    }
    else if (flagVariables == 0)
    {
      QMessageBox::warning(this,"Warning","Please insert values");
    }
    else if(flagTrainFile == 0)
    {
      QMessageBox::warning(this,"Warning","Please insert train file");
    }
    else if(flagTestFile == 0)
    {
        QMessageBox::warning(this,"Warning","Please insert test file");
    }
    else if(flagVariables == 1 && flagTrainFile == 1 && flagTestFile == 1)
    {
        clearResults();
```

```
        outputMethodSlot();
        panel->setCurrentIndex(1);

        initProgram();
        genome.resize(pattern_dimension * genomeSize);

        if(foldcount)
          foldValidation();
        else
          runOneFold();

        doneProgram();

        flagClear++;
    }
}

void MainWindow::clearButtonSlot()
{
    panel->setCurrentIndex(0);
    if (flagClear == 1) clearResults();
}

void MainWindow::clearResults()
{
    messageArea->clear();
}

void MainWindow::importTrainSlot()
{
    QString filename = QFileDialog::getOpenFileName(this,
                            "Open Train File", ".", "Txt file(*.txt *csv)" );
    if (filename.size() != 0)
```

```
    {
        QFile file(filename);
        if (!file.open(QIODevice::ReadOnly | QIODevice::Text))
            return;

        myTrainFile=(char *)malloc(1024);

        QByteArray ba=filename.toLatin1();
        strcpy(myTrainFile,ba);
    }
    flagTrainFile = 1;
}

void MainWindow::importTestSlot()
{
    QString filename = QFileDialog::getOpenFileName(this,
                            "Open Test File", ".", "Txt file(*.txt *csv)" );
    if (filename.size() != 0)
    {
        QFile file(filename);
        if (!file.open(QIODevice::ReadOnly | QIODevice::Text))
            return;

        myTestFile=(char *)malloc(1024);

        QByteArray ba=filename.toLatin1();
        strcpy(myTestFile,ba);
    }
    flagTestFile = 1;
}

void MainWindow::displayInstructions()
```

```
{
   QWidget *widgetIn = new QWidget;
   panel->addWidget(widgetIn);

   QHBoxLayout *instructionsLayout = new QHBoxLayout;
      widgetIn->setLayout(instructionsLayout);

      instructions = new QLabel;
      instructions-
>setText("<html><body><center><u>~Instructions~</u></center></body></html>"
                  "<html><body><center> </center></body></html>"
                  "<html><body><center>1. Set your
Variables</center></body></html>"
                  "<html><body><center>2. Train algorithm </center></body></html>"
                  "<html><body><center>3. Import data for
classification</center></body></html>"
                  "<html><body><center>4. Have fun</center></body></html>");
      QFont font = instructions->font();
      font.setPointSize(32);
      font.setBold(true);
      instructions->setFont(font);
      instructionsLayout->addWidget(instructions);
}

void MainWindow::displayResult()
{
   widgetRe = new QWidget;
   panel->addWidget(widgetRe);

   resultLayout = new QVBoxLayout;
   widgetRe->setLayout(resultLayout);
```

```
    messageArea = new QTextEdit();
    messageArea->setReadOnly(true);
}

void MainWindow::outputMethodSlot()
{
    outputMethodDialog *d = new outputMethodDialog(this);
    int result = d->exec();
    if (result == QDialog::Accepted)
    {
        choice = d->getChoice();
        if (choice==0)
            strcpy(output_method,SIMPLE_METHOD);
        else if (choice==1)
            strcpy(output_method,CSV_METHOD);
        else
            strcpy(output_method,FULL_METHOD);
    }
}

int MainWindow::myabs(int x)
{
    return x<0?-x:x;
}

void MainWindow::printConfusionMatrix()
{
    QString s1,s2;
    int i,j;
    vector<double> T;
    vector<double> O;
```

```
    p->getOutputs(T,O);
    int N=T.size();
    int nclass=p->getClass();
    int **CM;
    s1 = "** CONFUSION MATRIX ** Number of classes:
"+QString::number(nclass)+"\n";
    CM=new int*[nclass];
    for(i=0;i<nclass;i++) CM[i]=new int[nclass];
    for(i=0;i<nclass;i++)
       for(j=0;j<nclass;j++) CM[i][j] = 0;

    for(i=0;i<N;i++) CM[(int)T[i]][(int)O[i]]++;
    for(i=0;i<nclass;i++)
    {
       for(j=0;j<nclass;j++)
       {
         s2+=QString::number(CM[i][j])+" "+" ";
          if (j==nclass-1) s2+="\n";
       }
       printf("\n");
       delete[] CM[i];
    }
    delete[] CM;

    messageArea->setText(messageArea->document()->toPlainText()+"\n"+s1+s2+"\n");
    resultLayout->addWidget(messageArea);
}

void MainWindow::initProgram()
{
   //parse_cmd_line(argc,argv);
```

```
  srand(random_seed);
  p=new ClassProgram(myTrainFile);
  pattern_dimension = p->getClass()-1;
  pop=new Population(numberChromosomes,genomeSize * pattern_dimension,p);
  pop->setSelectionRate(selectionRate);
  pop->setMutationRate(mutationRate);
}

void MainWindow::doneProgram()
{
   delete p;
   delete pop;
}

void MainWindow::runGenetic()
{
   QString s1,s2,s3,s4;
   int i;
   for(i=1;i<numberGenerations;i++)
    {
       pop->nextGeneration();
       genome=pop->getBestGenome();
       s=p->printF(genome);
       if(!strcmp(output_method,FULL_METHOD))
          ok_to_print=1;
       pop->evaluateBestFitness();
       double bestf=pop->getBestFitness();
       ok_to_print=0;
       if(fabs(bestf)<1e-7) break;

       if(strcmp(output_method,CSV_METHOD)==0)
```

```
        {
           if(strlen(myTestFile)==0)
           {
             s1 = QString::number(i);
             s2 = QString::number(-pop->getBestFitness(),'f',2);
             messageArea->setText(messageArea->document()-
>toPlainText()+"\n"+s1+s2);
             resultLayout->addWidget(messageArea);

           }
           else
           {
              s1 = QString::number(i);
              s2 = QString::number(-pop->getBestFitness(), 'f',2);
              s3 = QString::number(-p->getClassError(genome,myTestFile),'f',2);
              messageArea->setText(messageArea->document()-
>toPlainText()+"\n"+s1+"."+" "+" "+s2+" "+" "+" "+s3+"\n");
              resultLayout->addWidget(messageArea);
           }
        }

        if(!strcmp(output_method,FULL_METHOD))
        {
           if(strlen(myTestFile)==0)
           {
              s1 = QString::number(i)+" ";
              s2 = QString::number(-pop->getBestFitness(),'f',2);
              s3 = s.c_str();
              messageArea->setText(messageArea->document()-
>toPlainText()+"\n"+"GENERATION = "+s1+
                                   "FITNESS = "+s2+"%"+" "+"\n"+"PROGRAMS = "+s3);
```

```
            resultLayout->addWidget(messageArea);

        }

        else
        {
            s1 = QString::number(i)+" ";
            s2 = QString::number(-pop->getBestFitness(),'f',2);
            s3 = QString::number(-p->getClassError(genome,myTestFile),'f',2);
            s4 = s.c_str();
            messageArea->setText(messageArea->document()-
>toPlainText()+"\n"+"GENERATION = "+s1+
                                 "FITNESS = "+s2+"%"+" "+"TEST_ERROR =
"+s3+"%"+"\n"+"PROGRAMS = "+s4);
            resultLayout->addWidget(messageArea);
        }
      }
    }

}

void MainWindow::foldValidation()
{
   QString s1,s2,s3,s4,s5,s6,s7,s8,s9,s10,s11;
   int i,j;
   srand(random_seed);
   vector<Data> trainx,testx;
   vector<Data> totalx;
   Data  totaly;
   Data trainy,testy;
   vector<int> Index;
   p->getTrainData(totalx,totaly);
```

```
Index.resize(totalx.size());
for(i=0;i<Index.size();i++) Index[i]=i;
for(i=0;i<Index.size();i++)
{
   int randpos=myabs(rand() % Index.size());
   int t=Index[i];
   Index[i]=Index[randpos];
   Index[randpos]=t;
}
int testsize=Index.size() / foldcount;
int trainsize=Index.size() - testsize;
trainx.resize(trainsize);
trainy.resize(trainsize);
testx.resize(testsize);
testy.resize(testsize);
for(i=0;i<trainsize;i++) trainx[i].resize(totalx[0].size());
for(i=0;i<testsize;i++) testx[i].resize(totalx[0].size());
double avg_train_error=0.0;
double avg_test_error=0.0;
int nclass=p->getClass();
vector<double> T;
vector<double> O;
int **CM;
CM=new int*[nclass];
for(i=0;i<nclass;i++) CM[i]=new int[nclass];
for(i=0;i<nclass;i++)
   for(j=0;j<nclass;j++) CM[i][j] = 0;
for(i=0;i<foldcount;i++)
{
   pop->reset();
```

```
s1 = "FOLD COUNT: \n";
s2 = QString::number(i);
messageArea->setText(messageArea->document()->toPlainText()+"\n"+s1+s2);
resultLayout->addWidget(messageArea);

int icount=0;
for(j=i*testsize;j<(i+1)*testsize;j++)
{
   testx[icount]=totalx[j];
   testy[icount]=totaly[j];
   icount++;
}
icount=0;
for(j=0;j<totalx.size();j++)
{
   if(icount>=trainsize) break;
   if(j>=i*testsize && j<(i+1)*testsize) continue;
   trainx[icount]=totalx[j];
   trainy[icount]=totaly[j];
   icount++;
}
p->setTrainData(trainx,trainy);
runGenetic();
p->fitness(genome);
s=p->printF(genome);
avg_train_error+=fabs(pop->getBestFitness());
avg_test_error+=fabs(p->getClassError(genome,testx,testy));
p->getOutputs(T,O);
for(j=0;j<T.size();j++) CM[(int)T[j]][(int)O[j]]+=1;
s3 = "EXPRESSION = ";
s4 = s.c_str();
```

```
        messageArea->setText(messageArea->document()->toPlainText()+"\n"+s3+s4);
        resultLayout->addWidget(messageArea);
    }

    s5 = "AVERAGE OUTPUT\n";
    s6 = "AVERAGE TRAIN ERROR=\n";
    s7 = QString::number(avg_train_error/foldcount);
    s8 = "AVERAGE TEST  ERROR=\n";
    s9 = QString::number(avg_test_error/foldcount);
    s10 = "** CONFUSION MATRIX ** Number of classes: \n";
    s11 = QString::number(nclass);

    for(i=0;i<nclass;i++)
    {
        for(j=0;j<nclass;j++)
        {
            s2+=QString::number(CM[i][j])+" "+" ";
            if (j==nclass-1) s2+="\n";
        }

        delete[] CM[i];
    }
    delete[] CM;
}

void MainWindow::runOneFold()
{
    QString s1,s2,s3,s4,s5,s6,s7;
    runGenetic();
    p->fitness(genome);
    s=p->printF(genome);
    s1 = "FINAL OUTPUT";
```

```
   s2 = " EXPRESSION=\n";
   s3 = s.c_str();
   s4 = "TRAIN ERROR = ";
   s5= QString::number(fabs(pop->getBestFitness()), 'f', 2);
   if(strlen(myTestFile)!=0)
   {
      s6 = "CLASS ERROR = ";
      s7 = QString::number(-p->getClassError(genome,myTestFile), 'f',2);
      printConfusionMatrix();
   }

   messageArea->setText(messageArea->document()-
>toPlainText()+"\n"+s1+s2+s3+s4+s5+"%"+"\n"+s6+s7+"%");
   resultLayout->addWidget(messageArea);

}
```

outputmethoddialog.cpp

```
#include <outputmethoddialog.h>

outputMethodDialog::outputMethodDialog(QWidget *parent)
   :QDialog(parent)
{
   setFixedSize(250,150);
   setWindowTitle("Choose Output Method");

   mainWidget = new QWidget(this);
   mainWidget->setFixedSize(this->width(),this->height());
```

```
    mainLayout = new QVBoxLayout;
    mainWidget->setLayout(mainLayout);

    group = new QButtonGroup;

    simpleButton = new QRadioButton;
    simpleButton->setText("Simple Method");
    group->addButton(simpleButton);
    mainLayout->addWidget(simpleButton);

    csvButton = new QRadioButton;
    csvButton->setText("Csv Method");
    group->addButton(csvButton);
    mainLayout->addWidget(csvButton);

    fullButton = new QRadioButton;
    fullButton->setText("Full Method");
    group->addButton(fullButton);
    mainLayout->addWidget(fullButton);

    next = new QPushButton;
    next->setText("Next");
    mainLayout->addWidget(next);

    connect(csvButton,SIGNAL(toggled(bool)),this,SLOT(csvSlot(bool)));
    connect(fullButton,SIGNAL(toggled(bool)),this,SLOT(fullSlot(bool)));
    connect(simpleButton,SIGNAL(toggled(bool)),this,SLOT(simpleSlot(bool)));
    connect(next,SIGNAL(clicked(bool)),this,SLOT(accept()));
}

void outputMethodDialog::simpleSlot(bool selected){if (selected) choice =0;}

void outputMethodDialog::csvSlot(bool selected){ if(selected) choice = 1; }
```

```
void outputMethodDialog::fullSlot(bool selected){ if (selected) choice = 2; }

int outputMethodDialog::getChoice(){ return choice; }
```

population.cpp

```
# include <population.h>
# include <stdlib.h>
# include <string.h>
# include <math.h>
# include <iostream>

# define MAX_RULE       65536

extern double selectionRate;
extern double mutationRate;

/* Population constructor */
/* Input: genome count , genome size, pointer to Program instance */
Population::Population(int gcount,int gsize,Program *p)
{
   elitism=1;
   selection_rate = selectionRate;
   mutation_rate  = mutationRate;
   genome_count   = gcount;
   genome_size    = gsize;
   generation     = 0;
```

```
   program        = p;

   /* Create the population and based on genome count and size */
   /* Initialize the genomes to random */
   double f;
   genome=new int*[genome_count];
   children=new int*[genome_count];
   for(int i=0;i<genome_count;i++)
   {
      genome[i]=new int[genome_size];
      children[i]=new int[genome_size];
         for(int j=0;j<genome_size;j++)
            genome[i][j]=rand()%MAX_RULE;
   }
   fitness_array=new double[genome_count];
}

/* Reinitialize the population to random */
void    Population::reset()
{
   generation = 0;
   for(int i=0;i<genome_count;i++)
      for(int j=0;j<genome_size;j++)
            genome[i][j]=rand()%MAX_RULE;
   for(int i=0;i<genome_count;i++)
         fitness_array[i]=-1e+8;
}

/* Return the fitness of a genome */
double  Population::fitness(vector<int> &g)
{
```

```
   return program->fitness(g);
}

/* The selection of the chromosomes according to the fitness value is performed */
void    Population::select()
{
   int itemp[genome_size];
   for(int i=0;i<genome_count;i++)
   {
      for(int j=0;j<genome_count-1;j++)
      {
         if(fitness_array[j+1]>fitness_array[j])
         {
            double dtemp;
            dtemp=fitness_array[j];
            fitness_array[j]=fitness_array[j+1];
            fitness_array[j+1]=dtemp;

            memcpy(itemp,genome[j],genome_size*sizeof(int));
            memcpy(genome[j],genome[j+1],genome_size*sizeof(int));
            memcpy(genome[j+1],itemp,genome_size*sizeof(int));
         }
      }
   }
}

int     Population::tournament()
{
   double max_fitness=-1e+10;
   const int tournament_size =(genome_count<=100)?4:20;
   int    max_index=-1;
```

```
   int r;
// Select the best parents of  the candidates
   for(int j=0;j<tournament_size;j++)
   {
   r=rand() % (genome_count);
      if(j==0 || fitness_array[r]>max_fitness)
      {
         max_index=r;
         max_fitness=fitness_array[r];
       }
    }
   return max_index;
}

/* Crossover operation: based on tournament selection */
/* Select the tournament_size based on the genome count : */
/*     (if genome_count > 100 ) tournament_size = 10   else   tournament_size = 4 */
/* Select 2 chromosomes based on the tournament size and cross them over based on the
crossover probability */
/* There is 1 crossover point and it is random */
void    Population::crossover()
{
      int parent[2];
      int nchildren=(int)((1.0 - selection_rate) * genome_count);
   if(!(nchildren%2==0)) nchildren++;
      int count_children=0;
      while(1)
      {
      // The two parents are selected here according to the tournament selection procedure
            for(int i=0;i<2;i++)
            {
```

```
                parent[i]=tournament();
            }
        int pt1,pt2;
        // The one-point crossover is performed here (the point is pt1)
        pt1=rand() % genome_size;
        memcpy(children[count_children],
            genome[parent[0]],pt1 * sizeof(int));
        memcpy(&children[count_children][pt1],
           &genome[parent[1]][pt1],(genome_size-pt1)*sizeof(int));
        memcpy(children[count_children+1],
            genome[parent[1]],pt1 * sizeof(int));
        memcpy(&children[count_children+1][pt1],
           &genome[parent[0]][pt1],(genome_size-pt1)*sizeof(int));
        count_children+=2;
        if(count_children>=nchildren) break;
    }

    for(int i=0;i<nchildren;i++)
    {
        memcpy(genome[genome_count-i-1],
            children[i],genome_size * sizeof(int));
    }
}

void    Population::setElitism(int s)
{
    elitism = s;
}

/* Mutate the current population */
/* Standard mutation algorithm: mutate all chromosomes in the population based on the
```

```
mutation probability */
void    Population::mutate()
{
   int start = 1;
   for(int i=start;i<genome_count;i++)
   {
      for(int j=0;j<genome_size;j++)
      {
         double r=rand()*1.0/RAND_MAX;
         if(r<mutation_rate)
            genome[i][j]=rand() %MAX_RULE;
      }
   }
}

/* Evaluate the fitness for all chromosomes in the current population */
void    Population::calcFitnessArray()
{
   vector<int> g;
   g.resize(genome_size);

   for(int i=0;i<genome_count;i++)
   {
      for(int j=0;j<genome_size;j++) g[j]=genome[i][j];
      fitness_array[i]=fitness(g);
   }
}

/* Return the current generation */
int     Population::getGeneration() const
{
```

```
   return generation;
}

/* Return the genome count */
int     Population::getCount() const
{
   return genome_count;
}

/* Return the size of the population */
int     Population::getSize() const
{
   return genome_size;
}

/* Evolve the next generation */
void    Population::nextGeneration()
{
   if(generation)
   mutate();
   calcFitnessArray();

   select();
   crossover();
   ++generation;
}

/* Set the mutation rate */
void    Population::setMutationRate(double r)
{
   if(r>=0 && r<=1) mutation_rate = r;
}
```

```
/* Set the selection rate */
void    Population::setSelectionRate(double r)
{
   if(r>=0 && r<=1) selection_rate = r;
}

/* Return the selection rate */
double  Population::getSelectionRate() const
{
   return selection_rate;
}

/* Return the mutation rate */
double  Population::getMutationRate() const
{
   return mutation_rate;
}

/* Return the best fitness for this population */
double  Population::getBestFitness() const
{
   return  fitness_array[0];
}

/* Return the best chromosome */
vector<int> Population::getBestGenome() const
{
   vector<int> g;g.resize(genome_size);
   for(int i=0;i<genome_size;i++) g[i]=genome[0][i];
   return g;
}
```

```
/* Evaluate and return the best fitness for all chromosomes in the population */
double  Population::evaluateBestFitness()
{
   vector<int> g;g.resize(genome_size);
   for(int i=0;i<genome_size;i++) g[i]=genome[0][i];
   return fitness(g);
}

/* Destructor */
Population::~Population()
{
   for(int i=0;i<genome_count;i++)
   {
      delete[] children[i];
      delete[] genome[i];
   }
   delete[] genome;
   delete[] children;
   delete[] fitness_array;
}
```

program.cpp

```
# include <program.h>

Program::Program()
{
   start_symbol = NULL;
```

```
}

void    Program::setStartSymbol(Symbol *s)
{
   start_symbol = s;
}

Symbol  *Program::getStartSymbol() const
{
   return start_symbol;
}

string  Program::printRandomProgram(vector<int> &genome,int &redo)
{
   string str="";
   int count=0;
   Rule *r;
   int pos=genome[count]%start_symbol->getCountRules();
   r=start_symbol->getRule(genome[count]%start_symbol->getCountRules());
   redo = 0;
   str=r->printRule(genome,count,redo);
   return str;
}

double  Program::fitness(vector<int> &genome)
{
   return 0.0;
}

Program::~Program()
{
}
```

rule.cpp

```
# include <rule.h>

# include <iostream>
Rule::Rule()
{
   length =0;
}

/* Adds a symbol to the stack */
void    Rule::addSymbol(Symbol *s)
{
   data.push_back(s);
   length++;
}

/* Return the position in the stack of a particular symbol */
int     Rule::getSymbolPos(string s)
{
   for(int i=0;i<length;i++)
      if(data[i]->getName()==s) return i;
   return -1;
}

/* Return the symbol from the stack given its position */
Symbol  *Rule::getSymbol(int pos) const
{
   if(pos<0 || pos>=length) return NULL;
```

```
    return data[pos];
}

/* Sets a symbol in a given position in the stack */
void    Rule::setSymbol(int pos,Symbol *s)
{
    if(pos<0 || pos>=length) return;
    data[pos]=s;
}

/* Returns the stack length */
int     Rule::getLength() const
{
    return length;
}

/* Prints the expression for this grammar given an array of integer (a chromosome) */
string  Rule::printRule(vector<int> genome,int &pos,int &redo)
{

    string str="";
    string str2="";
    Rule *r;
    int old_pos;
    extern int wrapping;
    for(int i=0;i<length;i++)
    {
        Symbol *s=data[i];
        if(s->getTerminalStatus())
        {
            str=str+s->getName();
        }
```

```
        else
        {
            if(pos>=genome.size()) {redo++;pos=0;}
            r=s->getRule((genome[pos]%s->getCountRules()));
            pos++;
            if(pos>=genome.size()) {redo++;pos=0;}
            if(redo>=wrapping) return str;
            str2=r->printRule(genome,pos,redo);
            str=str+str2;
        }
    }
    return str;
}

/* Evaluates and returns the function result given an instance of the functions (as
produced by the given chromosome), */
/* the given grammar and an input vector X */
double  Rule::getValue(vector<int> genome,int &pos,int &redo,DoubleStack
&stack,double *X)
{
    extern int wrapping;
    for(int i=0;i<length;i++)
    {
        Symbol *s=data[i];
        if(s->getTerminalStatus())
        {
            string str=s->getName();
            if(str==string("+"))
            {
                double a,b;
                a=stack.pop();
```

```
        b=stack.pop();
        cout<<"a ="<<a<<"b="<<b<<endl;
        stack.push(b+a);
    }
    else
    if(str=="-")
    {
        double a,b;
        a=stack.pop();
        b=stack.pop();
        stack.push(b-a);
    }
    else
    if(str=="*")
    {
        double a,b;
        a=stack.pop();
        b=stack.pop();
        stack.push(b*a);
    }
    else
    if(str=="x")
    {
        stack.push(X[0]);
    }
}
else
{
if(pos>=genome.size()) {redo++;pos=0;}
int k=genome[pos]%s->getCountRules();
```

```
        Rule *r;
        double rnd;
        r=s->getRule(genome[pos]%s->getCountRules());
        pos++;
        if(pos>=genome.size()) {redo++;pos=0;}
        if(redo>=wrapping) return 0;
        return (r->getValue(genome,pos,redo,stack,X));
        }
    }
    return stack.top();
}

Rule::~Rule()
{
}
```

setdialogvariables.cpp

```
#include <setdialogvariables.h>

#include <QLabel>

setDialogVariables::setDialogVariables(QWidget *parent)
    :QDialog(parent)
{

    setFixedSize(300,400);
    setWindowTitle("Set Variables");
```

```
mainWidget = new QWidget(this);
mainWidget->setFixedSize(this->width(),this->height());

mainLayout = new QVBoxLayout;
mainWidget->setLayout(mainLayout);

QLabel *label1, *label2, *label3, *label4, *label5, *label6;

label1 = new QLabel;
label1->setText("Number of Generations:");
mainLayout->addWidget(label1);
generationsLineEdit = new QLineEdit;
mainLayout->addWidget(generationsLineEdit);

label2 = new QLabel;
label2->setText("Number of Chromosomes:");
mainLayout->addWidget(label2);
chromosomesLineEdit = new QLineEdit;
mainLayout->addWidget(chromosomesLineEdit);

label3 = new QLabel;
label3->setText("Selection Rate:");
mainLayout->addWidget(label3);
selectionLineEdit = new QLineEdit;
mainLayout->addWidget(selectionLineEdit);

label4 = new QLabel;
label4->setText("Mutation Rate:");
mainLayout->addWidget(label4);
mutationLineEdit = new QLineEdit;
mainLayout->addWidget(mutationLineEdit);

label5 = new QLabel;
```

```
    label5->setText("Wrapping Events:");
    mainLayout->addWidget(label5);
    wrappingEdit = new QLineEdit;
    mainLayout->addWidget(wrappingEdit);

    label6 = new QLabel;
    label6->setText("Genome Size:");
    mainLayout->addWidget(label6);
    genomeSizeEdit = new QLineEdit;
    mainLayout->addWidget(genomeSizeEdit);

    buttonLayout = new QHBoxLayout;
    mainLayout->addLayout(buttonLayout);

    submit = new QPushButton;
    submit->setText("Submit");
    buttonLayout->addWidget(submit);

    cancel = new QPushButton;
    cancel->setText("Cancel");
    buttonLayout->addWidget(cancel);

    connect(submit,SIGNAL(clicked(bool)),this,SLOT(accept()));
    connect(cancel,SIGNAL(clicked(bool)),this,SLOT(reject()));
}

int setDialogVariables::getNumberGenerations() {return generationsLineEdit-
>text().toInt();}

int setDialogVariables::getNumberChromosomes() {return chromosomesLineEdit-
>text().toInt();}

double setDialogVariables::getSelectionRate() {return selectionLineEdit-
```

```
>text().toDouble();}

double setDialogVariables::getMutationRate() {return mutationLineEdit-
>text().toDouble();}

int setDialogVariables::getWrappingEvent() {return wrappingEdit->text().toInt();}

int setDialogVariables::getGenomeSize() {return genomeSizeEdit->text().toInt();}
```

symbol.cpp

```
# include <symbol.h>
Symbol::Symbol()
{
   name = "";
   count_rules = 0;
   is_terminal = 1;
}

/* Set a symbol (name,terminal status) */
void    Symbol::set(string s,int status)
{
   setName(s);
   setTerminalStatus(status);
}

/* Set the name of the symbol */
void    Symbol::setName(string s)
```

```
{
   name = s;
}

/* Get the name of the symbol */
string  Symbol::getName() const
{
   return name;
}

/* Set the terminal status of a symbol */
void    Symbol::setTerminalStatus(int status)
{
   is_terminal=status %  2;
}

/* Get the terminal status of a symbol */
int     Symbol::getTerminalStatus() const
{
   return is_terminal;
}

/* Add a new rule to the current rule stack */
void    Symbol::addRule(Rule *r)
{
   rule.push_back(r);
   count_rules++;
}

/* Get a rule for this symbol by position: return NULL if out of bounds */
Rule    *Symbol::getRule(int pos) const
{
```

```
   if(pos<0 || pos>=count_rules) return NULL;
   return rule[pos];
}

/* Count the rules for this symbol */
int     Symbol::getCountRules() const
{
   return count_rules;
}

/* Cut the rules to N */
void    Symbol::cut(int N)
{
   count_rules=N;
   rule.resize(N);
}

Symbol::~Symbol()
{
}
```

main.cpp

```
#include <QApplication>

#include "mainwindow.h"
#include <locale.h>

int ok_to_print=1;
```

```
int main(int argc, char *argv[])
{
    setlocale(LC_ALL,"C");

    QApplication a(argc, argv);

    MainWindow w;
    w.showNormal();

    return a.exec();

}
```

ΒΙΒΛΙΟΓΡΑΦΙΑ

(1) https://www.qt.io/ [πρόσβαση 16 Αυγούστου 2018].

(2) https://www.ics.com/blog/getting-started-qt-and-qt-creator-windows [πρόσβαση 16 Αυγουστου 2018].

(3) https://web.stanford.edu/dept/cs_edu/qt-creator/qt-creator.shtml [πρόσβαση 16 Αυγούστου 2018].

(4) https://www.lucidar.me/en/dev-c-cpp/how-to-install-qt-creator-on-ubuntu-16-04/ [πρόσβαση 16 Αυγούστου 2018].

(5) http://www.mingw.org/wiki/MinGW [πρόσβαση 07 Αυγούστου 2018].

(6) http://osp.mans.edu.eg/rehan/ann/Artificial%20Neural%20Networks.htm [πρόσβαση 29 Αυγούστου 2018].

(7) http://www.deeplearningbook.org/ [πρόσβαση 07 Οκτωβρίου 2018].

(8) https://eclass.duth.gr/modules/document/file.php/OPE02101/HliadisNeuralNetworks.pps [πρόσβαση 13 Οκτωβρίου 2018].

(9) http://ai.uom.gr/Courses/AdvancedNeuralNetworks/Material/NeuralNetworksAll.pdf [πρόσβαση 13 Οκτωβρίου 2018].

(10) https://aetos.it.teithe.gr/~kdiamant/MachineLearning/MachineLearningLesson02.pdf [πρόσβαση 13 Οκτωβρίου 2018].

(11) http://www.aaai.org/Papers/AAAI/1984/AAAI84-031.pdf [πρόσβαση 16 Οκτωβρίου 2018].

(12) https://medium.com/@sifium/machine-learning-types-of-classification-9497bd4f2e14 [πρόσβαση 16 Οκτωβρίου 2018].

(13) https://www.analyticsvidhya.com/blog/2015/08/comprehensive-guide-regression/ [πρόσβαση 22 Οκτωβρίου 2018].

(14) https://www.worldfullofdata.com/most-used-machine-learning-algorithms/ [πρόσβαση 22 Οκτωβρίου 2018].

(15) http://historyreport.gr/index.php/%CE%A0%CF%81%CF%8C%CF%83%CF%89%CF%80%CE%B1/1969-2012-01-02-16-53-43 [πρόσβαση 23 Οκτωβρίου 2018].

(16) http://users.cs.uoi.gr/~itsoulos/publications/nnc_neurocomputing.pdf [πρόσβαση 23 Οκτωβρίου 2018].

(17) http://apothetirio.teiep.gr/xmlui/bitstream/handle/123456789/109/tlp_000379a.pdf?sequence=2 [πρόσβαση 28 Οκτωβρίου 2018].

(18) http://myweb.teleinfom.teiep.gr/gtsoulos/public_html/genetic.php [πρόσβαση 28 Οκτωβρίου 2018].

(19) http://www.icsd.aegean.gr/lecturers/kavallieratou/NN&EP_files/ci_6.pdf [πρόσβαση 27 Οκτωβρίου 2018].

(20) http://myweb.teleinfom.teiep.gr/gtsoulos/public_html/publications/phd.pdf [πρόσβαση 27 Οκτωβρίου 2018].

(21) ΔΙΑΜΑΝΤΑΡΑΣ ΚΩΝΣΤΑΝΤΙΝΟΣ, (2007), «ΤΕΧΝΗΤΑ ΝΕΥΡΩΝΙΚΑ ΔΙΚΤΥΑ», ΚΛΕΙΔΑΡΙΘΜΟΣ

(22) http://www.e-conometrics.gr/2013/07/25/apli-palindromisi/ [πρόσβαση 2 Νοεμβρίου 2018].

(23) http://repfiles.kallipos.gr/html_books/93/04a-main.html [πρόσβαση 3 Νοεμβρίου 2018].

(24) https://github.com/itsoulos/GenClass [πρόσβαση 14 Αυγούστου 2018]