ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΗΛΕΚΤΡΟΝΙΚΩΝ
ΥΠΟΛΟΓΙΣΤΩΝ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ ΚΑΙ ΔΙΝΤΥΩΝ

Υλοποίηση αλγορίθμου KioustelidisQuadratic σε περιβάλλον iOS

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Πέτρος Μιτσίγιωργης

Βόλος, Μάρτιος 2011

Υλοποίηση αλγορίθμου KioustelidisQuadratic σε περιβάλλον iOS

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Πέτρος Μιτσίγιωργης

Επιβλέπων: Αλκιβιάδης Γ. Ακρίτας

Αναπλ. Καθηγητής Π.Θ.

(Υπογραφή)

……

Αλκιβιάδης Γ. Ακρίτας

Αναπληρωτής Καθηγητής Παν. Θεσσαλίας

(Υπογραφή)

……

Δασκαλοπούλου Ασπασία

Επίκουρη Καθηγήτρια Παν. Θεσσαλίας

Βόλος, Μάρτιος 2011

(Υπογραφή)

……

Πέτρος Μιτσίγιωργης

Διπλωματούχος Μηχανικός Ηλεκτρονικών Υπολογιστών, Τηλεπικοινωνιών και Δικτύων Πανεπιστημίου Θεσσαλίας

Στους γονείς μου

ΠΕΡΙΕΧΟΜΕΝΑ

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον επιβλέπων καθηγητή μου κ. Αλκιβιάδη Γ. Ακρίτα για την ευκαιρία που μου έδωσε να ασχοληθώ με ένα τόσο ενδιαφέρον και σύγχρονο θέμα καθώς και για την πολύτιμη βοήθεια και άριστη καθοδήγηση που μου είχε προσφέρει, κατά την εκπόνηση της παρούσας διπλωματικής εργασίας. Επίσης θα ήθελα να ευχαριστήσω θερμά και το δεύτερο μέλος της εξεταστικής επιτροπής, καθηγήτρια κα. Δασκαλοπούλου Ασπασία. Τέλος θα ήθελα να αναφέρω και να ευχαριστήσω τη συμφοιτήτρια και στενή μου φίλη Τριανταφυλλιά Σωτήρχου, χωρίς της οποίας την βοήθεια και συμπαράσταση αυτή η διπλωματική εργασία δε θα είχε ολοκληρωθεί.

Βόλος, Μάρτιος 2011 Μιτσίγιωργης Πέτρος

ΥΛΟΠΟΙΗΣΗ ΑΛΓΟΡΙΘΜΟΥ KIOUSTELIDIS QUADRATIC ΣΕ ΠΕΡΙΒΑΛΛΟΝ iOS

Κεφάλαιο 1 Εισαγωγικές Έννοιες

1.1 Εισαγωγή

Στη διπλωματική αυτή περιγράφεται το πως έφτασα να εξοικειωθώ με το περιβάλλον ανάπτυξης εφαρμογών iOS και πως ξεκίνησα να γράφω και να αναπτύσσω τη δική μου εφαρμογή στην οποία θα υλοποιούσα τον αλγόριθμο KioustelidisQuadratic.

Έχοντας τις γνώσεις προγραμματισμού που αποκόμισα από τις προπτυχιακές μου σπουδές σε γλώσσες C και Java ξεκίνησα να καταλάβω και να μάθω κάτι εντελώς καινούριο για μένα, το περιβάλλον ανάπτυξης εφαρμογών iOS. Αυτό που ίσως έκανε ακόμα πιο δύσκολο το έργο μου, ήταν η έλλειψη του απαραίτητου εξοπλισμού, όπως και η ενασχόληση με ένα καινούριο για μένα λειτουργικό, τα Mac OS X.

1.2 Συσκευές iOS

Καταρχάς να αναφέρουμε ότι όταν λέμε iOS μιλάμε για το λειτουργικό σύστημα που τρέχει σε όλες τις συσκευές iPod Touch, iPhone και iPad. Δηλώνοντας αυτό καταλαβαίνετε για το πόση απήχηση μπορεί να έχει μια εφαρμογή που θα έχει αναπτύχθει σε iOS όταν παγκόσμια έχουν πωληθεί πάνω από 160 εκατομμύρια τέτοιες συσκευές. Θεωρητικά εαν αναπτύξετε μια εφαρμογή iOS μπορεί να τρέξει σε όποιαδήποτε από αυτές.
Πέρα από τη μεγάλη διαθεσιμότητα των συσκευών αυτών αξίζει να αναφέρουμε και τις σημαντικές δυνατότητες που διαθέτουν. Όπως κάθε “έξυπνη” συσκευή σήμερα, έτσι και οι iΣυσκευές διαθέτουν ισχυρό επεξεργαστή αρχιτεκτονικής ARM, μονάδα 3D γραφικών, μια μεγάλη γκάμα συνδεσιμότητας όπως GSM, HSDPA, WiFi, Bluetooth, αισθητήρες GPS, Accelerometer, Gyro sensor, Proximity sensor [1]. Συγκεκριμένα στις συσκευές τελευταίας γενιάς υπάρχει το Apple A4 chip το οποίο περικλείει σε ένα μόνο chip ένα επεξεργαστή ARM Cortex A8 στα 1Ghz, μονάδα γραφικών PowerVR SGX535 και άλλες τεχνολογίες που το καθιστούν πιο ισχυρό και από ένα μηχάνημα desktop Pentium 3 με ας πούμε μια κάρτα γραφικών Intel GMA500.
Οπότε από άποψη hardware οι συσκευές iOS προσφέρουν μια πλατφόρμα για δημιουργία και ανάπτυξη μιας μεγάλης γκάμας εφαρμογών με πολλές δυνατότητες. Το μόνο που χρειάζεται, πέρα από την εξοικείωση με το περιβάλλον ανάπτυξης εφαρμογών iOS, είναι μια καλή ιδέα και δημιουργικότητα και προτού το καταλάβει κάποιος θα μπορεί να ανεβάσει στο Appstore τη πρώτη του εφαρμογή. Το Appstore είναι η πλατφόρμα που ανέπτυξε η Apple για τη συγκέντρωση και διαμοίραση όλων των εφαρμογών για iOS συσκευές. Στο Appstore μπορεί κάποιος να βρεί κάθε είδους εφαρμογής που μπορεί να είναι είτε δωρεάν είτε επί πληρωμής.

1.3 Τι χρειάστηκε

Από άποψη γνώσεων θα πρέπει κάποιος να ξέρει κάποια C-based γλώσσα (C, C#, C++) και να είναι εξοικειωμένος με όρους αντικειμενοστράφειας όπως κλάση, αντικείμενο, μέθοδος κτλ. Όπως ήδη ανάφερα ήξερα ήδη να χειρίζομαι αρκετά άνετα τη γλώσσα C και ήμουν αρκετά εξοικειωμένος με τη γλώσσα Java. Παρόλα αυτά επειδή αυτή ήταν η πρώτη μου επαφή με την Objective C (η γλώσσα προγραμματισμού σε iOS) ξεκίνησα από τα πιο βασικά και θα φροντίσω να αναπτύξω και να χτίσω σε αυτά που έμαθα περισσότερο στο μέλλον.
Από άποψη λογισμικού τώρα χρειάστηκα λειτουργικό Macintosh. Αν και η ανάπτυξη εφαρμογών είναι δυνατή και σε λειτουργικό Windows, παρόλα αυτά δεν είναι κάτι που συνιστάται. Στα Windows η ανάπτυξη εφαρμογών είναι δυνατή μόνο μέσω εφαρμογών τρίτων (Flash CS5, Airplay SDK) ή μηχανές παιχνιδιών (Unity 3D, Stonetrip S3D) ή web τεχνολογιών (HTML/CSS/Javascript). Με αυτές τις λύσεις θα είναι δυνατή μόνο η ανάπτυξη εφαρμογών χαμηλής απόδοσης και πολλές φορές δε θα είναι δυνατό το ανέβασμα της εφαρμογής στο Appstore.
Το λειτουργικό Macintosh έπρεπε να είναι έκδοσης Mac OS X 10.5 και πάνω, το οποίο να τρέχει σε κάποιο Intel-based υπολογιστή Apple. Η πιο οικονομική λύση που θα μπορούσε κάποιος να προτείνει είναι ο Apple Mac Mini ο οποίος αυτή τη χρονική περίοδο ξεκινάει στα 700 ευρώ. Το μόνο που λείπει από κει και πέρα είναι το λογισμικό iOS SDK το οποίο περιέχει όλα τα εργαλεία και documentation που θα χρειαστουν για να ξεκινήσει κάποιος. Κύριο εργαλείο της σουίτας iOS SDK είναι η εφαρμογή Xcode η οποία αποτελεί τη διεπαφή με όλα τα υπόλοιπα εργαλεία που θα χρειαστούν.
Για την προμήθευση του iOS SDK χρειάστηκε να εγγραφώ και να δημιουργήσω δωρεάν το Apple ID μου. Η δημιουργία του Apple ID είναι αναγκαία και για την ένταξή αργότερα σε κάποιο Developer Program, όπως το iOS Developer University Program εφόσον είναι διαθέσιμο σε κάποια σχολή και αφού προσκληθεί κάποιος από τον διδάσκοντα. Δυστυχώς κατά τη διάρκεια εκπόνησης της διπλωματικής μου κάτι τέτοιο δεν ήταν διαθέσιμο στη σχολή μου. Τη δημιουργία του Apple ID μου την έκανα εδώ και έπειτα κατέβασα το iOS SDK από εδώ [2].
Από τη στιγμή που ήταν διαθέσιμα όλα όσα ανέφερα ήμουν έτοιμος για να ξεκινήσω την ανάπτυξη της εφαρμογής μου. Όπως θα προσέξατε δεν έχω αναφέρει την ανάγκη για κάποια συσκευή iOS. Αυτό δεν είναι τυχαίο αφού η σουίτα iOS SDK συμπεριλαμβάνει και μια σειρά από προσομοιωτές (iPod Touch, iPhone, iPad) στους οποίους μπόρεσα να προσομοιώσω την εφαρμογή μου σε κάθε στάδιο της ανάπτυξής της.

1.4 Εναλλακτικές Υπολογιστή Mac

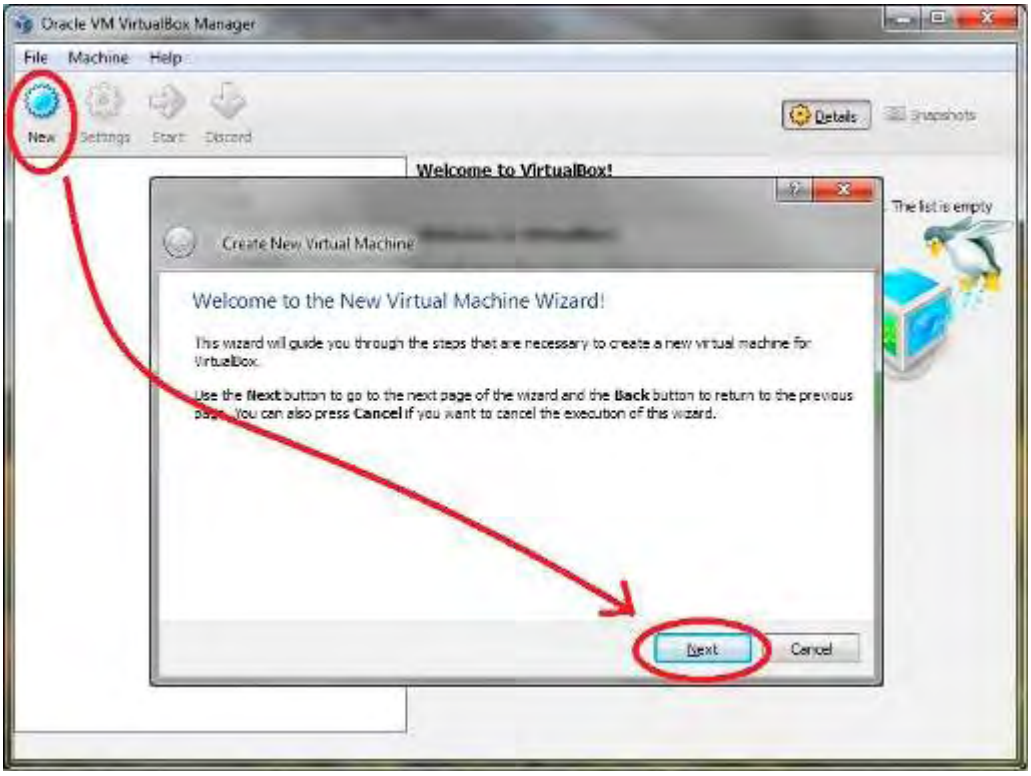
Όπως ανέφερα, κατά τη διάρκεια της εκπόνησης της διπλωματικής μου, υπήρξαν κάποιες δυσκολίες όσο αφορά τον εξοπλισμό που χρειάστηκα. Η κυριότερη δυσκολία ήταν η αδυναμία ύπαρξης υπολογιστή Mac. Δυστυχώς δεν υπήρχε η δυνατότητα για άμεση απόκτηση υπολογιστή Mac. Για το λόγο κατέφυγα σε μερικές άλλες εναλλακτικές ως προς την ύπαρξη υπολογιστή Mac. Δεν αποτελούν την ιδανικότερη λύση (το ιδανικότερο θα ήταν η αγορά ενός Mac υπολογιστή) αλλά σίγουρα αποτελούν καλύτερη εναλλακτική από την ανάπτυξη εφαρμογών iOS σε περιβάλλον Windows.

1.4.1 Εικονικό μηχάνημα Mac

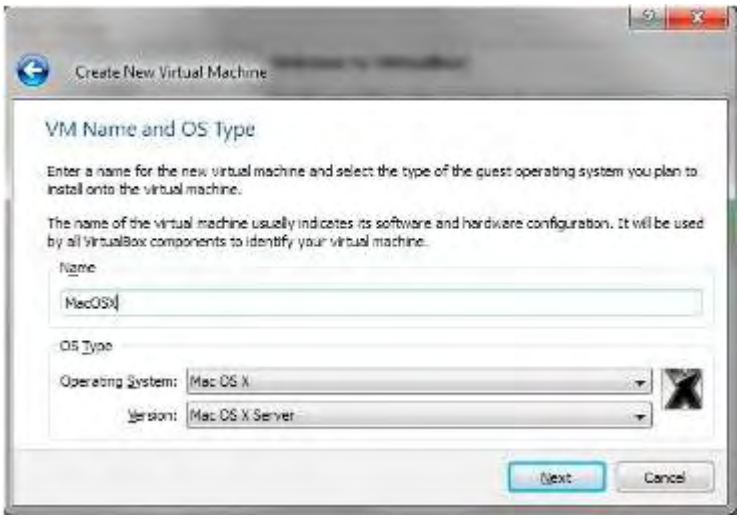
Ως πρώτη εναλλακτική θα αναφέρω το τρέξιμο Macintosh σε εικονικό μηχάνημα από υπολογιστή που τρέχει λειτουργικό Windows. Η λύση αυτή ήταν απόλυτα ασφαλής και θα μπορούσε να την ακολουθήσει σχετικά οποιοσδήποτε. Για τη συγκεκριμένη λύση χρειάστηκα μια εφαρμογή Εικονικής Μηχανής (το VirtualBox το οποίο είναι και δωρεάν) και ένα δισκάκι Mac OS X installation.

Ο μόνος περιορισμός που υπήρχε ήταν ότι ο υπολογιστής έπρεπε να διαθέτει ένα σχετικά σύγχρονο επεξεργαστή Intel. Συγκεκριμένα κάποιο που να ενσωματώνει τη τεχνολογία virtualization της Intel (VT-X). Εγώ φυσικά διέθετα υπολογιστή με επεξεργαστή AMD, με την αντίστοιχη τεχνολογία virtualization (AMD-V) και έτσι ξεκίνησα την απόπειρα για το στήσιμο του εικονικού μηχανήματος. Να αναφέρω ότι εν τέλει η απόπειρα αυτή στο συγκεκριμένο υπολογιστή με επεξεργαστή AMD δεν ήταν επιτυχής. Παρόλ' αυτά τη διαδικασία την εκτέλεσα σε άλλο υπολογιστή με Intel πεξεργαστή στον οποίο μπορούσα να έχω κατά καιρούς πρόσβαση.

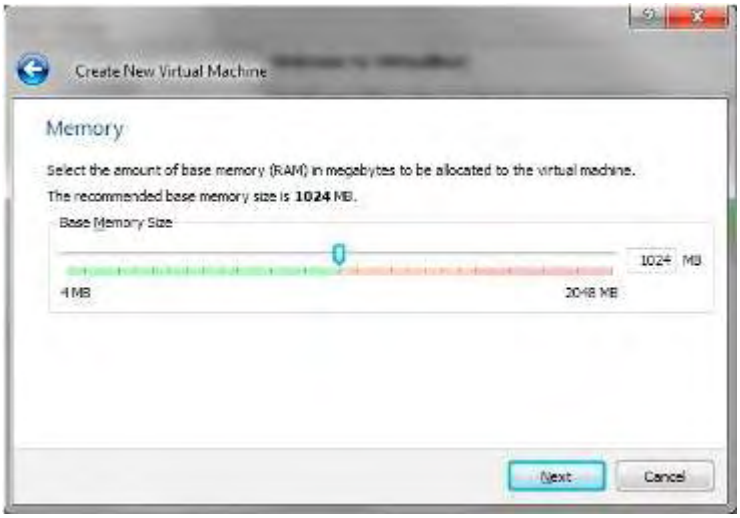
Να επαναλάβω ότι η απόπειρα έγινε για εγκατάσταση του λειτουργικού Mac OS X σε ένα εικονικό σύστημα. Για την ακρίβεια, σε ένα virtual machine το οποίο έφτιαξα με το VirtualBox. Τα βήματα που ακολούθησα είναι συγκεκριμένα για αυτό το εργαλειο (VirtualBox) το οποίο προμηθεύτηκα δωρεάν από εδώ. Αφού προμηθεύτηκα και το δισκάκι εγκατάστασης Mac OS X, δημιούργησα στο δίσκο του υπολογιστή ένα αρχείο ISO από το δισκάκι, με τη βοήθεια του δωρεάν ImgBurn. Ακολούθως ξεκίνησα τη διαδικασία για την εγκατάσταση του Snow Leopard σ' ένα VirtualBox VM, την οποία διατύπωσα σε μορφή βημάτων για να μπορεί να την ακολουθήσει και οποιοσδήποτε άλλος επιθυμεί. Αν λοιπόν ενδιαφέρεται κάποιος για την εξής διαδικασία ορίστε τα βήματα που πρέπει να ακολουθήσει:



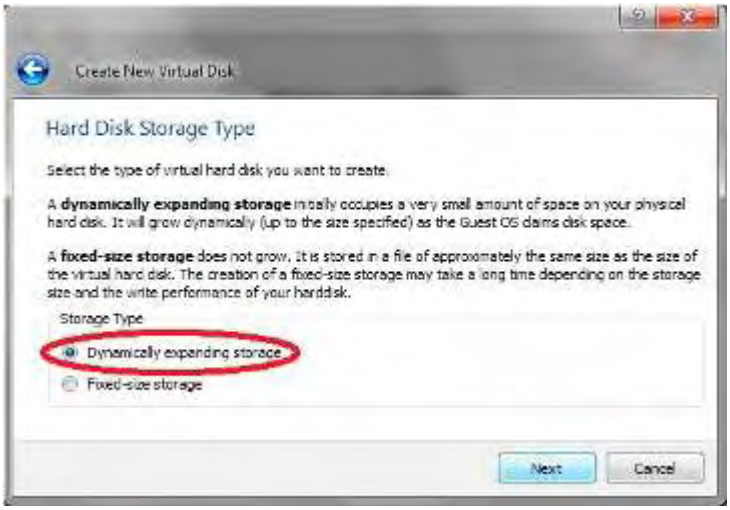
Αφού ξεκινήσετε το VirtualBox το πρώτο που έχετε να κάνετε είναι να δημιουργήσετε μια νέα εικονική μηχανή πατώντας το κουμπί New.



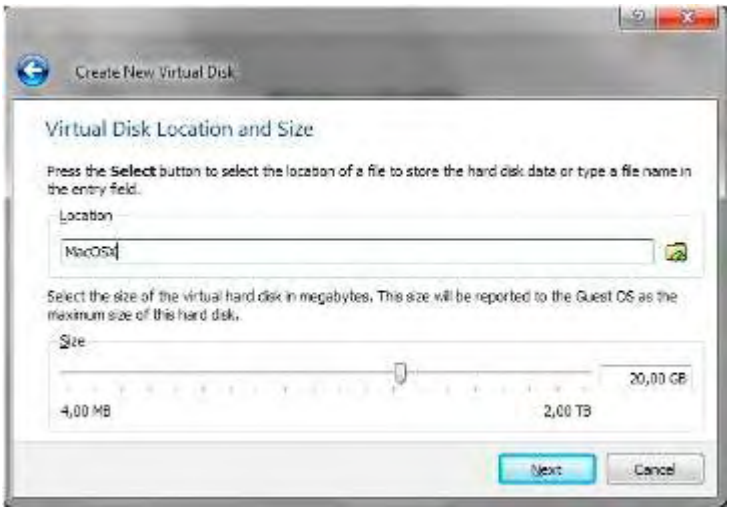
Ονομάστε τη όπως θέλετε, αλλά μη ξεχάσετε να επιλέξετε το Mac OS X ως το λειτουργικό που θα εγκαταστήσετε.



Ακολούθως, καθορίστε τη μνήμη του συστήματος που θα χρησιμοποιήσει η εικονική μηχανή. Φροντίστε να είναι τουλάχιστον 1GB.



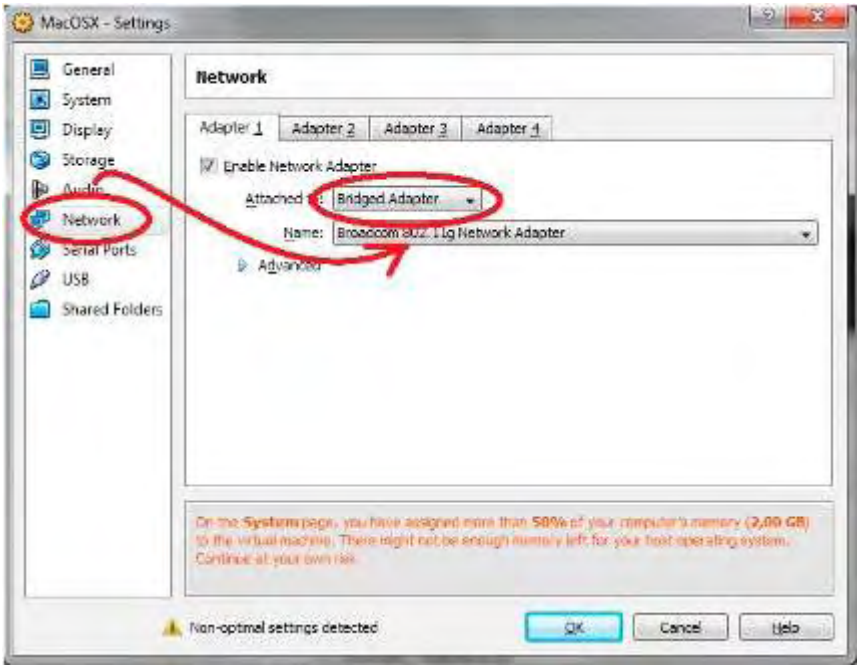
Για να ολοκληρωθεί η δημιουργία της εικονικής μηχανής απομένει η δημιουργία ενός εικονικού δίσκου. Με την επιλογή Boot Hard Disk να είναι ticked, θα επιλέξετε Create new hard disk και θα διαλέξετε την επιλογή Dynamically Expanding Storage. Αυτό επειδή ίσως να μην σας χρειαστεί άμεσα όλο αυτό το μέγεθος, γι αυτό το λόγο δεν είναι ανάγκη να δεσμεύσετε πραγματικό χώρο από το δίσκο του υπολογιστή σας αλλά να δεσμευτεί αργότερα δυναμικά.



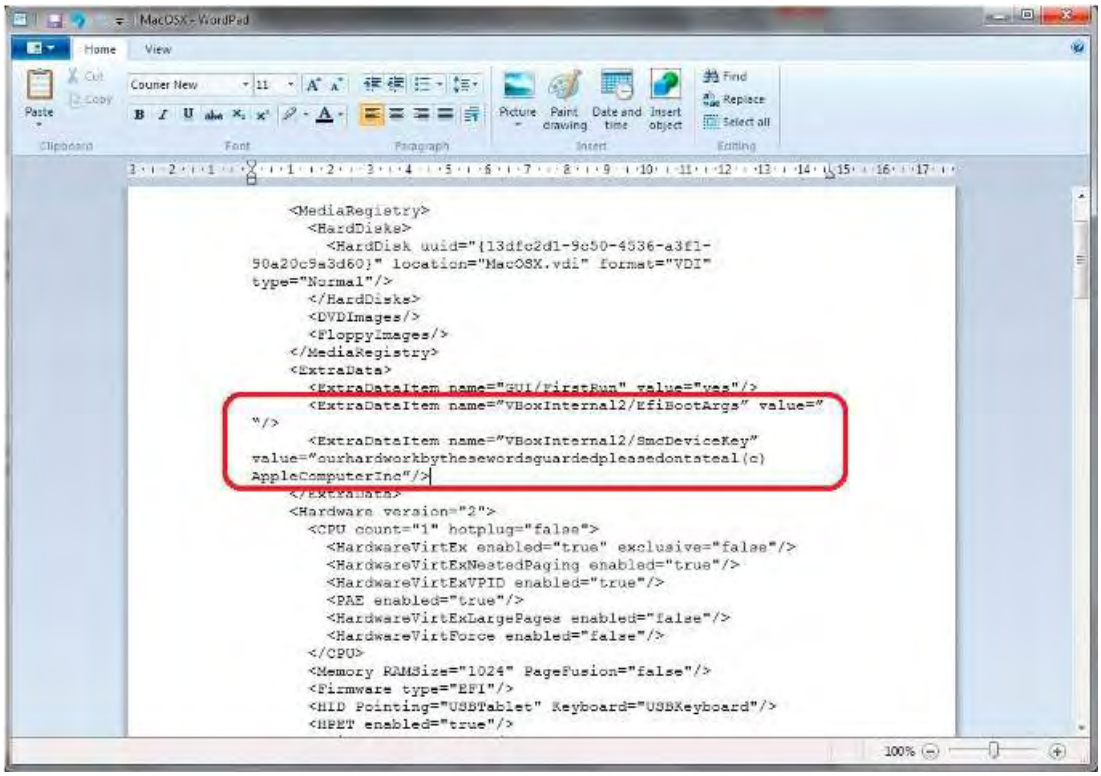
Στη συνέχεια καθορίστε το μέγιστο μέγεθος του δίσκου (20GB τουλάχιστον) και αποδώστε του κάποιο όνομα.
Σε αυτό το στάδιο το VirtualBox έχει δημιουργήσει τη νέα σας εικονική μηχανή. Πριν την ενεργοποιήσετε όμως, θα πρέπει να κάνετε μερικές ακόμα ρυθμίσεις.

Αφού είναι επιλεγμένη η εικονική συσκευή που μόλις δημιουργήσατε, ξεκινήστε πατώντας το κουμπί Settings. Στο παράθυρο που εμφανίζεται επιλέξτε τη περιοχή ρυθμίσεων Storage, επιλέξτε το δισκάκι με το όνομα Empty, κάντε κλικ στο εικονίδιο με το δισκάκι CD/DVD που είναι στα δεξιά και μετά πατήστε το κουμπί με το κίτρινο φακελάκι Choose a virtual CD/DVD disk file . Από το παράθυρο που θα προκύψει εντοπίστε και επιλέξτε το αρχείο ISO με το

Mac OS X. Τέλος πατήστε το κουμπί Open. Μ’ αυτό το τρόπο η εικονική μηχανή θα λειτουργεί σαν να της είχατε εισάγει ένα αληθινό δισκάκι με το Mac OS X!



Πριν κλείσετε το παράθυρο ρυθμίσεων, καλό θα ήταν να επισκεφθείτε και τη περιοχή Network. Εκεί επιλέξτε το Bridged Adapter, ώστε η εικόνική μηχανή να μοιράζεται την ίδια κάρτα δικτύου με το πραγματικό σύστημα. Έτσι η εικονική μηχανή θα έχει πρόσβαση σε όλα τα δίκτυα που συνδέεται ο υπολογιστής σας. Αργότερα, αυτό θα διευκολύνει την επικοινωνία της εικονικής μηχανής με το διαδίκτυο και την Apple για τη λήψη του απαραίτητου λογισμικου αλλά και επίσημων ενημερώσεων του λειτουργικού.
Απομένει μια λεπτομέρεια πριν ξεκινήσετε την εικονική μηχανή και την εγκατάσταση του Mac OS X. Πρόκειται για μια επέμβαση στο αρχείο ρυθμίσεων που διατηρεί το VirtualBox για την εικονική μηχανή.



Κλείστε το πρόγραμμα και μεταβείτε στη θέση:
τοπικός_δισκος:\Users\όνομα_χρήστη\VirtualBox VMs\όνομα_VM
Αν αντί Windows 7 χρησιμοποιείτε τα XP ή κάποια διανομή Linux, οι αντίστοιχες θέσεις είναι οι ακόλουθες:
τοπικός_δισκος:\Documents and Settings\όνομα_χρήστη\VirtualBox VMs\όνομα_VM
και
/home/όνομα_χρήστη/VirtualBox VMs/όνομα_VM/
Προφανώς στη θέση του τοπικού δίσκου θα πρέπει να βάλετε το γράμμα του δικού σας τοπικού δίσκου (συνήθως C:\), στη θέση του όνομα_χρήστη πρέπει να βάλετε το username σας, ενώ στη θέση του όνομα_VM το όνομα που δώσατε στη εικονική συσκευή σας.
Σε κάθε περίπτωση στη συγκεκριμένη θέση θα βρείτε 3 αρχεία. Με κάποιο απλό text editor θα ανοίξετε εκείνο που έχει το ίδιο ακριβώς όνομα μ' εκείνο της εικονικής μηχανής.
Μέσα στο αρχείο αναζητήστε τη περιοχή ExtraData. Όταν την εντοπίσετε, προσθέστε στο τέλος της τις παρακάτω γραμμές:

```
<ExtraDataItem name="VBoxInternal2/EfiBootArgs" value=" "/>
<ExtraDataItem name="VBoxInternal2/SmcDeviceKey"
value="ourhardworkbythesewordsguardedpleasedontsteal(c)AppleComputerInc"/>
```

Αμέσως μετά αποθηκεύστε τις αλλαγές, κλείστε το αρχείο και ξεκινήστε εκ νέου το VirtualBox.
Αυτή τη φορά μπορείτε επιτέλους να ενεργοποιήσετε την εικονική μηχανή πατώντας το κουμπί Start. Το αρχείο ISO που της είχατε προσαρτήσει θα φορτωθεί και θα ξεκινήσει η εγκατάσταση του Mac OS X.


Mac OS X
Use English for the main language
日本語を使用する
Utiliser le français comme langue principale
Deutsch als Standardsprache verwenden
Usar español como idioma principal
Usa l'italiano come lingua principale
Usar português do Brasil como idioma principal
Usar o português europeu como idioma principal
Gebruik Nederlands als hoofdtaal
Använd svenska som huvudspråk
Bruk norsk som hovedspråk
Brug dansk som hovedsprog
Käytä pääkielenä suomea
Использовать русский как главный язык
Użyj polskiego jako języka głównego


Mac OS X Installer File Edit Utilities Window
Install Mac OS X
To set up the installation of Snow Leopard, click Continue.
To repair your disk using Disk Utility, restore your computer from a Time Machine backup, or perform other tasks using Utilities, choose an option from the Utilities menu.
Continue

Επιλέξτε την επιθυμητή γλώσσα και πατήστε το βελάκι για να προχωρήσετε!

Σε λίγα δευτερόλεπτα θα φορτωθεί το πρόγραμμα που πραγματοποιεί την εγκατάσταση του λειτουργικού και στη πάνω πλευρά της οθόνης θα εμφανιστεί ένα μενού. Απ' αυτό πρέπει να ξεκινήσετε το Disk Utility. Βλέπετε ο δίσκος της εικονικής μηχανής δεν έχει καμιά κατάτμηση (partirion) ακόμα.

Όταν ξεκινήσει το προγραμματάκι επιλέξτε το (μοναδικό) δίσκο του εικονικού συστήματος και μεταβείτε στη καρτέλα Partition. Εκεί αρκεί να δώσετε ένα όνομα στη κατάτμηση που θέλετε να δημιουργηθεί και να πατήσετε στο κουμπί Partition.

Πλέον κάθε εξάρτημα της εικονικής μηχανής είναι έτοιμο να δεχτεί το Mac OS X. Κάντε ένα κλικ πάνω στο εικονίδιο του δίσκου και πατήστε το κουμπί Install.

Αυτό ήταν! Μετά από λίγα λεπτά, θα ‘χετε μπροστά σας την επιφάνεια εργασίας του Mac OS X!
Κάτι που θα παρατηρήσετε σύντομα είναι ότι η ανάλυση στην εικονική μηχανή επιμένει να μην αλλάζει με κανένα τρόπο! Μια αποτελεσματική λύση αν και λίγο άκομψη, είναι να επέμβετε πάλι στο αρχείο ρυθμίσεων που είχατε τροποποιήσει νωρίτερα. Για ακόμη μια φορά λοιπόν, κλείστε το VirtualBox, ανοίξτε το εν λόγω αρχείο και στο τέλος της περιοχής ExtraData προσθέστε τη παρακάτω γραμμή:

```
<ExtraDataItem name=”VBoxInternal2/EfiGopMode” value=”N”/>
```

Σημειώστε ότι στη θέση του N πρέπει να βάλετε ένα αριθμητικό ψηφίο από το 0 εως 4. Τα ψηφία αντιστοιχούν στις αναλύσεις 640x480, 800x600, 1024x768, 1280x1024, 1440x900. Αν για παράδειγμα επιλέξετε το 3, μόλις αποθηκεύσετε το αρχείο και ξεκινήσετε την εικονική μηχανή η ανάλυσή της θα είναι 1280x1024.
Κάτι άλλο που θα παρατηρήσετε σύντομα είναι ότι η εικονική μηχανή δεν έχει καθόλου ήχο. Γι αυτό το λόγο, μέσα από το Mac OS X θα πρέπει να κατεβάσετε και να εγκαταστήσετε έναν εξειδικευμένο driver ήχου. Συγκεκριμένα κατεβάστε από εδώ το αρχείο “VirtualBox ICH AC97 Audio Driver.zip”. Όταν ολοκληρωθεί η λήψη, εκτελέστε το πρόγραμμα που βρίσκεται μέσα στο ZIP. Κάντε τα προφανή κλικ και σε λίγη ώρα θα έχετε και ήχο!

1.4.2 Hackintosh

Εάν και η εγκατάσταση του εικονικού λειτουργικού Mac OS X ήταν επιτυχής στο συγκεκριμένο υπολογιστή με επεξεργαστή Intel που δοκίμασα, παρόλα αυτά το ότι στο συγκεκριμένο υπολογιστή δεν είχα άμεση πρόσβαση, δημιούργησε την ανάγκη να καταφύγω σε κάποια άλλη εναλλακτική. Έχοντας στην ιδιοκτησία μου ένα netbook αποφάσισα να πειραματιστώ μ' αυτό, και έτσι έκανα την απόπειρα να εγκαταστήσω σ' αυτό τα Hackintosh. Τα Hackintosh αποτελούν μια πειραγμένη έκδοση των Mac OS X η οποία επιτρέπει την εγκατάσταση τους native σε PC. Η εγκατάσταση των Hackintosh στο netbook μου αποτελούσε μονόδρομο αφού ο επεξεργαστής που διέθετε, Intel Atom, δε προσφέρει τη τεχνολογία Intel-VT. Το netbook που χρησιμοποίησα ήταν ένα HP Mini 1000.

Για άλλη μια φορά, φρόντισα να περιγράψω τη διαδικασία που ακολούθησα υπό μορφή βημάτων έτσι ώστε να μπορεί να την ακολουθήσει και οποιοσδήποτε άλλος επιθυμεί. Αν λοιπόν ενδιαφέρεται κάποιος για την εξής διαδικασία ορίστε τα βήματα που πρέπει να ακολουθήσει:

Χρειαστήκαμε:

- Έναν υπολογιστή Mac (Intel-based) ο οποίος τρέχει ήδη Mac OS X και ο οποίος διαθέτει μονάδα DVD και τουλάχιστον μια θύρα USB (θα τον χρειαστούμε περίπου 1-2 ώρες).
- Ένα δισκάκι εγκατάστασης του Mac OS X 10.6 Snow Leopard (build 10A432), είτε σε μορφή DVD ή ακόμα και σε μορφή εικονικού δίσκου dmg.
- Ένα φλασάκι (USB Flash drive) χωρητικότητας τουλάχιστον 8GB (Μπορούμε να χρησιμοποιήσουμε και εξωτερικό σκληρό δίσκο αλλά θα πρέπει να ολόκληρος άδειος)
- Ακόμη ένα φλασάκι με τουλάχιστον 100MB ελεύθερο χώρο.

Εφόσον έχετε όλα αυτά ξεκινήστε να κατεβάσετε τα αρχεία και λογισμικό που θα χρειαστούμε:

- NetbookBootMaker 0.8.4 RC1 – Supports Mac OS X 10.6.3
- NetbookInstaller 0.8.4 RC1 – Supports Mac OS X 10.6.3
- GeneralExtensions Bundle (includes SleepEnabler for Kernel 10.2.0 for Mac OS X 10.6.3)
- Tea's Mach Kernel
- DSDT Patcher 1.0.1e

Εφόσον τα κατεβάσετε, δημιουργήστε ένα φάκελο στην επιφάνεια εργασίας σας με το όνομα MacHackPC και μεταφέρετε τα αρχεία σ' αυτόν. Έπειτα μεταφέρετε το φάκελο στο φλασάκι με τα ελεύθερα 100MB.

Στο Mac υπολογιστή:

1. Εισάγετε το φλασάκι (αυτό που είναι τουλάχιστον 8GB) ή τον εξωτερικό σκληρό δίσκο.
2. Εισάγετε το DVD εγκατάστασης του Mac OS X 10.6 Snow Leopard, ή κάντε mount το dmg του (διπλό κλικ πάνω του).
3. Εκκινήστε το Disk Utility (χρησιμοποιήστε το SpotLight πάνω δεξιά για να το βρείτε).
4. Θα εμφανιστεί η εφαρμογή Disk Utility
5. Διαλέξτε το USB μέσο σας (φλασάκι ή δίσκο), που πρέπει να βρίσκεται στα αριστερά, επιλέξτε τη καρτέλα Partition, επιλέξτε "1 Partition" στο μενού επιλογών του Volume

Scheme, τότε αλλάξτε στο Volume Information το όνομα σε “MacOSXDVD” και για Format επιλέξτε Mac OS Extended (Journaled). Ακολούθως πατήστε το κουμπί Options.
6. Επιλέξτε “Master Boot Record” και πατήστε OK.
7. Θα επιστρέψετε έτσι πίσω στο παράθυρο του Disk Utility, όπου θα πατήσετε το Apply και ακολούθως το κουμπί Partition.
8. Εφόσον τελιώσει η διαδικασία, πατήστε στη καρτέλα Restore, επιλέξτε και τραβήξτε το “Mac OS X Install DVD” από τα αριστερά στο πεδίο Source, το “MacOSXDVD” πάλι από τα αριστερά στο πεδίο Destination και πατήστε Restore.
9. Ξαναπατάτε Restore. Κάπου εδώ θα σας ζητηθει να εισάγετε το κωδικό admin σας, τον εισάγετε και ξεκινάει η διαδικασία. Περιμένετε περίπου 10 λεπτά μέχρι να τερματίσει.
10. Εφόσον τελιώσει το Restore, τρέξτε την εφαρμογή NetBookBootMaker. Στο παράθυρο που θα ανοίξει, στο Select USB Partirion επιλέξτε το MacOSXDVD και θα πατήστε το κουμπί Prepare Boot Drive. Εισάγετε πάλι τον admin κωδικό σας. Εφόσον τελιώσει η διαδικασία θα εμφανιστεί το μήνυμα “The USB Dirve has been prepared. You may boot from it on your NetBook”
11. Έπειτα κάντε δεξί κλικ στο MacOSXDVD (από το desktop σας), και επιλέξτε Open. Μετά πατήστε Shift+Command+G, και στο διάλογο που θα εμφανιστεί γράψτε: “/Volumes/MacOSXDVD/System/Installation/Packages/” και πατήστε Go.
12. Ψάξτε και βρείτε το αρχείο “OSInstall.pkg.orig”, μεταφέρετέ το στο desktop σας και μετονομάστε το σε “OSInstall.pkg”. Τώρα επιτρέξτε στο παράθυρο απ’ όπου πήρατε το “OSInstall.pkg.orig”, ψάξτε βρείτε το “OSInstall.pkg” και διαγράψτε το (θα σας ζητηθεί πάλι ο κωδικός admin σας). Εφόσον διαγραφεί, απλά μεταφέρετε πίσω στο παράθυρό μας το αρχείο “OSInstall.pkg” από το desktop (πάλι εισάγετε κωδικό admin). Εφόσον μεταφερθεί, διαγράψτε και το αρχείο “OSInstall.pkg.orig”.
13. Τώρα κάντε δεξί κλικ στο “MacOSXDVD” και επιλέξτε Eject.

Τέλος με τον υπολογιστή Mac, τώρα συνεχίζουμε με το δικό μας υπολογιστή, τον HP Mini 1000.

Στο HP Mini 1000:

1. Εισάγετε το USB μέσο “MacOSXDVD” στον υπολογιστή, εκκινήστε τον υπολογιστή, και καθώς κάνει boot πατήστε το F9. Φροντίστε έτσι ώστε το USB μέσο “MacOSXDVD” να είναι το μόνο USB που είναι συνδεδεμένο στον υπολογιστή.
2. Θα βρεθείτε στο μενού με τις επιλογές boot, απ’ όπου θα επιλέξετε το USB Drive/External Hard drive.
3. Μ’ αυτό το τρόπο θα κάνει boot από το USB μέσο. Έπειτα θα βρεθείτε στο περιβάλλον εγκατάστασης του Mac OS. Επιλέγετε γλώσσα, πατάτε το κουμπί με το βελάκι και ακολούθως θα εμφανιστεί το παράθυρο Install Mac OS. Δεν πατάτε Continue, αλλά από το μενού πάνω, επιλέγετε Utilities->Disk Utility.
4. Τώρα επιλέξτε τον εσωτερικό σκληρό δίσκο του HP Mini ο οποίος θα βρίσκεται στα αριστερά, πηγαίνετε στη καρτέλα Partition, επιλέξτε “1 Partition” στο μενού επιλογών του Volume Scheme, αλλάξτε στο Volume Information το όνομα σε “Mac OS X” και για Format επιλέξτε Mac OS Extended (Journaled). Ακολούθως πατήστε το κουμπί Options, επιλέξτε GUID Partition Table και πατήστε OK.
5. Τώρα, πίσω στο παράθυρο του Disk Utility, πατάτε Apply και έπειτα Partition. Εφόσον

τελιώσει η διαδικασία κλείνετε το Disk Utility.
6. Τώρα απλά ακολουθείτε τις οδηγίες επί της οθόνης και εγκαθιστάτε το Mac OS.
7. Εφόσον τελιώσει η εγκατάσταση, ο υπολογιστής θα επανεκκινήσει. Καθώς κάνει boot πατάτε F9 και επιλέγετε να κάνει boot πάλι από το USB μέσο.
8. Εδώ, όταν δείτε στην οθόνη τον Mac OS X Bootloader, πατήστε όποιοδήποτε κουμπί για να διακόψετε τη διαδικασία.
9. Έπειτα πατήστε το arrow πλήκτρο για να επιλέξετε τον εσωτερικό δίσκο του HP Mini και πατήστε Enter. Από δω, απλά ακολουθήστε τις επί της οθόνης οδηγίες για να ολοκληρώσετε τη διαδικασία Standard Apple Setup για το υποτιθέμενο Mac σας (ή καλύτερα το MacHackPC σας :-)).
10. Σε αυτό το σημείο θα χρειαστούμε το άλλο φλασάκι. Το εισάγουμε στο HP Mini, και εκκινούμε την εφαρμογή NetbookBootMaker. Επιλέξτε τον εσωτερικό σκληρό δίσκο σας "Mac OS X" και πατήστε Prepare Boot Drive (θα σας ζητηθεί να εισάγετε τον admin κωδικό σας τον οποίο επιλέξατε κατα την εγκατάσταση του Mac OS στον HP Mini σας στο βήμα 9).
11. Τώρα, εκκινήστε την εφαρμογή NetBookInstaller και πατήστε Install.
12. Εφόσον ολοκληρωθεί η διαδικασία, επανεκκινήστε τον υπολογιστή σας. Θα πρέπει να κάνει boot στα Mac OS X. Αφού εκκινήσουν μπορείτε πλέον να κάνετε Eject το USB μέσο "MacOSXDVD".

Η εγκατάσταση του Mac OS X ολοκληρώθηκε. Δεν έχουμε τελιώσει όμως εντελώς. Υπάρχουν κάποιες ρυθμίσεις που πρέπει να κάνουμε ακόμα για να καταστήσουμε όλο το υλικό του υπολογιστή μας (hardware) λειτουργικό.

Ήχος και TrackPad

1. Ανοίξτε ένα παράθυρο Finder, και ακολούθως πατήστε στο πληκτρολόγιο SHIFT+ALT+G. Τώρα πληκτρολογήστε "/Extra" και πατήστε GO (εδώ βασικά μπαίνετε σε ένα hidden φάκελο που ονομάζεται Extra)
2. Τώρα, πάρτε το NewGeneralExtensions Bundle από το φλασάκι και μεταφέρετε το στο φάκελο Extra (αφήστε το να κάνει ovewrite το ήδη υπάρχων αρχείο, θα σας ζητηθεί κωδικός admin).
3. Τώρα πάρτε το Tea's Mach Kernel από το φλασάκι και μεταφέρετε το στο root του εσωτερικού σας δίσκου (πάλι θα σας ζητηθεί κωδικός admin).
4. Τώρα πηγαίνετε πίσω στο φάκελο Extra και ψάξτε για το αρχείο "com.apple.Boot.plist" και αντιγράψτε το στο desktop σας.
5. Ανοίξτε το αρχείο "com.apple.Boot.plist" απ' το desktop σας, και ψάξτε για τη γραμμή 12 η οποία μοιάζει κάπως έτσι <string>mach_kernel</string>. Αλλάξτε αυτή τη γραμμή σε <string>mach_kernel_atom2</string> και αποθηκεύστε την αλλαγή.
6. Τώρα, αντιγράψτε το αλλαγμένο plist αρχείο πίσω στο φάκελο Extra (αφήστε το να κάνει ovewrite το ήδη υπάρχων αρχείο, θα σας ζητηθεί κωδικός admin).
7. Τώρα, πάρτε το αρχείο SleepEnabler.Kext από το φάκελο "SleepEnabler for Kernal 10.2.0 for Mac OS X 10.6.3" που είχαμε κατεβάσει στην αρχή και μεταφέρετε το στο φάκελο Extra/GeneralExtensions.
8. Τώρα τρέξτε το UpdateExtra.app που βρίσκεται στο φάκελο Extra και πατήστε Update Extensions. Αυτή η διαδικασία θα κρατήσει για λίγο (πάλι θα σας ζητηθεί κωδικός admin).

Τελιώσαμε και με αυτό. Τώρα είσαστε έτοιμοι για να κάνετε update τα Snow Leopard σε 10.6.3. Βρείτε τον Updater του Mac OS στο MacHackPC σας και κάντε update. Αυτό θα πάρει αρκετό χρόνο. Ακολουθεί η διαδικασία για να φτιάξετε τη λειτουργία Sleep.

Λειτουργία Sleep

1. Τρέξτε την εφαρμογή "DSDT Patcher 1.0.1e" που βρίσκεται στο φάκελο DSDT_Patcher1.0.1e στο φλασάκι (θα παρατηρήσετε ότι θα τρέξει σε περιβάλλον κονσόλας).
2. Πατήστε Enter για να συνεχίσετε.
3. Όταν θα σας ξητηθεί πατήστε 0 και έπειτα το Enter και περιμένετε μέχρι να εμφανιστεί το μήνυμα "Process Completed". Εφόσον γίνει μπορείτε πλέον να κλείσετε τη κονσόλα.
4. Τώρα, κάντε copy το αρχείο iasl το οποίο βρίσκεται στο DSDT_Patcher1.0.1e/Tool/, ανοίξτε ένα καινούριο παράθυρο Finder και πατήστε στο πληκτρολόγιο SHIFT+ALT+G. Όταν ανοίξει το textbox, πληκτρολογήστε /usr/bin/ και πατήστε GO. Αντιγράξτε το αρχείο iasl εδώ.
5. Τώρα πηγαίνετε στο φάκελο DSDT_Patcher1.0.1e/Debug/ και ανοιξτε το αρχείο dsdt.dsl. Όταν ερωτηθείτε "Open with" επιλέξτε TextEdit.
6. Τώρα πατήστε ALT+F και ψάξτε για το "Device USB". Τώρα βρείτε τον ακόλουθο κώδικα και βάλτε τον σε σχόλια χρησιμοποιώντας /* και */.

Current Code

```
Device (EUSB)
{
Name (_ADR, 0x001D0007)
Method (_PRW, 0, NotSerialized)
{
Return (GPRW (0x0D, 0x04))
}
}
```

Changed Code

```
/*  Device (EUSB)
{
Name (_ADR, 0x001D0007)
Method (_PRW, 0, NotSerialized)
{
Return (GPRW (0x0D, 0x04))
}
}
*/
```

7. Τώρα πατήστε πάλι ALT+F και ψάξτε για το “Method (_L0D, 0, NotSerialized)”. Βάλτε σε σχόλια τη γραμμή που φαίνεται παρακάτω.

Current Code

```
Method (_L0D, 0, NotSerialized)
{
Notify (\_SB.PCI0.EUSB, 0x02)
Notify (\_SB.PWRB, 0x02)
}
```

Changed Code

```
Method (_L0D, 0, NotSerialized)
{
/* Notify (\_SB.PCI0.EUSB, 0x02) */
Notify (\_SB.PWRB, 0x02)
}
```

8. Τώρα, ξανά ALT+F, ψάξτε για το “Device (PWRB)” και κάντε τις εξής αλλαγές:

Current Code

```
Device (PWRB)
{
Name (_HID, EisaId ("PNP0C0C"))
Name (_UID, 0xAA)
Name (_STA, 0x0B)
}
}
```

Changed Code

```
Device (PWRB)
{
Name (_CID, EisaId ("PNP0C0C"))
Name (_UID, 0xAA)
Name (_STA, 0x0B)
}
}
```

9. Τώρα που αλλάξαμε ότι έπρεπε να αλλάξουμε, αποθηκεύστε το αρχείο.
10. Τώρα ανοίξτε τη κονσόλα (Terminal) και πληκτρολογήστε "*sudo -s*". Όταν σας ζητηθεί κωδικός admin εισάγετε τον και πατήστε Enter.
11. Τώρα πληκτρολογήστε: *cd /GO/TO/MacHackPC/DSDT_Patcher1.0.1e/Debug/* και πατήστε Enter.
12. Έπειτα πληκτρολογήστε: "*iasl -ta dsdt.dsl*" και πάλι Enter. Όταν εκτελεστεί η εντολή θα εμφανιστεί το μήνυμα "Compilation Completed".
13. Τώρα ανοίξτε τον Finder και μεταβείτε στο /DSDT_Patcher1.0.1e/Debug/ και αντιγράξτε το αρχείο dsdt.aml στο φάκελο Extra του υπολογιστή σας.
14. Επανεκκινήστε τον υπολογιστή.

Εικονίδιο του σκληρού δίσκου

1. Ανοίξτε το Terminal και πληκτρολογήστε "*sudo -s*". Εισάγετε κωδικό admin και πατήστε Enter.
2. Έπειτα πληκτρολογήστε τα παρακάτω σε μια μόνο γραμμή και πατήστε Enter.

```
mv
/System/Library/Extensions/IOStorageFamily.kext/Contents/Resources/External.icns
/System/Library/Extensions/IOStorageFamily.kext/Contents/Resources/External.icns.b
ack
```

3. Πληκτρολογήστε πάλι τα παρακάτω σε μια μόνο γραμμή και πατήστε Enter.

```
cp /System/Library/Extensions/IOStorageFamily.kext/Contents/Resources/Internal.icns
/System/Library/Extensions/IOStorageFamily.kext/Contents/Resources/External.icns
```

4. Το ίδιο...

```
chown -R root:wheel /System/Library/Extensions/IOStorageFamily.kext
```

5. Ξανά...

```
chmod -R 755 /System/Library/Extensions/IOStorageFamily.kext
```

6. Τώρα επανεκκινήστε τον υπολογιστή και απολάυστε το HP Mini σας το οποίο πλέον τρέχει τα Mac OS X 10.6.3.

Κεφάλαιο 2 Περιβάλλον iOS

2.1 iOS SDK

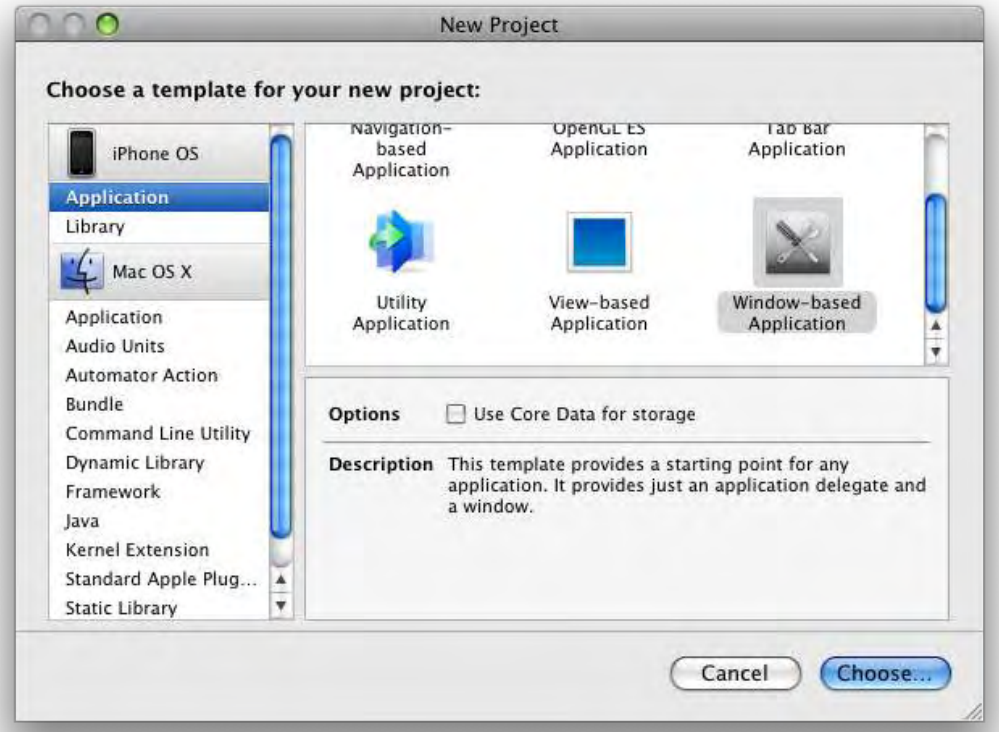
Το iOS SDK αποτελεί το λογισμικό με όλα όσα χρειάζεται ένας προγραμματιστής για να αναπτύξει και να δημιουργήσει εφαρμογές για μια iOS συσκευή. Μπορεί να το βρει κανείς στο δισκάκι εγκατάστασης των Mac OS X, αλλά είναι προτιμότερο να μην το κάνει. Ο λόγος είναι επειδή το πιο πιθανόν η έκδοση που βρίσκεται στο δισκάκι να μην είναι η τελευταία αλλά να είναι ξεπερασμένη και παλιά. Καλύτερα να το κατεβάσει από την ιστοσελίδα της Apple, πράγμα που μπορεί να κάνει εντελώς δωρεάν. Το μόνο που χρειάζεται είναι να έχεις κάποιο Apple ID.
Η σουίτα iOS SDK λοιπόν αποτελείται, εκτός των άλλων, από όλο τον απαιτούμενο κώδικα, βιβλιοθήκες, packages, frameworks για τις κλάσεις που είναι διαθέσιμες από την Apple, για να χρησιμοποιήσει ο προγραμματιστής και να αναπτύξει την εφαρμογή του.
Από κει και πέρα συμπεριλαμβάνει την εφαρμογή Xcode, η οποία είναι και η σημαντικότερη όλων, μέσω της οποίας θα μπορέσει κάποιος να αναπτύξει την εφαρμογή του.
Η τελευταία έκδοση που κυκλοφορεί αυτή τη χρονική στιγμή είναι το iOS SDK 4.2 με το Xcode 3.2.5. Για τη διπλωματική αυτή χρησιμοποιήθηκε όμως το iOS SDK 3.1 με το Xcode 3.1 το οποίο ήταν διαθέσιμο κατά την έναρξη της εκπόνησης της. [2]

2.2 Xcode

Αν και αποτελεί σουίτα με ένα αριθμό εργαλείων και εφαρμογών για τη δημιουργία και ανάπτυξη μιας εφαρμογής, παρόλα αυτά περιλαμβάνει και την εφαρμογή με το ίδιο όνομα, Xcode.

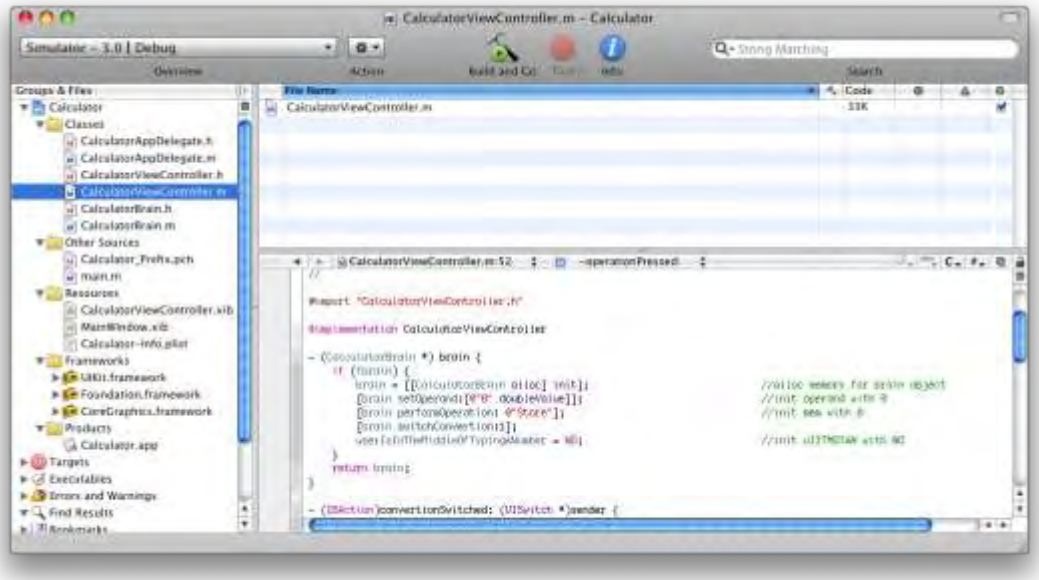
2.2.1 Xcode

Η εφαρμογή Xcode προσφέρει το περιβάλλον για τη δημιουργία και τροποποίηση ενός project, τον editor για την εγγραφή του κώδικα και μέσω αυτής τη διασύνδεση με όλα τα απαραίτητα κομμάτια της σουίτας Xcode αλλά και του SDK, για να γίνει compile μια εφαρμογή. Με την ολοκλήρωση της δημιουργίας μιας εφαρμογής, προσφέρει και τη δυνατότητα να την περάσεις στη iΣυσκευή σου ή να την ανεβάσεις στο AppStore της Apple.



Κατά τη δημιουργία ενός project μπορεί κάποιος να επιλέξει για ποιο λειτουργικό θέλει να αναπτύξει μια εφαρμογή. Αυτό συμβαίνει επειδή το Xcode αποτελεί τη σουίτα για την ανάπτυξη εφαρμογών και για το Mac OS X. Στη δική μας περίπτωση, αφού επιλέξουμε Application για iOS, έπειτα έχεις να διαλέξεις το template για την εφαρμογή που θα αναπτύξεις. Εκτός των άλλων υπάρχουν οι επιλογές για Application Navigation-based, OpenGL ES, View-based, Window-based κτλ.
Τα template αυτά βοηθούν στο να παραχθούν αυτόματα τα αρχεία των απαιτούμενων κλάσεων και ένας αριθμός από έτοιμα κομμάτια κώδικα, με σκοπό τη βελτίωση της

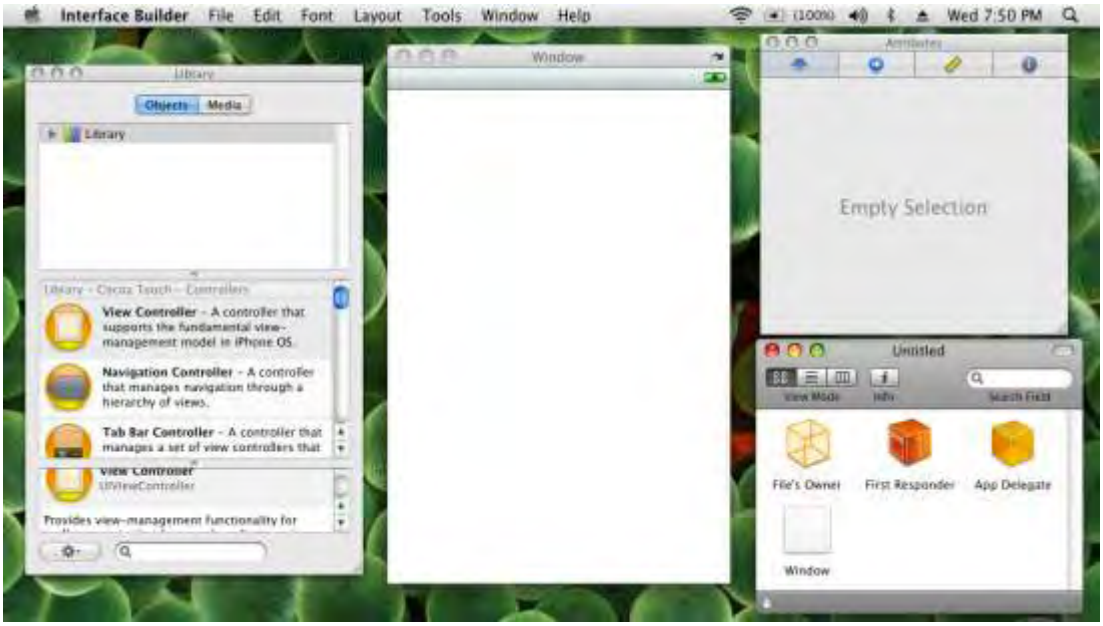
παραγωγικότητας και οργάνωσης του project σου.



Το κυρίως παράθυρο του Xcode αποτελείται από το πλαίσιο με την οργάνωση των αρχείων του project σε φακέλους, το παράθυρο με τα ονόματα των αρχείων κώδικα και τον editor για την τροποποίηση των αρχείων κώδικα. Από πάνω βρίσκεται το κυρίως μενού επιλογών και μερικές συντομεύσεις για το compile και τρέξιμο της εφαρμογής προς ανάπτυξη.
Αξίζει να αναφερθούμε στη συντόμευση “Build and Go” με την οποία ξεκινάει η διαδικασία Compile και Build και ακολούθως εκκινείται αυτόματα η εφαρμογή Simulator στην οποία θα τρέξει σε περιβάλλον προσομοίωσης η εφαρμογή προς ανάπτυξη.
Ένα αρκετά σημαντικό αρχείο αποτελεί το αρχείο με τη κατάληξη .xib που βρίσκεται στο φάκελο Resources και το οποίο είναι υπεύθυνο για το γραφικό περιβάλλον του χρήστη, το GUI της εφαρμογής προς ανάπτυξη. Κάνοντας διπλό κλικ σε αυτό το αρχείο εκκινείται αυτόματα η εφαρμογή Interface Builder, η οποία προσφέρει τα εργαλεία για τη τροποποίηση του αρχείου .xib.

2.2.2 Interface Builder

Όπως αναφέραμε πιο πάνω, η εφαρμογή Interface Builder(IE) βρισκεται στη σουίτα Xcode και μπορεί να εκκινήσει αυτόματα με το άνοιγμα ενός αρχείου .xib. Η εφαρμογή IE είναι το εργαλείο για τη σχεδίαση του GUI μιας εφαρμογής προς ανάπτυξη και στη συνέχεια της διασύνδεσης του με τον κώδικα. Αποτελείται από τα παράθυρα Library, Window, Inspector και το παράθυρο με το τίτλο του αρχείο .xib.



Στο παράθυρο Library βρίσκονται όλα τα διαθέσιμα αντικείμενα που μπορούν να προστεθούν σ' ένα γραφικό περιβάλλον. Περιέχει κουμπιά, ετικέτες, textfield, διακόπτες κτλ. Απλά παίρνεις ότι θες από το παράθυρο αυτό και το τραβάς στο Window.
Το Window αποτελεί το παράθυρο με την υποτιθέμενη οθόνη της συσκευής (πχ. iPhone) και μέσα σ' αυτό κτίζεις το πως θα εμφανίζεται η εφαρμογή προς ανάπτυξη. Προσφέρονται βοηθητικές γραμμές για στοίχιση των αντικειμένων και στο πάνω μέρος της εμφανίζεται η μπάρα με την ένδειξη της μπαταρίας όπως θα ήταν στη πραγματική συσκευή για τη σωστή εκμετάλλευση του διαθέσιμου χώρου. Επιλέγοντας το κάθε αντικείμενο μέσα στο Window εμφανίζονται τα χαρακτηριστικά του στο παράθυρο Inspector.
Το παράθυρο Inspector όπως ήδη ανάφερα, συγκρατεί τα χαρακτηριστικα ενός αντικειμένου που εμφανίζεται στην οθόνη. Διαθέτει 4 καρτέλες, Atributes, Connections, Size και Identity. Για το Attributes δε χρειάζεται να μπούμε σε λεπτομέρειες αρκεί να αναφέρουμε ότι σ' αυτό θα βρεις χαρακτηριστικά όπως είναι ο τίτλος του αντικειμένου, χρώμα, σκίαση, διαφάνεια. Στο Connections υπάρχουν οι διαθέσιμες ενέργειες τις οποίες μπορεί να προκαλέσει ένα αντικείμενο και τι θα ενεργοποιήσει μ' αυτές, πχ. κάποια μέθοδο. Στο Size υπάρχουν οι συντεταγμένες ενός αντικειμένου, οι διαστάσεις του αλλά και επιλογές για Autosizing ή Aligmnment. Αυτές είναι ιδιαίτερα σημαντικές όταν θα θελήσετε να δημιουργήσετε μια εφαρμογή η οποία θα μπορεί να περιστρέφει το GUI της. Τέλος στο Identity βρίσκονται πληροφορίες για τη κλάση ενός αντικειμένου.
Το παράθυρο με το τίτλο του .xib αρχείου μας (Untitled εδώ) τώρα είναι σημαντικό για τη διασύνδεση του GUI με το κώδικα της εφαρμογής. Αρκεί να πούμε ότι ο κύβος με το όνομα File's Owner αντιπροσωπεύει τα αρχεία των κλάσεων και μέσω αυτού επιτυγχάνεται η συσχέτιση του GUI με τις μεθόδους και τη λειτουργία της εφαρμογής προς ανάπτυξη.

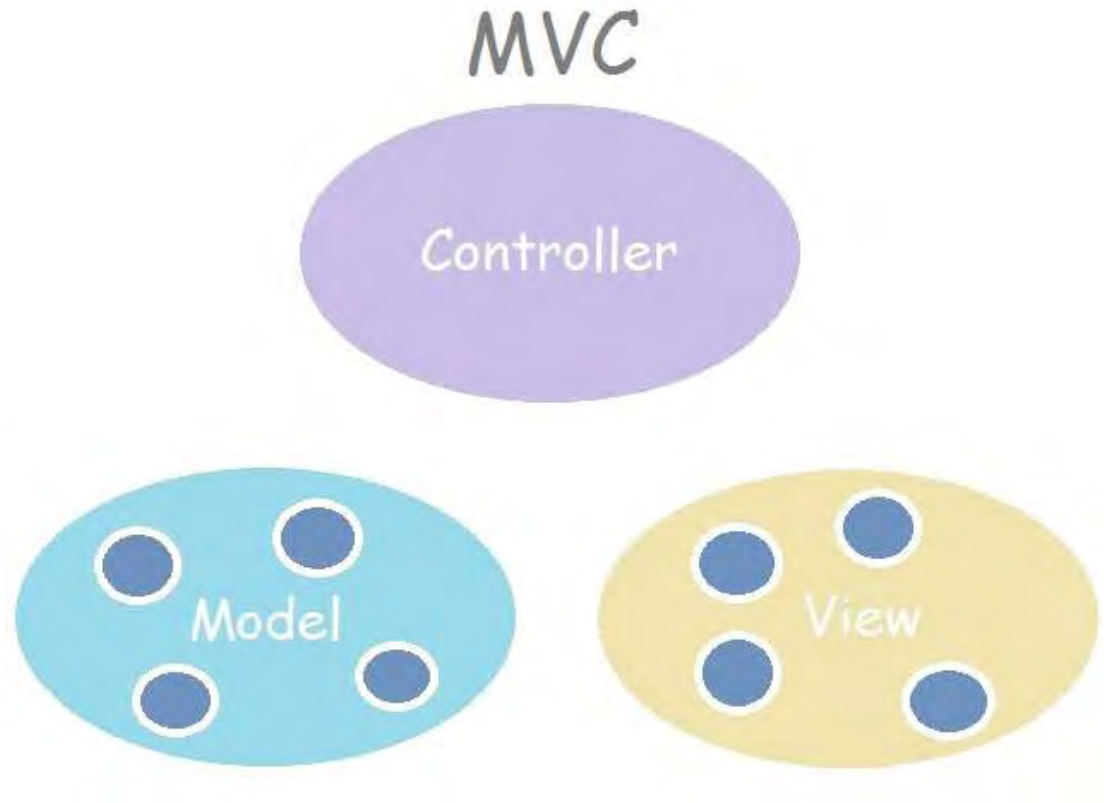
2.2.3 Simulator

Ο Simulator αποτελεί το εργαλείο στο οποίο θα τρέξουμε την εφαρμογή προς ανάπτυξη και θα επιτρέψει να εισαχθεί είσοδος από τον χρήστη. Το ποντίκι προσομοιώνει το άγγιγμα του δακτύλου, και ότι σχεδιάσαμε στον Interface Builder εμφανίζεται εδώ. Έπειτα αλληλεπιδρά με τον χρήστη, εμάς, και τρέχει το κώδικα που αναπτύξαμε. Προσφέρει ακόμα και σύνδεση Internet χρησιμοποιώντας το δίκτυο του υπολογιστή αλλά και ήχο. Κάτι σημαντικό που αξίζει να αναφέρουμε είναι η επιλογή Hardware > Rotate που μας δίνει τη δυνατότητα να δούμε πως αντιδρά η εφαρμογή όταν θα περιστρεφόταν η πραγματική συσκευή.

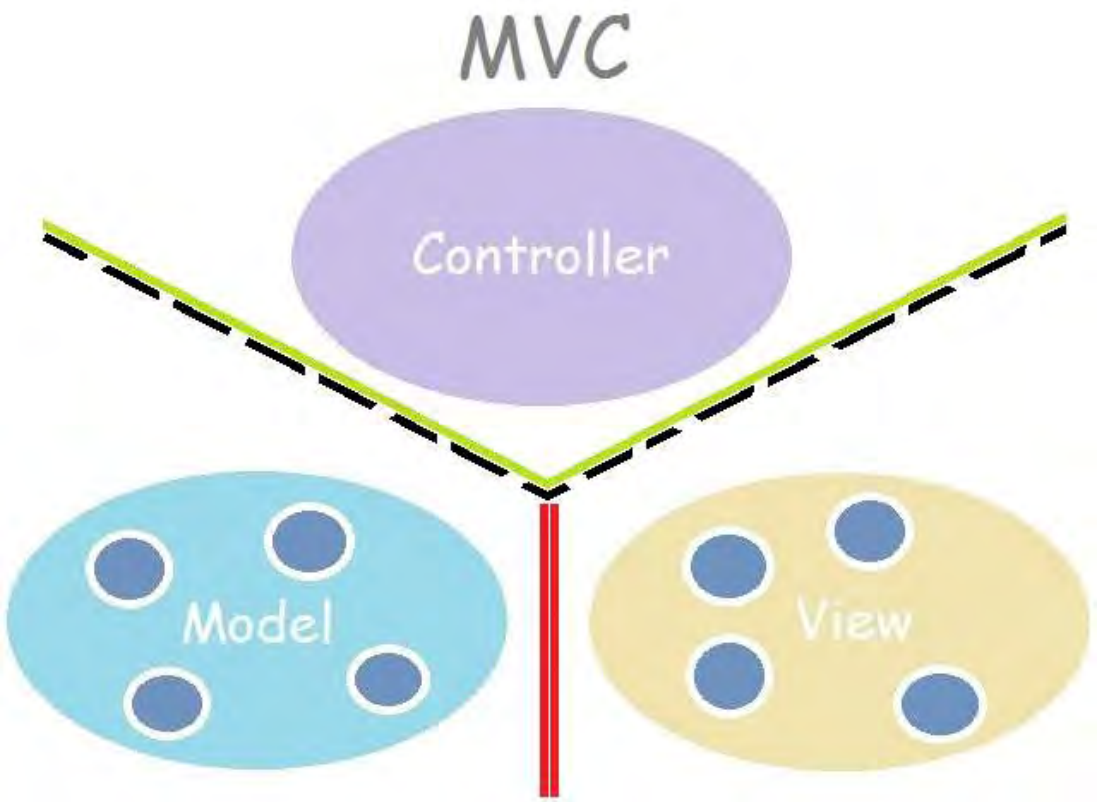
2.3 Εγγραφή Κώδικα

2.3.1 Μοντέλο MVC

Πρόκειται για ένα μοντέλο που έχει υιοθετηθεί στη δημιουργία εφαρμογών για iOS με Objective-C. Αν και δεν είναι υποχρεωτικό να το ακολουθήσει κάποιος παρόλα αυτά εάν θέλει να δημιουργήσει μια αποδοτική, ευανάγνωστη (τόσο από τον εαυτό του αλλά και από τους υπόλοιπους) εφαρμογή καλό είναι να το ακολουθήσει.
Το MVC είναι το ακρωνύμιο για τις λέξεις Model - View - Controller. Κάθε αντικείμενο που δημιουργούμε για μια εφαρμογή μας πρέπει να ξέρουμε σε ποιο από τα 3 “στρατόπεδα” ανήκει. Μ‘ αυτό το τρόπο είναι πιο εύκολη η οργάνωση του κώδικά μας, για την διόρθωση και εντόπιση τυχών σφαλμάτων αλλά και για την επικοινωνία πολλών αντικειμένων μεταξύ τους [3].
Το Model αποτελεί το πυρήνα και εγκέφαλο ενός αντικειμένου. Σ’ αυτό περιγράφεται και υλοποιείται πλήρως η λειτουργία της εφαρμογής μας, χωρίς όμως να περιγράφεται και το πως εμφανίζεται στην οθόνη. Ο Controller είναι το τμήμα το οποίο είναι υπεύθυνο για το πως το μοντέλο (Model) εμφανίζεται στην οθόνη. Τέλος το View αποτελεί τα “τσιράκια” του Controller, και είναι υπεύθυνο για να εμφανίζει την έξοδο στο χρήστη αλλά και να αποτελεί τη διεπαφή μεταξύ αυτού και της εφαρμογής (Controller).

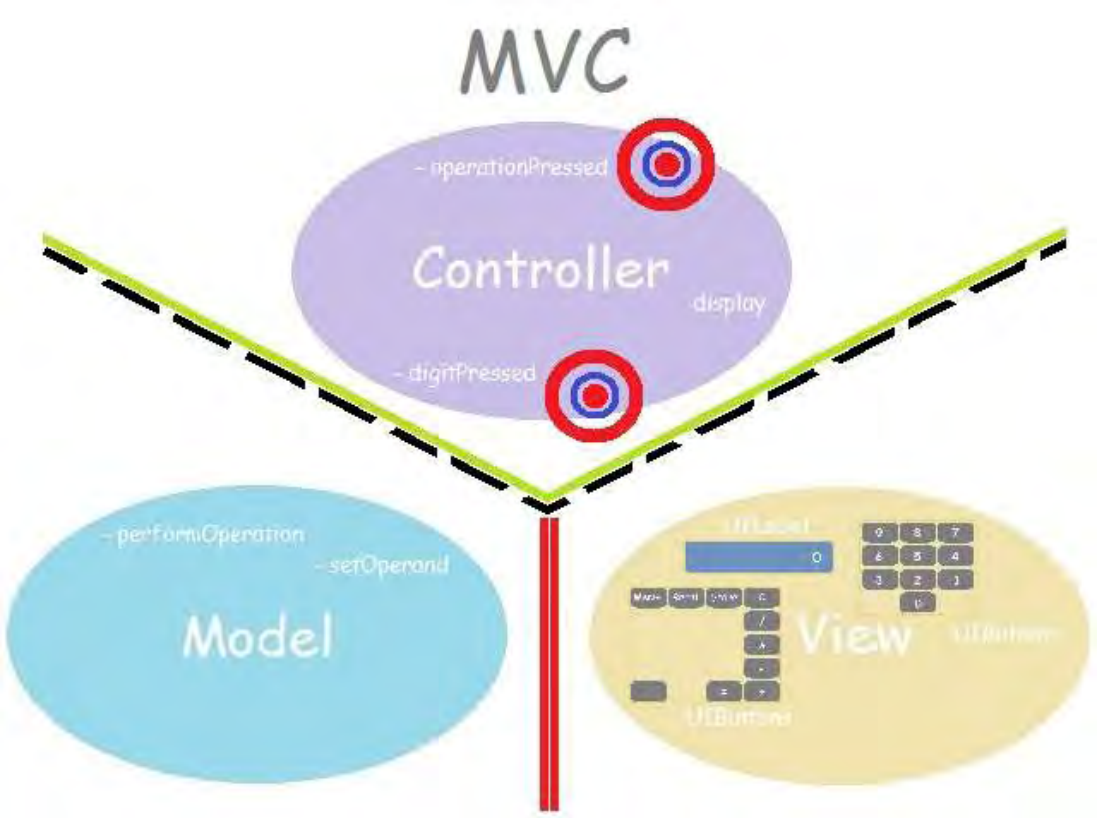


Περιγράψαμε τι είναι η κάθε ομάδα, αλλά πως επικοινωνούν μεταξύ τους? Λοιπόν η επικοινωνία είναι δυνατή μεταξύ τους όπως παρουσιάζεται στο επόμενο σχήμα.



Μεταξύ του Model και View δεν υπάρχει καμιά επικοινωνία, εξού και η διπλή κόκκινη γραμμή. Τώρα μεταξύ του Controller και των 2 άλλων ομάδων υπάρχουν κάποιες αλληλεπιδράσεις, πιο ελεύθερες απ' τη μεριά του Controller και κάπως πιο ελεγχόμενες απ' τις άλλες 2 ομάδες. Τι εννοούμε? Για να εξηγήσουμε καλύτερα πως εγκαθίσταται και λειτουργεί η επικοινωνία μεταξύ των 3 ομάδων ας δούμε ένα παράδειγμα μιας απλής εφαρμογής, πχ. ενός Calculator.

Πώς θα χωρίζονταν τα κομμάτια κώδικα μιας τέτοιας εφαρμογής? Λοιπόν, κάπως έτσι:



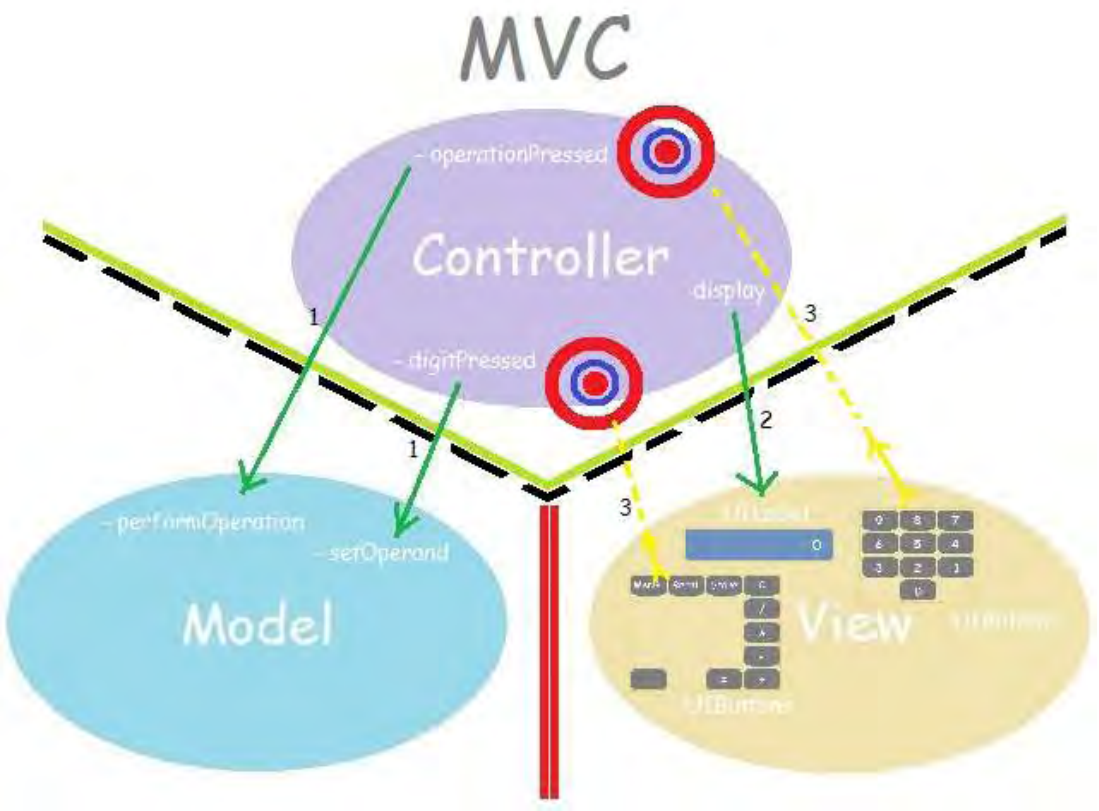
Model
performOperation: εκτελεί την πράξη και υπολογίζει το αποτέλεσμα
setOperand: θέτει τον τελεστή

Controller
operationPressed: η μέθοδος που ενεργοποιείται όταν ο χρήστης πατήσει κάποια πράξη
digitPressed: η μέθοδος που ενεργοποιείται όταν ο χρήστης πατήσει κάποιο ψηφίο
display: η οντότητα που είναι υπεύθυνη για το τι θα απεικονίζεται στην οθόνη του Calculator

Εδώ αξίζει να παρατηρήσουμε τους 2 “στόχους” που βρίσκονται στις 2 μεθόδους του Controller. Αποτελούν στοιχεία μέσω των οποίων πραγματοποιείται η επικοινωνία μεταξύ του Controller και του View. Περισσότερα θα πούμε στη συνέχεια.

View
UIButtons: κουμπιά στο περιβάλλον χρήστη
UILabel: ετικέτα κειμένου στο περιβάλλον χρήστη

Και πως επικοινωνούν μεταξύ τους? Κάπως έτσι:



1. Ο Controller μπορεί να επικοινωνήσει άμεσα με το Model. Μ' αυτό το τρόπο μπορεί να καλέσει την setOperand για να τεθεί ο τελεστής, ή την performOperation για να εκτελεστεί κάποια πράξη. Αν και ο Controller παίρνει την είσοδο από το χρήστη (μέσω του View εν πρώτης φυσικά), το Model είναι αυτό που θα την αξιοποιήσει και θα ενημερώσει τη κατάσταση της εφαρμογής.
2. Εξίσου άμεσα μπορεί να επικοινωνήσει ο Controller και με το View. Διατηρώντας την οντότητα display μπορεί κάθε φορά να ενημερώνει την οθόνη του Calculator με την απαιτούμενη έξοδο. Το View δεν είναι ποτέ υπεύθυνο για τα δεδομένα που προβάλει.
3. Τώρα όπως αναφέραμε και πριν, ο Controller διατηρεί αυτούς τους στόχους, τους οποίους έχει προσαρτήσει σε κάποιες μεθόδους του (operationPressed και digitPressed), οι οποίοι περιμένουν να ανταποκριθούν στην αντιδραση(action) του περιβαλλοντος χρήστη. Με το που πατάει ο χρήστης ένα κουμπί, είτε αυτό είναι ψηφίο είτε είναι πράξη, παράγεται κάποια αντιδρασή(action) στην οποία ανταποκρίνεται ο Controller, ενεργοποιώντας τις μεθόδους operationPressed kai digitPressed αντίστοιχα και από κει και πέρα επικοινωνεί όπως χρειάζεται με το Model για να ενημερώσει τη κατάσταση της εφαρμογής.

Το όλο σενάριο που περιγράψαμε αποτελεί μόνο μια επιμέρους εικόνα για τα όσα διαδραματίζονται ή μπορούν να διαδραματιστούν στο μοντέλο MVC. Για την ώρα όμως θα αρκεστούμε μόνο σ' αυτά.

2.3.2 Objective-C

Για να μεταφέρουμε τον αλγόριθμό σε iOS και να υλοποιήσουμε την εφαρμογή μας, έπρεπε να μάθουμε πρώτα τη γλώσσα στην οποία αναπτύσσονται εφαρμογές για το iOS, την Objective-C.

Ιστορικά

Πρόκειται για μια αντικειμενοστραφής προσέγγιση της γλώσσας C που ανέπτυξε ο Brad Cox και Tom Love στη δεκαετία του 80’ και την λάνσαραν μέσω της εταιρίας τους, την Stepstone. Αργότερα εξαγοράστηκαν τα δικαιώματα της γλώσσας από την εταιρία Next, εταιρία την οποία ξεκίνησε ο Steve Jobs αμέσως μετά την αποχώρηση του από την Apple. Όταν αργότερα η εταιρία NeXT και το λειτουργικό NeXTStep OS της εξαγοράστηκαν από την Apple, ο Steve Jobs εκτός από το επιχειρηματικό μυαλό του έφερε πίσω μαζί του και το καινούριο του project, την Objective-C. Έτσι αποτέλεσε τη γλώσσα στην οποία θα στηριζόταν η ανάπτυξη εφαρμογών για Mac OS X αλλά αργότερα και του iOS [4].

Σχέση με την C

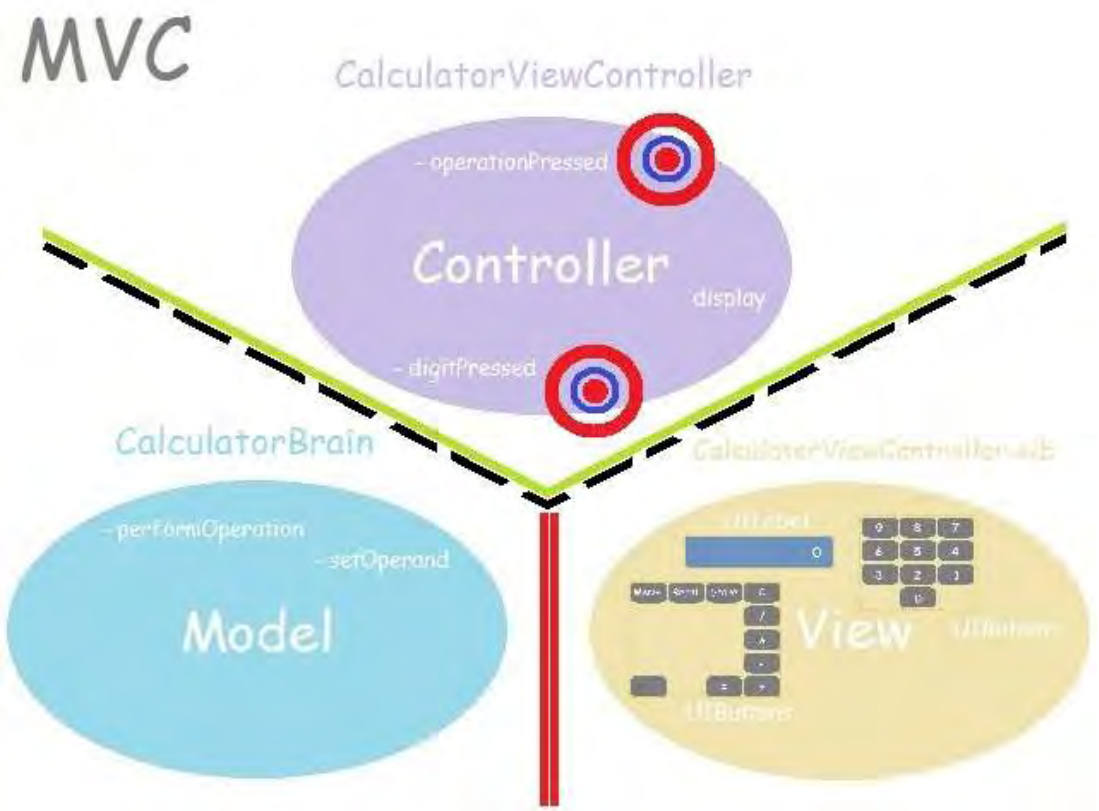
Αν και αποτελεί εξολοκλήρου μια καινούρια γλώσσα, παρόλα αυτά όντας πιστή στις ρίζες της, ο μεταγλωττιστής της, τουλάχιστον αυτός που χρησιμοποιείται στο Xcode, μπορεί να μεταγλωττίσει και να τρέξει natively και κώδικα C. Έχοντας λοιπόν μια μεγαλύτερη οικειότητα με τη γλώσσα C, ο αλγόριθμός KioustelidisQuadratic υλοποιήθηκε κυρίως σε C. Παρόλα αυτά όμως το UI και όλα τα υπόλοιπα κομμάτια κώδικα, πέρα του αλγόριθμου, γράφτηκαν σε Objective-C.

Αντικείμενα, Κλάσεις, Μεταβλητές, Μεθόδοι

Η Objective-C όντας αντικειμενοστραφής γλώσσα έχει να κάνει κυρίως με αντικείμενα και κλάσεις. Ισχύουν όλες οι αρχές του αντικειμενοστραφούς προγραμματισμού, όπως κληρονομικότητα, πολυμορφισμός, αφαίρεση, ενθυλάκωση και ως εργαλεία έχουμε πάλι τις μεταβλητές και τις μεθόδους.

Για την δημιουργία κάθε καινούριας κλάσης συντάσσεται ένα αρχείο με το όνομα της κλάσης και κατάληξη .m. Κάθε αρχείο .m που δημιουργείται έχει το αντίστοιχο αρχείο .h του με το ίδιο όνομα και μόνη διαφορά τη κατάληξή τους. Το αρχείο .h όπως και σε πολλές άλλες γλώσσες αποτελεί το public API της εκάστοτε κλάσης.

Για άλλη μια φορά θα πάρουμε ως παράδειγμα την εφαρμογή που είδαμε και πιο πάνω, το Calculator μας.



Model

CalculatorBrain.h

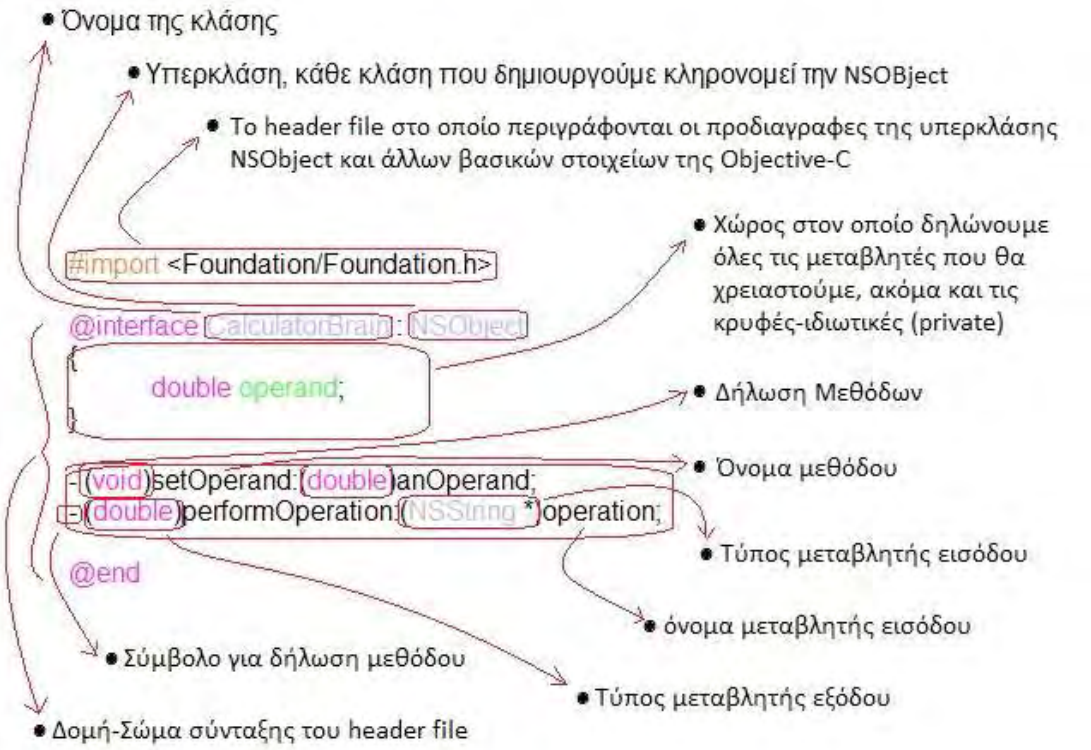
```
#import <Foundation/Foundation.h>

@interface CalculatorBrain : NSObject {
    double operand;
}

- (void)setOperand:(double)anOperand;
- (double)performOperation:(NSString *)operation;

@end
```

Αν και μικρό κομμάτι κώδικα, παρόλα αυτά περιέχει αρκετή πληροφορία για το πως συντάσσεται ένα header αρχείο στην Objective-C και από τι στοιχεία αποτελείται. Για πάμε να τα δούμε ένα ένα:



Υπάρχει ένας μεγάλος αριθμός τύπων μεταβλητών στην Objective-C, οι περισσότεροι οι οποίοι είναι οι συνήθεις που βρίσκεις σε κάθε γλώσσα. Να πούμε όμως ότι η Objective-C αναγνωρίζει όλους τους τύπους και της κανονικής C και μάλιστα χρησιμοποιεί και η ίδια αρκετούς απ' αυτούς.
Στην προηγούμενη εικόνα πιο πάνω φαίνεται πως και πού δηλώνουμε τις μεταβλητές της κλάσης μας. Τι δικαιοδοσία έχουν όμως? Όταν δεν υποδηλώνουμε καθόλου τη "δικαιοδοσία" μιας μεταβλητής τότε αυτή θεωρείται protected. Τι γίνεται όμως με τις private και public? Λοιπόν, ο τρόπος για να δηλώσουμε τέτοιες μεταβλητές είναι ο εξής:

```
@interface CalculatorBrain: NSObject
{
@public
        double operand;
@private
        doulbe a;

double b;

@protected
        double c;
}
```

Να αναφέρουμε τώρα ότι σχετικά με τις μεθόδους, υπάρχει ένας ιδιαίτερος τρόπος για το

πως δηλώνουμε μια μέθοδο με περισσότερες από μια μεταβλητές εισόδου. Το ενδιαφέρον σ' αυτό είναι ότι το όνομα της μεθόδου πλέον, χωρίζεται σε κομμάτια ανάλογα με τον αριθμό των μεταβλητών εισόδου. Για παράδειγμα, εάν θέλαμε να δηλώσουμε τη μέθοδο addAandB, η οποία προσθέτει 2 αριθμούς και επιστρέφει το αποτέλεσμα, θα το κάναμε κάπως έτσι:

```
- (double)addA:(double)a andB:(double)b;
```

Το όνομα της μεθόδου θα διαβαζόταν addA(colon)andB(colon), όπου colon η άνω και κάτω τελεία και θα είχε 2 μεταβλητές εισόδου, την a και την b.
Κάτι άλλο που αφορά στις μεταβλητές εισόδου σε μεθόδους, όταν πρόκειται για πέρασμα αντικειμένου σε μέθοδο, όπως με το (NSString) στη μέθοδο performOperation, πάντα γίνεται με τη χρήση pointer. Αυτό συμβαίνει λόγω της οργάνωσης της μνήμης τύπου heap. Δε περνάμε ΠΟΤΕ αντικείμενο σε μέθοδο χωρίς τη χρήση pointer, με μόνη εξαίρεση το τύπο μεταβλητής (id) για τον οποίο θα μιλήσουμε αμέσως παρακάτω.
Τέλος να αναφέρουμε και το τύπο μιας ειδικής μεταβλητής, της (id), η οποία αποτελεί ένα γενικευμένο τύπο αντικειμένου. Ο τύπος (id) μπορεί να προσκολλάται σε οποιοδήποτε αντικείμενο και μ' αυτόν να το χειριζόμαστε χωρίς να μας ενδιαφέρει ο πιο εξειδικευμένος τύπος του (πχ. NSString, NSArray κτλ). Όσο επικίνδυνο και αν ακούγεται αυτό, στη πραγματικότητα υπάρχουν τρόποι να ξέρεις τι αντικείμενο βρίσκεται πίσω από μια μεταβλητή (id) και το κατά πόσο μπορεί να αντιδράσει σε κάποια συγκεκριμένη μέθοδο. Η χρήση μεταβλητών (id) είναι σύνηθες φαινόμενο.
Λοιπόν, είδαμε το τρόπο σύνταξης του header αρχείου της κλάσης του Model μας. Τι γίνεται όμως με την υλοποίηση της. Πάμε να δούμε το implementation αρχείο .m:

CalculatorBrain.m

```
#import "CalculatorBrain.h"

@implementation CalculatorBrain

- (void)setOperand:(double)anOperand
{
    operand = anOperand;
}
- (double)performOperation:(NSString *)operation
{
    [operation sendMessage:argument];
    return aDouble;
}
@end
```

Τι σημαίνουν όλα αυτά?

Σ' αυτό το implementation είδαμε πως υλοποιούμε τις μεθόδους που είχαμε δηλώσει στο αρχείο header μας. Τι γίνεται όμως με τις private μεθόδους που μπορεί να θέλει κάποιος να υλοποιήσει? Λοιπόν, η δήλωση των private μεθόδων γίνεται κατευθείαν στο implementation αρχείο μας με τον εξής τρόπο:

```
@interface CalculatorBrain()
-(void) privateMethod:(double)a;
@end

@implementation CalculatorBrain
-(void) privateMethod:(double)a
{
}
@end
```

Τέλος με το Model. Ακολουθεί ο Controller.

Controller

CalculatorViewController.h

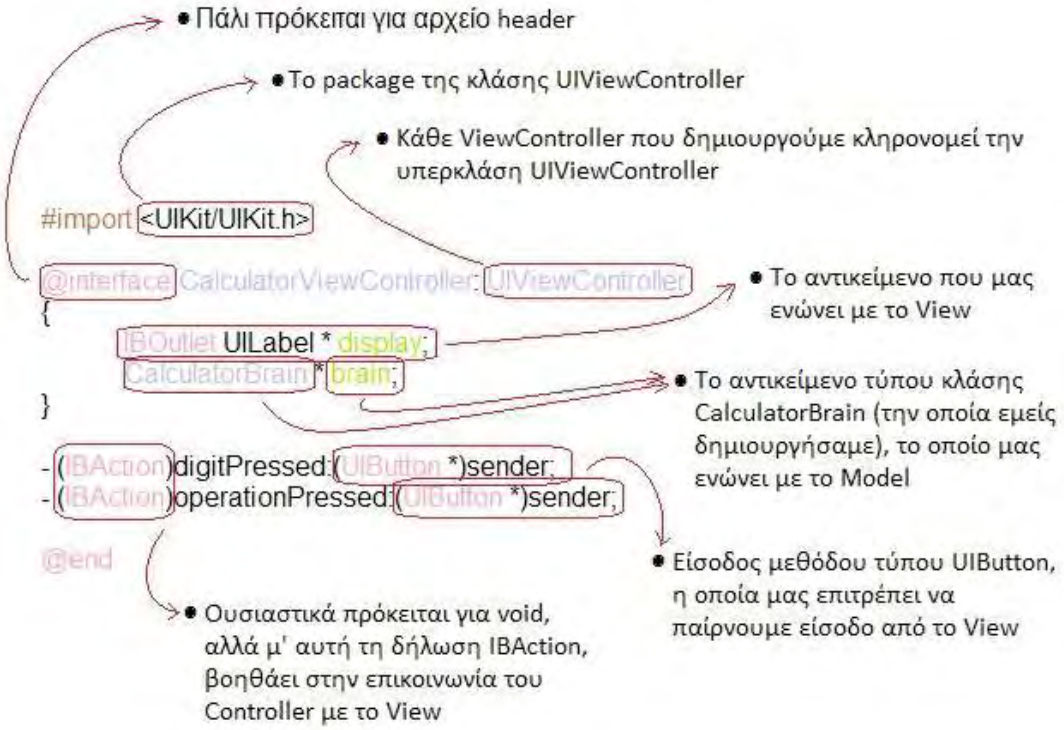
```
#import <UIKit/UIKit.h>

@interface CalculatorViewController: UIViewController
{
    IBOutlet UILabel * display;
    CalculatorBrain * brain;
}

- (IBAction)digitPressed:(UIButton *)sender;
- (IBAction)operationPressed:(UIButton *)sender;

@end
```

Τι είναι όμως όλα αυτά?



Το implementation του CalculatorViewController.m δε χρειάζεται να το δούμε, απλά να αναφέρουμε ότι σ' αυτό φροντίζουμε να καλέσουμε τις μεθόδους που έχουμε υλοποιήσει στο Model μας.

View

Το μόνο που υπολείπεται για την ολοκλήρωση της εφαρμογής, είναι το View. Το κομμάτι όμως αυτό δεν θα το υλοποιήσουμε με κώδικα αλλά με τη βοήθεια γραφικού περιβάλλοντος (κουμπία, drag and drop). Το γραφικό αυτό περιβάλλον που θα πρέπει να χρησιμοποιήσουμε δεν είναι άλλο από το Interface Builder.

CalculatorViewController.xib

Πέρα από τη σχεδίαση του UI μας φυσικά πρέπει να γίνουν και οι απαραίτητες διασυνδέσεις μεταξύ των κουμπιών-labels και των οντοτήτων UI στον Controller μας. Αυτό γίνεται πάλι στο Interface Builder με τη βοήθεια του παραθύρου με το τίτλο CalculatorViewController.xib που βρίσκεται κάτω δεξιά (βλέπε screenshot). Εδώ ο κύβος με το όνομα File's Owner αντιπροσωπεύει τον Controller μας. Οι διασυνδέσεις που πρέπει να γίνουν είναι οι εξής:

1. Με το δεξί κλικ του ποντικιού πατάμε και κρατάμε πάνω στο κάθε κουμπί ψηφίο του UI μας. Θα εμφανιστεί μια μπλε γραμμή την οποία τραβάμε προς το κύβο File's Owner και αφήνουμε. Πάνω από το κύβο θα εμφανιστούν οι επιλογές digitPressed και operationPressed. Επιλέγουμε digitPressed. Επαναλαμβάνουμε για κάθε ψηφίο.
2. Ακολουθούμε την ίδια διαδικασια με το 1 και για τα κουμπιά με τις πράξεις, με μόνη διαφορά την επιλογή του operationPressed. Επαναλαμβάνουμε για κάθε σύμβολο πράξης.
3. Κάνουμε το ίδιο και με το UILabel της οθόνης του Calculator μας, αυτή τη φορά πάνω από το κύβο δεν θα εμφανιστεί καμιά επιλογή. Επίσης για το UILabel, κάνουμε κάτι επιπλέον. Πατάμε αυτή τη φορά με δεξί κλικ πάνω στο κύβο File's Owner και τραβάμε προς το UILabel. Αφήνουμε και στις επιλογές που εμφανίζονται επιλέγουμε εδώ το display. Έτσι εγκαθιστάται 2-way επικοινωνια μεταξύ Controller και View.

Αυτό ήταν και η ολοκλήρωση του παραδείγματος μας. Είδαμε πως δημιουργούνται και αλληλεπιδρούν τα αντικείμενα, κλάσεις, μεταβλητές και μέθοδοι, πάντα μέσω του μοντέλου MVC. Δεν αποτελεί τεκμηριωμένη διαδικασία με όλα τα βήματα και τις λεπτομέρειες. Παραλείψαμε αρκετά, ίσως και βασικά στοιχεία τα οποία θα έπρεπε να κάνει κάποιος εάν ήθελε να τρέξει την εφαρμογή και να λειτουργήσει. Αυτό όμως δεν ήταν ο σκοπός μας, απλά θέλαμε να περιγράψουμε συνοπτικά τα βήματα που πρέπει να ακολουθηθούν, και το κάναμε!

Properties

Όπως σε κάθε αντικειμενοστραφής γλώσσα, μια σωστή προσέγγιση είναι οι μεταβλητές που δηλώνονται στο αρχειο header μιας κλάσης να προστατεύονται με δικαιοδοσία private. Από κει και πέρα όμως πώς θέτουμε σε αυτές και πως παίρνουμε από αυτές τη τιμή τους? Έστω μια μεταβλητή operand. Κανονικά θα έπρεπε να δημιουργήσουμε τις μεθόδους setOperand και operand για να θέτουμε και να διαβάζουμε την operand αντίστοιχα. Λοιπόν, στην Objective-C προσφέρεται η διευκόλυνση των properties. Αντί να δηλώνουμε και να υλοποιούμε τις μεθόδους αυτές, απλά μπορούμε να γράψουμε @property double operand στο αρχείο header και @synthesize operand στο implementation αρχείο και είμαστε έτοιμοι. Κοιτάξτε πως αντιστοιχούν οι δύο τρόποι:

1ος τρόπος

Header file:

```
@interface CalculatorBrain: NSObject
{
@private
        double operand;
}
- (double) operand;
- (void) setOperand:(double)anOperand;
@end
```

Implementation file:

```
@implementation CalculatorBrain
- (double) operand {
        return operand;
}
- (void) setOperand:(double)anOperand
{
        operand = anOperand;
}
@end
```

2ος τρόπος

Header file:

```
@interface CalculatorBrain: NSObject
{
@private
        double operand;
}
@property double operand;
@end
```

Implementation file:

```
@implementation CalculatorBrain
@synthesize operand;
@end
```

Αφού ακολουθηθεί ο τρόπος property - synthesize, μπορεί κάποιος πλέον να δημιουργεί ένα αντικείμενο της κλάσης CalculatorBrain *brain, και να διαβάζει και να γράφει την operand χρησιμοποιώντας τη τελεία:

```
brain.operand = 1.0;
double value = brain.operand;  // value = 1.0;
```

Τύποι δεδομένων

nil

Η τιμή ενός δείκτη προς αντικείμενο που δεν δείχνει πουθενά.

```
id obj = nil;
NSString *hello = nil;
```

Όπως το 0 για ένα απλό τύπο μεταβλητής (int, double κτλ). Συγκεκριμένα δεν είναι σαν 0, ένα αντικείμενο όταν δημιουργείται αρχικοποιεί αυτόματα όλες τις μεταβλητές του σε 0. Το ίδιο και δείκτες προς αντικείμενα όταν δηλώνονται αρχικοποιούνται σε nil. Στέλλοντας μήνυμα σε ένα nil αντικείμενο είναι εντάξει, δεν επιστρέφεται κάποιο σφάλμα. Απλά δεν γίνεται τίποτα. Αυτή είναι και η επιθυμητή λειτουργία για το nil.

NSString

Strings κωδικοποίησης Unicode. Χρησιμοποιούνται στο iOS αντι για το τύπο char * της C. Ένα NSString μπορεί να δημιουργηθεί χρησιμοποιώντας τη μορφή @”foo”. Ένα NSString δε μπορεί να τροποποιηθεί. Είναι μη τροποποιήσιμα. Γενικά αυτή η τεχνική είναι μια στρατηγική που ακολουθείται στο iOS για καλύτερη διαχείριση της μνήμης. Όταν δημιουργηθεί ένα NSString δε μπορεί πλέον να αλλάξει ή να αφαιρεθεί και να προσκολληθεί κάποιο γράμμα εκ των υστέρων. Είτε θα το δημιουργήσεις όπως θες, είτε θα επαναγράψεις πάνω απ’ αυτό. Αυτό γίνεται καλώντας κάποια από τις μεθόδους του.

```
[display setText:[[display text] stringByAppendingString:digit]];
display.text = [display.text stringByAppendingString:digit]; // same but with properties
display.text = [NSString stringWithFormat:@“%g”, brain.operand]; // class method
```

NSMutableString

Η τροποποιήσιμη έκδοση του NSString. Εδώ ένα NSMutableString μπορεί να αλλάξει ή να αφαιρεθεί και να προσκολληθεί κάποιο γράμμα εκ των υστέρων.

```
NSMutableString *mutString = [[NSMutableString alloc] initWithString:@“0.”];
[mutString appendString:digit];
```

NSNumber

Αντικείμενο βασισμένο σε απλούς τύπους int, float, double, BOOL, κτλ.

```
NSNumber *num = [NSNumber numberWithFloat:36.5];
float f = [num floatValue];
```

Προσφέρει ένα αριθμό χρήσιμων μεθόδων.

NSArray

Διατεταγμένη συλλογή(πίνακας) αντικειμένων. Μη τροποποιήσιμος άλλη μια φορά. Μερικές χρήσιμες μέθοδοι:

```
+ (id)arrayWithObjects:(id)firstObject, ...;
- (int)count;
- (id)objectAtIndex:(int)index;
- (void)makeObjectsPerformSelector:(SEL)aSelector;
- (NSArray *)sortedArrayUsingSelector:(SEL)aSelector;
- (id)lastObject; // returns nil if there are no objects in the array (convenient)
```

NSMutableArray

Τροποποιήσιμη έκδοση του NSArray.

```
- (void)addObject:(id)anObject;
- (void)insertObject:(id)anObject atIndex:(int)index;
- (void)removeObjectAtIndex:(int)index;
```

NSDictionary

Πίνακας κατακερμάτωσης(hash) αντικειμένων. Χειρίζεσαι τα αντικείμενα χρησιμοποιώντας κλειδιά. Μη τροποποιήσιμος κι αυτός. Για κλειδιά χρησιμοποιούνται αντικείμενα τύπου NSString και πρέπει να αντιδρούν στις μεθόδους (NSUInteger)hash και (BOOL)isEqual:(NSObject *)obj. Σημαντικές μέθοδοι:

```
- (int)count;
- (id)objectForKey:(id)key;
- (NSArray *)allKeys;
- (NSArray *)allValues;
```

NSMutableDictionary

Η τροποποιήσιμη έκδοση του NSDictionary.

```
- (void)setObject:(id)anObject forKey:(id)key;
- (void)removeObjectForKey:(id)key;
- (void)addEntriesFromDictionary:(NSDictionary *)otherDictionary;
```

NSSet

Αδιάτακτη συλλογή αντικειμένων, μη τροποποιήσιμη. Χρήσιμες μέθοδοι:

```
- (int)count;
- (BOOL)containsObject:(id)anObject;
- (id)anyObject;
- (void)makeObjectsPerformSelector:(SEL)aSelector;
- (id)member:(id)anObject; // uses isEqual: and returns a matching object (if any)
```

NSMutableSet

Η τροποποιήσιμη έκδοση του NSSet.

```
- (void)addObject:(id)anObject;
- (void)removeObject:(id)anObject;
- (void)unionSet:(NSSet *)otherSet;
- (void)minusSet:(NSSet *)otherSet;
- (void)intersectSet:(NSSet *)otherSet;
```

Διαχείριση μνήμης

Δημιουργία αντικειμένων

Τι χρειάζεται να κάνει κάποιος για να δημιουργήσει ένα αντικείμενο? Πολύ απλά πρέπει να δεσμεύσει μνήμη γι' αυτό από το heap (alloc), και έπειτα να το αρχικοποιήσει (init). Η διαδικασία αυτή γίνεται με τη χρήση του συνδυασμού alloc-init. Για κάθε τύπο αντικειμένου δεσμεύεται η απαιτούμενη μνήμη και γίνονται οι απαραίτητες ενέργειες για τη σωστή αρχικοποίηση του. Ένα αντικείμενο μπορεί να έχει περισσότερες από μια init:

```
NSString *myString = [[NSString alloc] initWithString:@“0.”];
NSString *myString = [[NSString alloc] initWithFormat:@“%d”, zero];
```

Και φυσικά υπάρχουν και ένα σωρό μέθοδοι που αρχικοποιούν ένα αντικείμενο και στο επιστρέφουν:

```
NSString *newDisplay = [display.text stringByAppendingString:digit];
NSArray *keys = [dictionary allKeys];
NSString *lowerString = [string lowercaseString];
NSNumber *n = [NSNumber numberWithFloat:42.0];
NSDate *date = [NSDate date]; // returns the date/time right now
```

Το ίδιο συμβαίνει και με την alloc. Δεν είναι η μόνη που μπορεί να δεσμεύσει μνήμη. Το ίδιο μπορούν να κάνουν και οι εντολές copy και new, με την copy να είναι ιδιαίτερα δημοφιλής όταν πρόκειται για NSStrings.
Τι γίνεται όμως με όλη αυτή τη δέσμευση μνήμης. Ποιος είναι υπεύθυνος για τη σωστή και αποδοτική δέσμευση-αποδέσμευση της? Λοιπόν στη διάσωση έρχεται μια λειτουργία της Objective-C που ονομάζεται Reference Counting.

Reference Counting

Δυστυχώς ή ευτυχώς η Objective-C δε διαθέτει κάποια διαδικασία Garbage Collector που μπορεί να συναντούμε σε άλλες γλώσσες όπως στη Java. Κάθε φορά που δεσμεύεται μνήμη για ένα αντικείμενο αυτός που κάλεσε μια από τις εντολές alloc, new ή copy, “χρεώνεται” το αντικείμενο και είναι υπεύθυνος αυτός για την αποδέσμευσή του.

```
CalculatroBrain *brain = [[CalculatroBrain alloc] init];
```

Αυτόματα κατά τη δημιουργία του αντικειμένου θεωρείται ότι ο αριθμός αναφορών προς το αντικείμενο αυτό είναι 1. Από κει και πέρα όποιος ενδιαφέρεται για το εν λόγω αντικείμενο και θέλει να φροντίσει ότι δε θα αποδεσμευτεί στη συνέχεια μπορεί να κάνει retain το αντικείμενο.

```
[brain retain];
```

Αυτόματα ο αριθμός αναφορών στο αντικείμενο αυτό θα αυξηθεί κατά ένα . Αντίστοιχα εάν κάποιος δε χρειάζεται ένα αντικείμενο μπορεί να δηλώσει το μη ενδιαφέρον του καλώντας release.

```
[brain release];
```

Τέλος υπάρχει και η δυνατότητα για δημιουργία ενός αντικειμένου και δήλωση για επιθυμία προσωρινής ιδιοκτησίας μέχρις ότου κάποιος άλλος να το κάνει retain και να το “χρεωθεί”. Αυτό γίνεται με τη μέθοδο autorelease. Πρόκειται για μια συνηθισμένη διαδικασία στην Objective-C για μεταβίβαση αντικειμένων μεταξύ ιδιοκτητών.

```
[brain autorelease];
```

Όταν λοιπόν ένα αντικείμενο γίνει release από όλους τους ιδιοκτήτες του και φτάσει το reference counter του στο 0, τότε αυτό καταστρέφεται και ελευθερώνεται η μνήμη που απασχολούσε.

Κεφάλαιο 3 Αλγόριθμος KioustelidisQuadratic

3.1 Περιγραφή

Ο KioustelidisQuadratic αποτελεί ένα αλγόριθμο για τη εύρεση φραγμάτων των θετικών ριζών ενός πολυωνύμου. Πρόκειται για την μετεξέλιξη του απλού αλγορίθμου Kioustelidis, ο οποίος στηρίχτηκε στο θεώρημα Akritas-Strzebonski-Vigklas, 2006, παρουσιάζοντας τη δική του “leading-coefficient” υλοποίηση.

Ένας αλγόριθμος για υπολογισμό τέτοιων φραγμάτων δημιουργήθηκε για τις ανάγκες μιας άλλης διαδικασίας, της απομόνωσης πραγματικών ριζών με τον αλγόριθμο των συνεχών κλασμάτων που ακούει στο όνομα VAS (Vincent - Akritas - Strzebonski). Εδώ φυσικά να πούμε ότι το όνομα Akritas που αναφέρω για 2η φορά, δεν είναι άλλος από τον κύριο Αλκιβιάδη Ακρίτα, ο καθηγητής ο οποίος μου έδωσε την ευκαιρία να ασχοληθώ με την υλοποίηση του αλγορίθμου KioustelidisQuadratic ως θέμα για τη διπλωματική αυτή.

Ο υπολογισμός, λοιπόν, των κάτω και άνω φραγμάτων των τιμών των θετικών ριζών ενός πολυωνύμου είναι μια πολύ σημαντική πρώτη διαδικασία για τη μέθοδο απομόνωσης των πραγματικών ριζών VAS ενός πολυωνύμου. Η αποδοτικότητα της μεθόδου VAS σχετίζεται άμεσα με το πόσο καλός θα είναι ο υπολογισμός αυτός [5].

Τέλος να αναφέρουμε ότι η υλοποίηση τέτοιων αλγορίθμων σε φορητές συσκευές αποτελεί ότι πιο ιδανικό από άποψη απαίτησης μνήμης. Οι αλγόριθμοι αυτού του τύπου δεν παρουσιάζουν καμιά ανάγκη για αποθήκευση μεγάλου όγκου δεδομένων παρά μόνο ενός πίνακα για συγκράτηση των συντελεστών του πολυωνύμου και κάποιων βοηθητικών πινάκων πάλι μεγέθους ανάλογου της τάξης της πολυωνυμικής εξίσωσης. Από κει και πέρα είναι στο χέρι της υπολογιστικής ισχύς μιας φορητής συσκευής για το πόσο γρήγορα θα επιστραφεί το αποτέλεσμα στην έξοδο. Και ξέροντας ότι οι τελευταίες συσκευές iOS τρέχουν σε ARM επεξεργαστή με ταχύτητα 1GHz, είμαστε σίγουροι ότι το αποτέλεσμα θα επιστρέφεται σε ικανοποιητικό χρόνο, για χαμηλής τάξης πολυώνυμα δε σε αμελητέο!

Είσοδος: πολυωνυμική εξίσωση p(x)=0, με ακέραιους συντελεστές, χωρίς πολλαπλές ρίζες
Έξοδος: Οι τιμές ub και lb, άνω και κάτω φράγμα των τιμών των θετικών ριζών του πολυωνύμου p(x).

Άνω Όριο ub:

Κάθε αρνητικός συντελεστής a_i ζευγαρώνεται με το κάθε θετικό συντελεστή a_j του πολυωνύμου διαιρεμένο με 2^(i-j), όπου i-j η απόσταση μεταξύ των θέσεων των δύο συντελεστών. Επιλέγεται το max-min αποτέλεσμα.

$$ub_{KQ} = \max_{\{a_i<0\}} \min_{\{a_j>0 : j>i\}} \sqrt[j-i]{-\frac{a_i}{\frac{a_j}{2^{j-i}}}}$$

Για το κάτω Όριο Ib γίνονται παρόμοιοι υπολογισμοί επεξεργάζοντας διαφορετικά το πολυώνυμο.

3.2 Υλοποίηση σε Mathematica

```
KQ[p_] := Module[{cl = CoefficientList[p, Variables[p]], len,
   m, n, q = -1, tempmin, tempmax = 0, timestart = TimeUsed[]},
  If[Last[cl] < 0, cl = -cl];
  len = Length[cl];

  If[len <= 1, Return[0]];

  Do[
   If[cl[[m]] < 0,
    (*then*)
    tempmin = ∞;
    Do[
     If[cl[[n]] > 0,
      q = N[(-cl[[m]]/cl[[n]])^(1/(n - m))];
      If[q < tempmin, tempmin = q]
     ],
     {n, len, m + 1, -1}
    ];
    If[tempmax < tempmin, tempmax = tempmin];
   ],
   {m, len - 1, 1, -1}
  ];
  {2 * tempmax, TimeUsed[] - timestart}]
```

3.3 Η δική μου υλοποίηση

Upper Bound

```
- (double)getUB {
        if(len<=1) {
                ub = 0;
        }
        else {
                for(m=len-1; m>=1; m--) {
                        if(clu[m-1]<0) {
                                tempmin = 999999999*1.0;

                                for(n=len; n>=m+1; n--) {
                                        if(clu[n-1]>0) {
                                            temp1=-clu[m-1]*1.0/clu[n-1]*1.0;
                                            temp2=1.0/(n-m);
                                            q=powf(temp1,temp2);
                                            if(tempmin>q) {
                                                tempmin=q;
                                             }
                                        }
                                }
                                if(tempmax<tempmin) {
                                        tempmax=tempmin;
                                }
                        }
                }
                ub=ceil((65*1.0/64*1.0)*tempmax);
        }
        return ub;
}
```

Lower Bound

```
- (double)getLB {
        temp1=0.0;
        temp2=0.0;
        q=-1.0;
        tempmax=0.0;
        tempmin=0.0;
        len=cmax+1;

        if(len<=1) {
                lb = 0;
        }
        else {
                for(m=2; m<=len; m++) {
                        if(cll[m-1]<0) {
                                tempmin = 999999999*1.0;

                                for(n=1; n<=m-1; n++) {
                                        if(cll[n-1]>0) {
                                                temp1=-cll[m-1]*1.0/cll[n-1]*1.0;
                                                temp2=1.0/(m-n);
                                                q=powf(temp1,temp2);
                                                if(tempmin>q) {
                                                        tempmin=q;
                                                }
                                        }
                                }
                                if(tempmax<tempmin) {
                                        tempmax=tempmin;
                                }
                        }
                }
                lb=powf(2,-ceil(log2f(tempmax)));
        }
        return lb;
}
```

Κεφάλαιο 4 Η εφαρμογή KioustelidisQ

4.1 Εύρεση Upper και Lower Bound

Model

KQBrain.h

```
//
//  KQBrain.h
//  KioustelidisQ
//
//  Created by Peter on 2/26/11.
//  Copyright 2011 __MyCompanyName__. All rights reserved.
//

#import <Foundation/Foundation.h>

@interface KQBrain : NSObject {
	int *cl,*clu,*cll;
	int i,flag,len,cmax,ctemp,y,m,coeff,n,powCoef;
	double temp1,temp2,q,tempmax,tempmin,ub,lb;
	char chr,chrr;
}

@property (readonly) int *cl;
@property (readonly) int cmax;

- (void)getCoeffs: (NSString *)equation;
- (double)getUB;
- (double)getLB;

@end
```

KQBrain.m

```
//
//  KQBrain.m
//  KioustelidisQ
//
//  Created by Peter on 2/26/11.
//  Copyright 2011 __MyCompanyName__. All rights reserved.
//
```

```
#import "KQBrain.h"

@implementation KQBrain

@synthesize cl;
@synthesize cmax;

- (void)brainInit {
	flag=1;
	m=-1;
	q=-1.0;
	i=0;len=0;cmax=0;ctemp=0;y=0;
	coeff=0;n=0;temp1=0.0;temp2=0.0;powCoef=0;
	tempmax=0.0;
	tempmin=0.0;ub=0.0;lb=0.0;
}

- (void)getCoeffs: (NSString *)equation {

	self.brainInit;

	NSString *tempString1 = [[NSString alloc]
initWithFormat:equation];
	NSString *tempString = [[tempString1
componentsSeparatedByCharactersInSet: [NSCharacterSet
whitespaceCharacterSet]] componentsJoinedByString: @""];
	NSCharacterSet *doNotWant = [NSCharacterSet
characterSetWithCharactersInString:@"~!@#$%&*()"];
	tempString = [[tempString componentsSeparatedByCharactersInSet:
doNotWant] componentsJoinedByString: @""];

	len= [tempString length];

	char tempMatrix[len];

	for(i=0,y=i+1; (i<len); i++,y++)
	{
		chr=(char )[tempString characterAtIndex:i];

		if(chr =='^')
		{
			chrr=(char)[tempString characterAtIndex:y];

			flag=0;
			ctemp= chrr-'0';
```

```
                if(cmax<ctemp)
                {
                        cmax = ctemp;
                }
            }
            else if(flag==1 && chr=='x')
            {
                cmax=1;
                flag=0;
            }
        }

        cl=(int *)malloc((cmax+1)*sizeof(int));
        clu=(int *)malloc((cmax+1)*sizeof(int));
        cll=(int *)malloc((cmax+1)*sizeof(int));

        for(i=0;i<=cmax;i++)
        {
            cl[i]=0;
        }

        i=0;

        flag=1;

        while(flag)
        {
            if(i<len)
            {
                chr=(char )[tempString characterAtIndex:i];
            }
            else if(i==len && chr=='x' )
            {
                break;
            }
            else
            {
                flag=0;
            }
            if(chr=='x' || i==len || ((chr=='+' || chr=='-') && i>0
&& m!=-1))
            {
                if(chr=='x' && m==-1)
                {
```

```
                        coeff=1;
                }
                else if(chr=='x' && m==0 && tempMatrix[0]=='+')
                {
                        coeff=1;
                }
                else if(chr=='x' && m==0 && tempMatrix[0]=='-')
                {
                        coeff=-1;
                }
                else
                {
                        for(y=0;m>=0;m--,y++)
                        {
                                if(tempMatrix[m]!='+' &&
tempMatrix[m]!='-')
                                {
                                        coeff= coeff + (tempMatrix[m] -
'0')* pow(10,y);
                                }
                                else if(tempMatrix[m]=='-')

                                {
                                        coeff=-coeff;
                                }
                        }
                }

                if(chr=='x' && i<len-1 && (char )[tempString
characterAtIndex:i+1]=='^')
                {
                        powCoef=(char )[tempString
characterAtIndex:i+2] -'0';
                        i=i+2;
                        cl[powCoef]=coeff;
                        coeff=0;
                }
                else if(chr=='x')
                {
                        cl[1]=coeff;
                        coeff=0;
                }
                else
                {
                        cl[0]=coeff;
```

```
                        coeff=0;
                        i--;
                }
                m=-1;
        }
        else
        {
                m++;
                tempMatrix[m]=chr;
        }

        i++;
}

flag=1;

if(cl[cmax]<0)
{
        for(i=0; i<=cmax; i++)
        {
                clu[i]=-cl[i];
        }
}
else
{
        for(i=0; i<=cmax; i++)
        {
                clu[i]=cl[i];
        }
}

if(cl[0]<0)
{
        for(i=0; i<=cmax; i++)
        {
                cll[i]=-cl[i];
        }
}
else
{
        for(i=0; i<=cmax; i++)
        {
                cll[i]=cl[i];
        }
}
```

```
	len = cmax+1;

}
- (double)getUB {
	if(len<=1)
	{
		ub = 0;
	}
	else
	{
		for(m=len-1; m>=1; m--)
		{
			if(clu[m-1]<0)
			{
				tempmin = 999999999*1.0;

				for(n=len; n>=m+1; n--)
				{
					if(clu[n-1]>0)
					{
						temp1=-clu[m-1]*1.0/clu[n-1]*1.0;
						temp2=1.0/(n-m);
						q=powf(temp1,temp2);
						if(tempmin>q)
						{
							tempmin=q;
						}
					}
				}
				if(tempmax<tempmin)
				{
					tempmax=tempmin;
				}
			}
		}
		ub=ceil((65*1.0/64*1.0)*tempmax);
	}
	return ub;
}
- (double)getLB {
	temp1=0.0;
	temp2=0.0;
	q=-1.0;
	tempmax=0.0;
```

```
        tempmin=0.0;
        len=cmax+1;

        if(len<=1)
        {
              lb = 0;
        }
        else
        {
              for(m=2; m<=len; m++)
              {
                    if(cll[m-1]<0)
                    {
                          tempmin = 999999999*1.0;

                          for(n=1; n<=m-1; n++)
                          {
                                if(cll[n-1]>0)
                                {
                                      temp1=-cll[m-1]*1.0/cll[n-1]*1.0;
                                      temp2=1.0/(m-n);
                                      q=powf(temp1,temp2);
                                      if(tempmin>q)
                                      {
                                            tempmin=q;
                                      }
                                }
                          }
                          if(tempmax<tempmin)
                          {
                                tempmax=tempmin;
                          }
                    }
              }
              lb=powf(2,-ceil(log2f(tempmax)));
        }
        return lb;
}

@end
```

Ο εγκέφαλος της εφαρμογής μας, ο κώδικας που θα αποτελέσει το Model μας. Πέρα από όλες τις απαραίτητες μεταβλητές που χρειάζονται, υπάρχουν οι μεθόδοι getUB, getLB και getCoeffsEquation. Οι 2 πρώτες αποτελούν την υλοποίηση του αλγορίθμου KioustelidisQ, και η τελευταία είναι υπεύθυνη για τη μετατροπή της πολυωνυμικής εξίσωσης που

παίρνουμε ως είσοδο, σε πίνακα με τους συντελεστές της.
Συγκεκριμένα η getCoeffsEquation, για κάθε πολυώνυμο εκφρασμένο σε δυνάμεις του x, δημιουργεί τον αντίστοιχο πίνακα με τις θέσεις 0,1,2,3...N (όπου N η μεγαλύτερη δύναμη στην οποία συναντάμε το x να είναι υψωμένο) στις οποίες θέσεις φιλοξενούνται η σταθερά, ο συντελεστής του x υψωμένο στη 1η δύναμη, ο συντελεστής του x υψωμένο στη 2η δύναμη, ο συντελεστής του x υψωμένο στη 3η δύναμη κ.ο.κ.
Για τη διαδικασία της μετατροπής αυτής διαπερνάμε το αλφαριθμητικό που συγκρατεί τη πολυωνυμική εξίσωση πολλαπλές φορές μέχρι να πάρουμε όλες τις απαραίτητες πληροφορίες.

Controller #1

KioustelidisQViewController.h

```
//
//  KioustelidisQViewController.h
//  KioustelidisQ
//
//  Created by Peter on 2/26/11.
//  Copyright __MyCompanyName__ 2011. All rights reserved.
//

#import <UIKit/UIKit.h>
#import "KQBrain.h"
#import <AVFoundation/AVAudioPlayer.h>
#import <AVFoundation/AVFoundation.h>

@interface KioustelidisQViewController : UIViewController
<AVAudioPlayerDelegate> {
    IBOutlet UILabel *display;
    IBOutlet UITextField *txtEquation;
    KQBrain *brain;
    double ub;
    double lb;
    AVAudioPlayer *player;
}

@property (nonatomic,retain) IBOutlet UITextField *txtEquation;
@property (nonatomic,retain) IBOutlet  UILabel *display;
@property (nonatomic,retain) AVAudioPlayer *player;

- (IBAction) computeBounds;
- (IBAction) makeKeyboardGoAway;
- (IBAction) graph;
- (IBAction) instructions;
- (IBAction) play;
```

```
@end
```

KioustelidisQViewController.m

```
//
//  KioustelidisQViewController.m
//  KioustelidisQ
//
//  Created by Peter on 2/26/11.
//  Copyright __MyCompanyName__ 2011. All rights reserved.
//

#import "KioustelidisQViewController.h"
#import "GraphViewController.h"

@implementation KioustelidisQViewController

@synthesize txtEquation;
@synthesize display;
@synthesize player;

- (IBAction) play {
    [self.player play];
}

- (KQBrain *) brain {
    if (!brain) {
      brain = [[KQBrain alloc] init];
    }
    return brain;
}

- (IBAction) instructions {
    NSString *instructionsString = @"Insert an equation in the
textfield and press either BoundIt to get Upper and Lower Bounds, or
press Graph to plot its graph.\nEquation example: 3x^2+2x^3+x-7\nNo
need to use the * sign for multiplication, no need to enter the
powers of x in a specific order.";

    UIAlertView *alert = [[UIAlertView alloc]
initWithTitle:@"Instructions"

    message:instructionsString

delegate:nil
```

```
cancelButtonTitle:@"Done"

otherButtonTitles:nil];
    [alert show];
    [alert release];
}

- (IBAction) graph {
    [[self brain] getCoeffs:[txtEquation text]];
    GraphViewController *gvc = [[GraphViewController alloc] init];
    gvc.expression=brain.cl;
    gvc.size=brain.cmax+1;
    //gvc.scale = (float)self.view.contentScaleFactor;
    gvc.scale=25;
    gvc.title=[txtEquation text];
    [self.navigationController pushViewController:gvc animated:YES];
    [gvc release];
}

- (IBAction) computeBounds {
    [txtEquation resignFirstResponder];

    [[self brain] getCoeffs:[txtEquation text]];
    ub = [[self brain] getUB];
    lb = [[self brain] getLB];

    NSString *msg=[[NSString alloc] initWithFormat:@"Upper bound:
%f\nLower bound: %f",ub,lb];

    display.lineBreakMode = UILineBreakModeWordWrap;
    display.numberOfLines = 0;
    display.textAlignment = UITextAlignmentRight;

    [display setText:msg];
    [msg release];
}

- (IBAction) makeKeyboardGoAway
{
    [txtEquation resignFirstResponder];
}

- (BOOL)textFieldShouldReturn:(UITextField *)textField
{
    [textField  resignFirstResponder];
```

```
    return YES;
}

-(BOOL)shouldAutorotateToInterfaceOrientation:
(UIInterfaceOrientation)interfaceOrientation
{
    return YES;
}

// Implement viewDidLoad to do additional setup after loading the
view, typically from a nib.
- (void)viewDidLoad {
    NSString *soundPath = [[NSBundle mainBundle]
pathForResource:@"sound" ofType:@"wav"];
    self.player =[[AVAudioPlayer alloc] initWithContentsOfURL:[NSURL
fileURLWithPath:soundPath] error:NULL];
    [player setDelegate: self];
    [player prepareToPlay];
    [super viewDidLoad];
}

- (void)didReceiveMemoryWarning {
    // Releases the view if it doesn't have a superview.
    [super didReceiveMemoryWarning];

    // Release any cached data, images, etc that aren't in use.
}

- (void)viewDidUnload {
    // Release any retained subviews of the main view.
    // e.g. self.myOutlet = nil;
}


- (void)dealloc {
    [super dealloc];
}

@end
```

Ο Controller της εφαρμογής μας, υπεύθυνος να μεσολαβήσει μεταξύ του View (χρήστης) και του Model (KQBrain).

```
IBOutlet UITextField *txtEquation;
IBOutlet UILabel *display;
```

Τα αντικείμενα του γραφικού περιβάλλοντος της εφαρμογής μέσω των οποίων θα πάρουμε άλλα και θα δώσουμε δεδομένα. Συγκεκριμένα το αντικείμενο txtEquation θα αναλάβει να διαβάσει από το View την πολυωνυμική εξίσωση που δόθηκε ως είσοδος ενώ με το display θα οδηγήσει στο View το αποτέλεσμα με τις τιμές των Upper και Lower bound.

```
- (IBAction) computeBounds;
- (IBAction) instructions;
- (IBAction) graph;
- (IBAction) play;
- (IBAction) makeKeyboardGoAway;
```

Από μεθόδους τώρα, έχουμε την computeBounds η οποία θα δώσει το έναυσμα της όλης διαδικασίας υπολογισμού των Upper και Lower bound, την instructions η οποία απλά εμφανίζει ένα μήνυμα με οδηγίες για το πως πρέπει να δίνεται η είσοδος, η graph η οποία θα προκαλέσει το σχεδιασμό της γραφικής παράστασης της πολυωνυμικής εξίσωσης της εισόδου, την play για αναπαραγωγή ήχου κατα το πάτημα των κουμπιών BoundIt και Graph και τέλος μια βοηθητική μέθοδο για απόσυρση του πληκτρολογίου μετά την εισαγωγή της εξίσωσής μας.
Τέλος αξίζει να αναφέρουμε και τη μέθοδο shouldAutorotateToInterfaceOrientation, η οποία θα επιτρέψει την περιστροφή του γραφικού μας περιβάλλοντος όταν περιστραφεί και η συσκευή στην οποία θα τρέχει.

AppDelegate

KioustelidisQAppDelegate.h

```
//
//  KioustelidisQAppDelegate.h
//  KioustelidisQ
//
//  Created by Peter on 2/26/11.
//  Copyright __MyCompanyName__ 2011. All rights reserved.
//

#import <UIKit/UIKit.h>

@interface KioustelidisQAppDelegate : NSObject
<UIApplicationDelegate> {
    UIWindow *window;
```

```
}

@property (nonatomic, retain) IBOutlet UIWindow *window;

@end
```

KioustelidisQAppDelegate.m

```
//
//  KioustelidisQAppDelegate.m
//  KioustelidisQ
//
//  Created by Peter on 2/26/11.
//  Copyright __MyCompanyName__ 2011. All rights reserved.
//

#import "KioustelidisQAppDelegate.h"
#import "KioustelidisQViewController.h"

@implementation KioustelidisQAppDelegate

@synthesize window;

- (void)applicationDidFinishLaunching:(UIApplication *)application {

    // Override point for customization after application launch
    UINavigationController *navcon = [[UINavigationController alloc]
init];
    KioustelidisQViewController *lvc = [[KioustelidisQViewController
alloc] init];
    lvc.title=@"KioustelidisQuadratic";
    [navcon pushViewController:lvc animated:NO];
    [lvc release];
    [window addSubview:navcon.view];
    [window makeKeyAndVisible];
}
/*
- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
    // Override point for customization after application launch.
    UINavigationController *navcon = [[UINavigationController alloc]
init];
    KioustelidisQViewController *qvc = [[KioustelidisQViewController
alloc] init];
    qvc.title=@"KioustelidisQuadratic";
    [navcon pushViewController:qvc animated:NO];
    [qvc release];
```

```
    [window addSubview:navcon.view];
    [window makeKeyAndVisible];

    return YES;
}
*/
- (void)dealloc {
    [window release];
    [super dealloc];
}
@end
```

Λόγω της ύπαρξης περισσότερων από ένα View και αντίστοιχα Controller, φροντίσαμε για τη δημιουργία ενός NavigationController, του navcon. Είναι υπεύθυνος να φορτώνει σε πρώτη φάση το View με το πεδίο εισαγωγής της εξίσωσης και τις διάφορες επιλογές για υπολογισμό των φραγμάτων ή τη σχεδίαση της γραφικής παράστασης και στη συνέχεια ενεργεί ανάλογα με το τι επιλέξουμε. Για τον υπολογισμο των φραγμάτων, εμφανίζονται στο ίδιο View το αποτέλεσμα με τις τιμές του, αλλά για τη σχεδίαση γραφικής παράστασης θα δημιουργηθεί και θα φορτωθεί καινούριο View με τον αντίστοιχο Controller. Στο καινούριο View λοιπόν θα εμφανίζεται η γραφική παράσταση σε πλήρη οθόνη με τις επιλογές για ZoomIn και ZoomOut.

Πέρα λοιπόν από τον κώδικα για τον αλγόριθμό του KioustelidisQuadratic στα πλαίσια της εφαρμογής που αναπτύξαμε, προσθέσαμε και το κώδικα για τη δυνατότητα σχεδίασης της γραφικής μας παράστασης.

4.2 Γραφική Παράσταση

Controller #2

GraphViewController.h

```
//
//  GraphViewController.h
//  Graph
//
//  Created by Peter on 2/12/11.
//  Copyright __MyCompanyName__ 2011. All rights reserved.
//

#import <UIKit/UIKit.h>
#import "GraphView.h"

@interface GraphViewController : UIViewController <GraphViewDelegate>
{
    GraphView *graphView;
    int *expression;
```

```
    int size;
    float scale;
}

@property (retain) IBOutlet GraphView *graphView;
@property int *expression;
@property float scale;
@property int size;

- (IBAction)zoomPressed:(UIButton *)sender;

@end
```

GraphViewController.m

```
//
//  GraphViewController.m
//  Graph
//
//  Created by Peter on 2/12/11.
//  Copyright __MyCompanyName__ 2011. All rights reserved.
//

#import "GraphViewController.h"

@implementation GraphViewController

@synthesize graphView;
@synthesize expression;
@synthesize scale;
@synthesize size;

- (float) eval:(float)x {
    int i;
    float y=0;

    for(i=0; i<size; i++)
    {
      y = y+self.expression[i]*powf(x,i);
      //NSLog(@"y=%f", y);
    }

    return y;
}

- (void) updateUI
{
```

```
    [self.graphView setNeedsDisplay];
}

- (float)getY:(GraphView *)requestor forX:(float)anX {
    float y = 0;

    if (requestor == self.graphView) {
      y = [self eval:anX];
    }
    return y;
}
- (float)getScale:(GraphView *)requestor {
    float theScale =0;

    if (requestor == self.graphView) {
      theScale = self.scale;
    }
    return theScale;
}

- (IBAction)zoomPressed:(UIButton *)sender {
    if([[sender.titleLabel text] isEqualToString:@"ZoomIn"])
      self.scale = self.scale+10;
    else if([[sender.titleLabel text] isEqualToString:@"ZoomOut"])
      self.scale = self.scale-10;
    [self updateUI];
}

-(BOOL)shouldAutorotateToInterfaceOrientation:
(UIInterfaceOrientation)interfaceOrientation
{
    return YES;
}

/*
// The designated initializer. Override to perform setup that is
required before the view is loaded.
- (id)initWithNibName:(NSString *)nibNameOrNil bundle:(NSBundle
*)nibBundleOrNil {
    if (self = [super initWithNibName:nibNameOrNil
bundle:nibBundleOrNil]) {
        // Custom initialization
    }
    return self;
}
```

```
*/

/*
// Implement loadView to create a view hierarchy programmatically,
without using a nib.
- (void)loadView {
}
*/


// Implement viewDidLoad to do additional setup after loading the
view, typically from a nib.
- (void)viewDidLoad {
    [super viewDidLoad];
    self.graphView.delegate=self;
    [self updateUI];
}

- (void)releaseOutlets
{
    self.graphView=nil;
}

/*
// Override to allow orientations other than the default portrait
orientation.
-
(BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)
interfaceOrientation {
    // Return YES for supported orientations
    return (interfaceOrientation == UIInterfaceOrientationPortrait);
}
*/

- (void)didReceiveMemoryWarning {
    // Releases the view if it doesn't have a superview.
    [super didReceiveMemoryWarning];

    // Release any cached data, images, etc that aren't in use.
}

- (void)viewDidUnload {
    // Release any retained subviews of the main view.
    [self releaseOutlets];
}
```

```
- (void)dealloc {
    [super dealloc];
}

@end
```

Ο Controller λοιπόν θα αναλάβει το δύσκολο έργο για το υπολογισμό των συντεταγμένων κάθε σημείου από το οποίο θα περνάει η γραφική παράσταση. Και για αυτό το Controller, Model αποτελεί το KQBrain. Όταν πατηθεί το κουμπί Graph και ξεκινήσει η διαδικασία δημιουργίας του καινούριου GraphView από τον GraphViewController, τα απαραίτητα δεδομένα θα φροντίσει να τα πάρει μέσω των properties cl και cmax του KQBrain. Ο κώδικας που φροντίζει για αυτό βρίσκεται στη μέθοδο graph KioustelidisQViewController και πρόκειται για το εξής σημείο:

```
gvc.expression=brain.cl;
gvc.size=brain.cmax+1;
```

Όπως καταλαβαίνετε, το cl αποτελεί τον πίνακα με τους συντελεστές της εξίσωσης μας επομένως την εξίσωσή μας, και η cmax τον ακέραιο που εκφράζει τη μεγαλύτερη δύναμη του x στο πολυώνυμό μας.

GraphView.h

```
//
//  GraphView.h
//  Graph
//
//  Created by Peter on 2/12/11.
//  Copyright 2011 __MyCompanyName__. All rights reserved.
//

#import <UIKit/UIKit.h>

@class GraphView;

@protocol GraphViewDelegate
- (float)getY:(GraphView *)requestor forX:(float)anX;
- (float)getScale:(GraphView *)requestor;
@end

@interface GraphView : UIView {
    id <GraphViewDelegate> delegate;
}
```

```
@property (assign) id <GraphViewDelegate> delegate;

@end
```

GraphView.m

```
//
//  GraphView.m
//  Graph
//
//  Created by Peter on 2/12/11.
//  Copyright 2011 __MyCompanyName__. All rights reserved.
//

#import "GraphView.h"
#import "AxesDrawer.h"

@implementation GraphView

@synthesize delegate;

- (id)initWithFrame:(CGRect)frame {
    if (self = [super initWithFrame:frame]) {
        // Initialization code
    }
    return self;
}

- (void)drawRect:(CGRect)rect {

    CGPoint midPoint;
    midPoint.x = self.bounds.origin.x + self.bounds.size.width/2;
    midPoint.y = self.bounds.origin.y + self.bounds.size.height/2;

    float scale = [self.delegate getScale:self];

    CGContextRef context = UIGraphicsGetCurrentContext();
    [AxesDrawer drawAxesInRect:self.bounds originAtPoint:midPoint
scale:scale];

    CGContextBeginPath(context);

    CGFloat x = -midPoint.x;
    CGFloat y = (-[self.delegate getY:self forX:(x/scale)])*scale;
```

```
    CGContextMoveToPoint(context, x+midPoint.x, y+midPoint.y);

    [[UIColor colorWithRed:0.2431 green:0.3843 blue:0.6549 alpha:1]
setStroke];
    //[[UIColor colorWithRed:0.3686 green:0.4313 blue:0.5137 alpha:1]
setStroke];

    CGContextSetLineWidth(context, 4.0);

    for(; x<self.bounds.size.width; x++)
    {
      y = (-[self.delegate getY:self forX:(x/scale)])*scale;
      CGContextAddLineToPoint(context, x+midPoint.x, y+midPoint.y);
    }

    CGContextStrokePath(context);
}

- (void)dealloc {
    [super dealloc];
}

@end
```

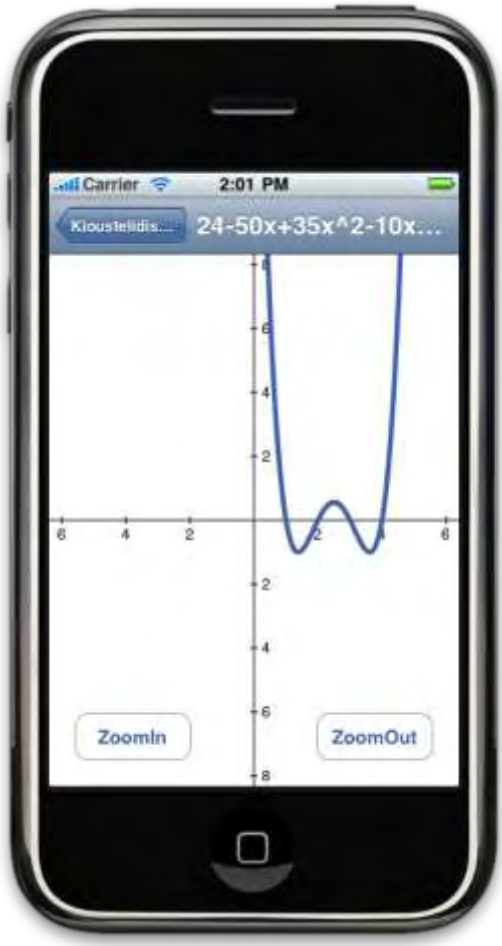
Για το σχεδιασμό της γραφικής παράστασης το κύριο ρόλο αναλαμβάνει η βιβλιοθήκη CG η οποία με την εντολή CGContextAddLineToPoint θα ενώσουμε διαδοχικά όλα τα σημεία της γραφικής παράστασης. Να αναφέρουμε ότι η σχεδίαση γίνεται σε επίπεδο pixel και για το λόγο αυτό χρειάστηκε να γνωρίζουμε το πόσα pixel υπάρχουν σε κάθε σημείο (point) στην οθόνη της συσκευής. Μέχρι και το iPhone 3GS, η τιμή αυτή ήταν 1.0f, μέχρι που η Apple ξεκίνησε να προσφέρει τις οθόνες Retina στις iΣυσκευές της (iPhone4 και iPad). Η οθόνες Retina λοιπόν υποστηρίζουν τη διπλάσια ποσότητα pixel σε κάθε σημείο (point). Για την ανάγκη αυτή να ξέρει ο κάθε προγραμματιστής σε ποια οθόνη θα φιλοξενείται η εφαρμογή του, δημιουργήθηκε η μέθοδος συστήματος contentScaleFactor, η οποία για κάποιο αντικείμενο view επιστρέφει ακριβώς την πληροφορία αυτή. Δυστυχώς η μέθοδος αυτή ξεκίνησε να προσφέρεται απο το Xcode 3.2 μετά ενώ εμείς ξεκινήσαμε και τελιώσαμε τη διπλώματική αυτή σε Xcode 3.1.
Για το σχεδιασμό των αξόνων χρησιμοποιήσαμε 2 έτοιμα αρχεία κώδικα τα οποία προμηθευτήκαμε από το μάθημα διδασκαλίας “Ανάπτυξη εφαρμογών σε iOS” του Πανεπιστημίου Standford. Τα αρχεία AxesDrawer.h, AxesDrawer.m.

4.3 Σενάριο Χρήσης - Screenshots

Η εφαρμογή ανοίγει και εμφανίζεται η οθόνη που βλέπεται στην εικόνα. Υπάρχει το πεδίο για εισαγωγή της πολυωνυμικής εξίσωσης και τα 3 κουμπιά Instruction, Graph και BoundIt.

Για οδηγίες για το σε ποια μορφή μπορούμε να εισάγουμε την εξίσωση προσφέρονται με το κουμπί Instructions. Αφού πατήσουμε το κουμπί Instructions εμφανίζεται στη οθόνη ένα παράθυρο τύπου AlertView στο οποίο υπάρχουν οι απαραίτητες εξηγήσεις.

Εισάγουμε λοιπόν την εξίσωση στο διαθέσιμο πεδίο. Έπειτα έχουμε την επιλογή για να υπολογιστούν τα άνω και κάτω φράγματα πατώντας το κουμπί BoundIt.



Αλλιώς την επιλογή να σχεδιάσουμε τη γραφική παράσταση πατώντας το κουμπί Graph.

Η εφαρμογή μας έχει υλοποιηθεί για να δουλεύει τόσο σε κάθετη όσο και σε οριζόντια διάταξη.

Μελλοντικές Βελτιώσεις

Pinch and Zoom

Αν και προσφέρουμε τη δυνατότητα για ZoomIn και ZoomOut χρησιμοποιώντας κουμπιά, θα μπορούσαμε να αναπτύξουμε στην εφαρμογή μας και τη δυνατότητα αναγώρισης gestures. Για τη δυνατότητα αυτή μπορεί να βοηθήσει η κλάση UITapGestureRecognizer η οποία σε ρόλο delegate θα μπορεί να προσφέρει τις δυνατότητες ZoomIn και ZoomOut όχι πλέον με κουμπιά αλλά με το γνωστό gesture Pinch n' Zoom [2].

CorePlot

Η δική μας προσέγγιση για τη σχεδίαση της γραφικής παράστασης αν και πλήρης παρόλα αυτά σίγουρα αφήνει περιθώρια βελτίωσης. Γενικά το θέμα με το σχεδιασμό γραφικών παραστάσεων σε iOS είναι ένα θέμα που απασχολεί πολλούς. Από μόνο του το iOS SDK δε συμπεριλαμβάνει κάποια κλάση ή κάποιες μεθόδους ειδικά για σχεδίαση γραφικών παραστάσεων. Έτσι δημιουργήθηκε η ανάγκη από αρκετούς για να αναπτυχθούν τέτοιες βιβλιοθήκες και να προσφερθούν στο ευρύ κοινό υπό μορφή framework. Μια τέτοια προσπάθεια αποτελεί και το CorePlot, το οποίο αποτελεί framework τόσο για iOS όσο και για Mac. Αν και βρίσκεται σε αρχικά στάδια ανάπτυξης παρόλα αυτά έχει ήδη χρησιμοποιηθεί ήδη από αρκετές εφαρμογές, μερικές εκ των οποίων είναι οι εξής [6]:

iOS

PinPal: Bowling scorecard
aBattery: Battery charge tracking
FeedCount: Feedburner analytics
AServPoint: Monitors JBossAS application servers
BabyBump: App for pregnancy
Shattered Puzzles: Jigsaw puzzle game
Chronic Pain Tracker: Monitor and communicate pain metrics
GoRow: Rowing workout planner
Babymate: Monitors baby's development
iDashStocks: Track stock prices
Property Evaluator: Real Estate Analysis
Swim 222: Track your swimming
Fetal Weight: Estimates fetal weight during pregnancy
Ina Ruwa: Advanced Weather app.
Joggy Coach: Keep track of jogging progress and goals.
Treibstoffverbrauch: Calculates fuel consumption (German language)
SmartGrid: EirGrid application that allows users to browse and view key energy-related data.
Cube Time and Expense Tracker: Track expenses on iPhone and iPad
Myworth: Personal net worth manager.
TrackIt!: Track weight loss and eating habits.
iPerformanceCoach: Helps increase productivity and focus.
Water Storage iPhone: Monitor water storage in Australia

Money Engine: Track and analyze your spending habits
CalcRE: Real Estate Calculator
Workout Logbook: Log your workouts.
Germany Naturalisation Test +: Study for German naturalization test.
Analytix: Google Analytics app.
ClickyTouch: Web Analytics app.
LeaseSaver: Milage Tracking for Lease Cars.
Invoice2go: Invoicing Software for iPad
Morse Elmer: Morse code trainin

Mac OS

Risk Engine: Monte Carlo Analysis for iWork Numbers and Excel
MacHeliosSim: Nuclear reaction calculator, and simulator for the HELIOS detector.
mPortfolio: Stock Market portfolio tracking.
Property Evaluator: Real Estate Investment Analysis

Προυποθέτει περιβάλλον iOS SDK 4 με Xcode 3.2 και πάνω, και για να το προμηθευτεί κάποιος πρέπει να κατεβάσει τον πηγαίο του κώδικα χρησιμοποιώντας το εργαλείο Mercurial. Για να το κατεβάσει λοιπόν κάποιος απλά ανοίγει terminal και ενώ είναι συνδεδεμένος με το διαδίκτυο και έχει ήδη εγκατεστημένο το Mercurial, εισάγει την εξής εντολή [7]:

```
hg clone http://core-plot.googlecode.com/hg/ core-plot
```

Από κει και πέρα αναλαμβάνει το Mercurial. Το μόνο που έχει να κάνει κάποιος μετά είναι να το συμπεριλάβει στο project της δικής του εφαρμογής και ακολουθώντας πιστά το API του συγκεκριμένου framework να σχεδιάσει τις δικές του γραφικές παραστάσεις. Ομολογουμένως το αποτέλεσμα είναι πιο άρτιο και αισθητικό από τη δική μας προσπάθεια:

Βιβλιογραφία - Links

[1] iPhone 4 Specifications: http://www.apple.com/au/iphone/specs.html

[2] iOS Developer Center: http://developer.apple.com/devcenter/ios/index.action

[3] iOS Development class, Standford University:

http://www.stanford.edu/class/cs193p/cgi-bin/drupal/

[4] Wentk, Richard (2009). *Cocoa: Volume 5 of Developer Reference Apple Developer Series*. John Wiley and Sons,. ISBN 0470495898.

[5] Linear and Quadratic Complexity Bounds on the Values of the Positive Roots of Polynomials, Journal of Universal Computer Science, vol. 15, no. 3 (2009), 523-537

[6] CorePlot: http://code.google.com/p/core-plot/

[7] Using CorePlot in an iOS application:

http://www.switchonthecode.com/tutorials/using-core-plot-in-an-iphone-application