Κεφάλαιο 1

Εισαγωγή στη C

1.1 Ιστορία της C

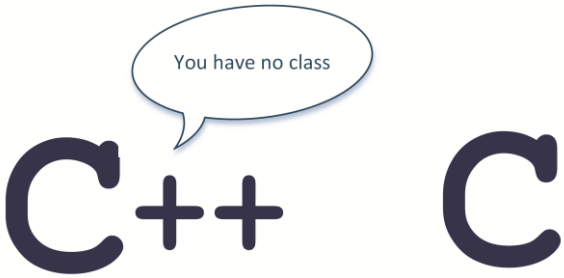
Η γλώσσα προγραμματισμού C δημιουργήθηκε στα εργαστήρια Bell στις αρχές της δεκαετίας του '70 για να μεταφερθεί το λειτουργικό σύστημα **Unix** από ένα σύστημα DEC PDP-7 σε ένα νεότερο PDP-11. Ο **Ken Thompson** είχε γράψει την αρχική έκδοση του Unix σε Assembly αλλά σύντομα συνειδητοποίησε πως έπρεπε να χρησιμοποιηθεί μία γλώσσα υψηλότερου επιπέδου. Αρχικά, προσπάθησε να χρησιμοποιήσει τη γλώσσα **B**, η οποία ήταν μια εξέλιξη της γλώσσας **BCPL** (Basic Combined Programming Language) που σχεδιάστηκε από τον Martin Richards. Ο **Dennis Ritchie**, ο οποίος συμμετείχε ως προγραμματιστής στην ομάδα, πρόσθεσε πάρα πολλά νέα στοιχεία στη γλώσσα B, ώστε τελικά η νέα γλώσσα να διαφέρει πολύ και να πάρει το όνομα **C**. Το Unix ξαναγράφτηκε σε C και έτσι ήταν πλέον πολύ ευκολότερο να μεταφερθεί σ' ένα άλλο σύστημα.

Με την πάροδο του χρόνου έγινε μια από τις πιο δημοφιλείς γλώσσες προγραμματισμού. Αν και είναι στενά συνδεδεμένη με το Unix, όλα σχεδόν τα λειτουργικά συστήματα διαθέτουν μεταγλωττιστή για τη γλώσσα C και πολλές άλλες γλώσσες προγραμματισμού βασίζονται σ' αυτή. Για παράδειγμα, η **C++** που αναπτύχθηκε από τον **Bjarne Stroustrup** το 1979 επίσης στα εργαστήρια Bell ως βελτίωση της γλώσσας προγραμματισμού C και αρχικά ονομάστηκε "C with Classes".

Σε αντίθεση με την BASIC ή την Pascal η γλώσσα C δε δημιουργήθηκε για την εκμάθηση του προγραμματισμού αλλά ως προγραμματιστικό εργαλείο. Είναι μία γενικής χρήσης γλώσσα προγραμματισμού και υποστηρίζει τον προγραμματισμό μέσω διαδικασιών με τη μορφή συναρτήσεων.

Αποτελεί ένα μείγμα γλώσσας "υψηλού" επιπέδου με πολλά χαρακτηριστικά γλώσσας μηχανής (πρόσβαση σε διευθύνσεις μνήμης, πράξεις σε επίπεδο bit κ.τ.λ.). Οι γλώσσες προγραμματισμού διακρίνονται σε "υψηλού" και "χαμηλού" επιπέδου, ανάλογα με το πόσο πολύ μοιάζουν στην ανθρώπινη γλώσσα. Όσο πιο "χαμηλό" είναι το επίπεδο μιας γλώσσας προγραμματισμού, τόσο πιο κοντά είναι στη μηχανή και τόσο πιο πολύπλοκος είναι ο προγραμματισμός μ' αυτή. Οι μεταγλωττιστές της γλώσσας C παράγουν όμως συμπαγή και ταχύτατα προγράμματα.

Το μεγαλύτερο πλεονέκτημα της C είναι η ελευθερία που δίνει στον προγραμματιστή να εφαρμόσει το προσωπικό του στυλ και να εκμεταλλευτεί όλες τις δυνατότητες του υπολογιστή. Επίσης, είναι μια γλώσσα γνωστή για τη μεταφερσιμότητα της, δηλαδή ο κώδικάς της μπορεί αυτούσιος ή με ελάχιστες αλλαγές να χρησιμοποιηθεί σε μηχανήματα διαφορετικών αρχιτεκτονικών. Βεβαίως, η λανθασμένη χρήση αυτής της ελευθερίας οδηγεί συχνά σε προγραμματιστικά σφάλματα. Επιπλέον, ο πλούτος τελεστών και το μικρό λεξιλόγιο που διαθέτει μπορεί να οδηγήσει σε δυσνόητο κώδικα, ο οποίος είναι δύσκολο να συντηρηθεί.



Στην αρχική έκδοση της C (**K&R** - από τους Kernighan και Ritchie, συγγραφείς του βιβλίου "The C Programming Language", 1978) προστέθηκαν καινούρια χαρακτηριστικά από διάφορους κατασκευαστές μεταγλωττιστών, με αποτέλεσμα την ύπαρξη ασυμβατοτήτων. Έτσι, έγινε εμφανής η ανάγκη για δημιουργία ενός επίσημου προτύπου (standard) που θα περιέγραφε τα χαρακτηριστικά της γλώσσας και θα το ακολουθούσαν όλοι οι κατασκευαστές μεταγλωττιστών C.

Το 1983 οργανώθηκε μια επιτροπή από το ANSI (American National Standards Institute) για την καθιέρωση ενός προτύπου. Μετά από έξι χρόνια, το Δεκέμβριο του 1989 δημοσιεύθηκε το πρότυπο *ANSI X3.159-1989 Programming Language C*, που είναι γνωστό ως **C89**. Ένα χρόνο αργότερα, το 1990, κυκλοφόρησε (με μικρές αλλαγές) το πρότυπο *ISO/IEC 9899:1990* από το ISO (International Organization for Standardization) ως C90. Το 1995 έγιναν μικρές αλλαγές (C95), ενώ η επόμενη ελαφρά προσαρμοσμένη έκδοση διατέθηκε το 1999 ως **C99**. Η τελευταία έκδοση με ονομασία **C11** δημοσιεύθηκε ως πρότυπο ISO/IEC 9899:2011 και τυποποιεί πολλά χαρακτηριστικά των σύγχρονων μεταγλωττιστών. Η σημαντικότερη αλλαγή αφορά την τυποποίηση του πολυνηματικού προγραμματισμού (Multithreaded Programming).

Για να γράψουμε "φορητό" κώδικα, συνιστάται η χρήση του πρότυπου C89 (ANSI C), το οποίο υποστηρίζεται από όλους τους σύγχρονους μεταγλωττιστές.

Στο βιβλίο αυτό θα χρησιμοποιήσουμε το πρότυπο ANSI C. Παρ' ότι πέρασε αρκετός χρόνος από την τελευταία τροποποίηση της γλώσσας, είναι σχετικά λίγοι οι μεταγλωττιστές που ακολουθούν πλήρως το νέο πρότυπο. Όπου γίνεται αναφορά σε νέα χαρακτηριστικά της γλώσσας, θα γίνεται ειδική σήμανση.

1.2 Ένα Απλό Παράδειγμα

Ας δούμε ένα απλό πρόγραμμα της C. Το παρακάτω πρόγραμμα είναι το λεγόμενο πρόγραμμα “Hello World” το οποίο τυπώνει τη φράση “hello, world” στην οθόνη. Χρησιμοποιείται συνήθως σε εισαγωγικά μαθήματα γλωσσών προγραμματισμού και δημοσιεύτηκε πρώτη φορά στο βιβλίο “The C Programming Language” των Brian Kernighan και Dennis Ritchie. Αν και διαφέρει λίγο από την αρχική έκδοση, η ουσία παραμένει η ίδια: να γνωρίσουμε τον τρόπο που συντάσσεται ένα απλό πρόγραμμα C.

```
1:   /* hello.c */
2:   #include <stdio.h>

3:   int main(void){
4:     printf("hello, world\n");
5:     return 0;
6:   }
```

Η 1η γραμμή του προγράμματος είναι ένα σχόλιο και δε λαμβάνεται υπόψη από τον μεταγλωττιστή. Τα σχόλια περικλείονται από τα σύμβολα **/*** (αρχή σχολίου) και ***/** (τέλος σχολίου) και βοηθούν στην κατανόηση ενός προγράμματος.

Η γραμμή 2 του προγράμματος

```
#include <stdio.h>
```

δεν είναι μια πραγματική εντολή της C, αλλά μια οδηγία προς τον προεπεξεργαστή (preprocessor directive) να ενσωματώσει το αρχείο επικεφαλίδας (header file) **stdio.h**, το οποίο περιέχει πληροφορίες για την πρότυπη βιβλιοθήκη εισόδου/εξόδου. Ο προεπεξεργαστής είναι ένα τμήμα του μεταγλωττιστή που αναλαμβάνει πριν απ’ αυτόν διάφορες οργανωτικές εργασίες. Η γραμμή αυτή εμφανίζεται σχεδόν σε κάθε αρχείο κώδικα της C.

Κάθε πρόγραμμα της C αποτελείται από μια βασική συνάρτηση που λέγεται **main**. Στις οριζόμενες από το χρήστη συναρτήσεις μπορούμε να δώσουμε ονόματα της επιλογή μας. Οι γραμμές 3 και 6 ορίζουν την αρχή και το τέλος της συνάρτησης main.

```
int main(void){

}
```

Μια συνάρτηση περιέχει μεταβλητές και εντολές και μπορεί να καλέσει άλλες συναρτήσεις. Τα άγκιστρα **{ }** υποδεικνύουν μια ενότητα (block) εντολών. Το σώμα όλων των συναρτήσεων πρέπει να περικλείεται σε άγκιστρα. Πολλοί προγραμματιστές τοποθετούν το αριστερό άγκιστρο **{** σε ξεχωριστή γραμμή:

```
int main(void)
{

}
```

Η C αδιαφορεί παγερά για το προσωπικό στυλ προγραμματισμού του καθενός, αρκεί να ακολουθούνται οι κανόνες σύνταξης της γλώσσας. Οι προγραμματιστές όμως, πρέπει να χρησιμοποιούν ένα καλό στυλ σύνταξης, ώστε ο κώδικάς τους να έχει ενιαία μορφή και να γίνεται εύκολα κατανοητός από άλλους αλλά και από τους ίδιους αργότερα.

Το **int** πριν από το όνομα της συνάρτησης προσδιορίζει τον τύπο της τιμής επιστροφής. Η συνάρτηση main θα επιστρέψει μια ακέραια τιμή (Integer) στο λειτουργικό σύστημα που την κάλεσε. Οι παρενθέσεις **( )** μετά το όνομα της συνάρτησης περικλείουν τη λίστα των ορισμάτων μέσω των οποίων η καλούσα συνάρτηση μεταβιβάζει στην καλούμενη μια λίστα από τιμές. Το **void** σημαίνει απουσία τιμής και στο συγκεκριμένο πρόγραμμα, δε μεταβιβάζονται τιμές στη συνάρτηση main. Οι εντολές της συνάρτησης περικλείονται ανάμεσα σε αγκύλες. Η συνάρτηση main περιέχει μία μόνο εντολή, την κλήση της συνάρτησης printf.

```
printf("hello, world\n");
```

Η printf είναι μια συνάρτηση βιβλιοθήκης και εκτυπώνει στην έξοδο (στην οθόνη) τα ορίσματά της, που στην περίπτωσή μας είναι το αλφαριθμητικό (string) “hello, world” και μια αλλαγή γραμμής **\n**. Ο χαρακτήρας **** ονομάζεται χαρακτήρας διαφυγής και υποδεικνύει ότι ο χαρακτήρας που τον ακολουθεί έχει ειδική σημασία. Εδώ, η ακολουθία διαφυγής **\n** υποδηλώνει την αλλαγή γραμμής.

Η εντολή

```
return 0;
```

επιστρέφει την τιμή μηδέν “0” στη διεργασία που κάλεσε τη main. Το μηδέν εξ ορισμού συμβολίζει την επιτυχή εκτέλεση ενός προγράμματος.

Κάθε εντολή πρέπει να τερματίζει με το ελληνικό ερωτηματικό “**;**”.

Πληκτρολογούμε τον κώδικα σε έναν επεξεργαστή κειμένου και το αποθηκεύουμε σ’ ένα αρχείο με επέκταση **.c**, για παράδειγμα **hello.c**. Το αρχείο αυτό αποτελεί τον πηγαίο κώδικα (source code) του προγράμματος.

*Τα αρχεία πηγαίου κώδικα πρέπει να έχουν επέκταση **.c**.*

1.3 Κανόνες και Συμβάσεις

Συνοψίζοντας αναφέρουμε μερικούς κανόνες και κάποιες συμβάσεις που πρέπει να ακολουθούμε, ώστε να παράγουμε καλύτερο κώδικα:

- Τα προγράμματα της C αποτελούνται από εντολές, οι οποίες εκτελούνται με τη σειριακά.
- Κάθε εντολή πρέπει να τερματίζεται με ελληνικό ερωτηματικό.
- Κάθε οδηγία προεπεξεργαστή πρέπει να βρίσκεται στην ίδια γραμμή.
- Χρησιμοποιούμε μία γραμμή ανά πρόταση και αφήνουμε κενές γραμμές για να ξεχωρίζουμε διαφορετικά τμήματα κώδικα.
- Η C είναι μια case-sensitive γλώσσα, δηλαδή κάνει διάκριση μεταξύ πεζών και κεφαλαίων γραμμάτων.
- Οι σταθερές συνηθίζεται να γράφονται με κεφαλαίους χαρακτήρες.
- Οι εντολές μιας ενότητας κώδικα π.χ. μιας συνάρτησης οριοθετούνται από άγκιστρα και καλό είναι να έχουν μια εσοχή 2 ή 4 χαρακτήρων.
- Τα κενά διαστήματα διευκολύνουν την ευανάγνωση του κώδικα.
- Οι γραμμές δεν πρέπει να έχουν μέγεθος μεγαλύτερο από 80 χαρακτήρες.
- Προτείνεται η χρήση αγκίστρων για την οριοθέτηση τμημάτων κώδικα, ακόμα και όταν αυτά δεν είναι απαραίτητα.

Κάποιες καλές προγραμματιστικές τεχνικές είναι:

- Αποφεύγουμε να χρησιμοποιούμε εξειδικευμένες λειτουργίες του μεταγλωττιστή, ώστε ο κώδικάς μας να μπορεί να μεταγλωττιστεί χωρίς προβλήματα σε διαφορετικά συστήματα.
- Η C επιτρέπει να γράψουμε “συμπτυγμένες” εντολές, οι οποίες είναι δύσκολα κατανοητές. Αποφεύγουμε σύνθετες εντολές και προσπαθούμε να γράφουμε “καθαρά” και κατανοητά προγράμματα.
- Τεκμηριώνουμε πολύ καλά τον κώδικα. Βασική αρχή είναι να έχουμε ένα ευπαρουσίαστο και ευανάγνωστο πρόγραμμα, ώστε να είναι εύκολη η συντήρησή του. Για παράδειγμα, κάθε συνάρτηση θα πρέπει να τεκμηριώνεται λεπτομερώς, ώστε να είναι εύκολα κατανοητή η λειτουργία και ο τρόπος κλήσης της.
- Τα ονόματα των μεταβλητών και των συναρτήσεων, αν και μπορούν να επιλεγούν ελεύθερα, θα πρέπει να δηλώνουν το περιεχόμενο και τη λειτουργία τους αντίστοιχα.

Όπως γίνεται φανερό, υπάρχουν πολύ λίγοι περιορισμοί όσον αφορά τη διάταξη των εντολών που αποτελούν ένα πρόγραμμα της C. Μπορούμε να προσθέτουμε κενά διαστήματα και αλλαγές γραμμής κατά βούληση, αρκεί να μη “χωρίζουμε” τις δεσμευμένες λέξεις και τα ονόματα μεταβλητών, συναρτήσεων κ.τ.λ. Πρέπει, όμως, να χρησιμοποιούμε ένα ευανάγνωστο και ομοιόμορφο στυλ γραφής, αν δε θέλουμε να έχουμε προβλήματα κατανόησης των προγραμμάτων μας. Ειδικά σε μεγάλα έργα, όπου συμμετέχουν περισσότερα άτομα, θα πρέπει όλα τα μέλη που εργάζονται για το έργο να ακολουθούν ένα κοινό στυλ προγραμματισμού.

1.4 Μεταγλώττιση και Εκτέλεση σε Σύστημα Linux

Μεταγλώττιση είναι η διαδικασία μετατροπής ενός προγράμματος από τη μορφή του πηγαίου κώδικα σε γλώσσα μηχανής. Το “εκτελέσιμο” αρχείο που παράγεται καλείται πολλές φορές και δυαδικό αρχείο (binary), επειδή αποτελείται από μια ακολουθία 0 και 1. Η μετάφραση από μια γλώσσα υψηλού επιπέδου στη γλώσσα μηχανής γίνεται με τη βοήθεια ειδικών προγραμμάτων που ονομάζονται μεταγλωττιστές (compilers). Σ’ ένα σύστημα Linux ο μεταγλωττιστής της C είναι συνήθως ο **gcc**, από το πρόγραμμα ελεύθερου λογισμικού GNU Compiler Collection.

Η διαδικασία της μεταγλώττισης είναι πλήρως αυτοματοποιημένη. Καλούμε τον μεταγλωττιστή από τη γραμμή εντολών υποδεικνύοντας το πηγαίο αρχείο και ενδεχομένως κάποιες επιλογές. Αν δεν υπάρχουν συντακτικά λάθη, παράγεται το αντικείμενο αρχείο (object file), το οποίο όμως δεν είναι εκτελέσιμο. Τα αρχεία αυτά έχουν επέκταση **.o**.

Στη συνέχεια καλείται (αυτόματα) ο συνδέτης (linker), που έχει τη δυνατότητα να συνδέσει ένα ή περισσότερα αντικείμενα αρχεία για την παραγωγή του εκτελέσιμου αρχείου (executable file). Στη φάση αυτή συνδυάζεται ο δυαδικός κώδικας των συναρτήσεων βιβλιοθήκης που καλούνται στον πηγαίο κώδικα με το αντικείμενο πρόγραμμα.

Στο λειτουργικό σύστημα Linux μπορούμε να πάρουμε βοήθεια για τη σύνταξη μιας εντολής με την κλήση της εντολής man, για παράδειγμα:

```
$ man gcc
```

Αν θέλουμε να μη δημιουργήσουμε εκτελέσιμο αρχείο αλλά μόνο το αντικείμενο πρόγραμμα, θα πληκτρολογήσουμε την εντολή:

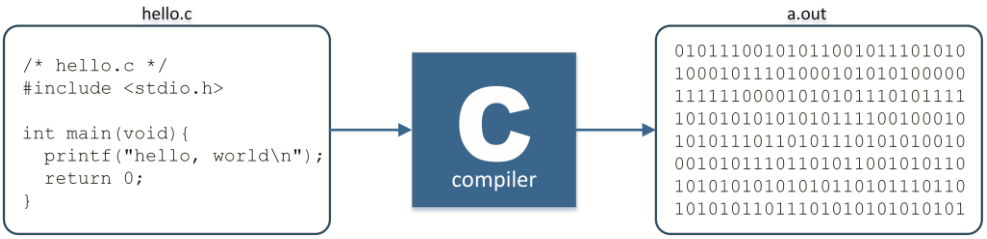
```
$ gcc -c filename.c
```

οπότε θα δημιουργηθεί μόνο το αρχείο filename.o.

Για να μεταγλωττίσουμε το πρόγραμμά μας πληκτρολογούμε την εντολή:

```
$ gcc hello.c
```

Αν δεν υπάρχουν στο πρόγραμμα συντακτικά λάθη, θα ανακαλύψουμε ότι στον τρέχοντα κατάλογο θα δημιουργηθεί ένα αρχείο με όνομα **a.out**.



Το αρχείο αυτό είναι εκτελέσιμο και μπορούμε να το τρέξουμε πληκτρολογώντας στη γραμμή εντολών του λειτουργικού συστήματος:

```
$ ./a.out
```

οπότε εμφανίζεται στην οθόνη μας η φράση **hello, world**.

Αν θέλουμε το πρόγραμμα να έχει ένα διαφορετικό όνομα από το προκαθορισμένο a.out, μπορούμε να χρησιμοποιήσουμε τον διακόπτη **-o** στην εντολή μεταγλώττισης. Έτσι, η εντολή

```
$ gcc hello.c -o hello
```

θα δημιουργήσει το εκτελέσιμο αρχείο **hello**, το οποίο εκτελείται πληκτρολογώντας στη γραμμή εντολών:

```
$ ./hello
```

Η γλώσσα προγραμματισμού C είναι μια εξαιρετικά μικρή γλώσσα. Πολλές από τις λειτουργίες της δεν περιλαμβάνονται στη γλώσσα καθ' αυτή, αλλά παρέχονται μέσω βιβλιοθηκών συναρτήσεων. Κάθε βιβλιοθήκη (library) διαθέτει ένα αρχείο επικεφαλίδας, το οποίο περιέχει τα πρωτότυπα των συναρτήσεών της, δηλώσεις ειδικών τύπων δεδομένων και μακροεντολές. Για να χρησιμοποιήσει ένα πρόγραμμα μια συνάρτηση βιβλιοθήκης, θα πρέπει να ενσωματωθεί στον κώδικά του το αρχείο επικεφαλίδας της και κατά τη μεταγλώττιση να γίνει σύνδεση με τον διακόπτη **-l**. Για παράδειγμα, αν θέλουμε να χρησιμοποιήσουμε σ' ένα πρόγραμμα τη συνάρτηση **sqrt**, η οποία υπολογίζει την τετραγωνική ρίζα ενός αριθμού, πρέπει να συμπεριλάβουμε στον κώδικα την οδηγία

```
#include <math.h>
```

ώστε να συνδέσουμε τη βιβλιοθήκη μαθηματικών συναρτήσεων (libm.a) κατά τη μεταγλώττιση με την εντολή:

```
$ gcc calculate.c -o calculate -lm
```

1.5 Λάθη Μεταγλώττισης

Οι μεταγλωττιστές εντοπίζουν μόνο τα συντακτικά λάθη των προγραμμάτων μας. Ο εντοπισμός και η διόρθωση των λαθών αυτών ονομάζεται αποσφαλμάτωση και περιλαμβάνει διάφορα βήματα. Ο απλούστερος τρόπος αποσφαλμάτωσης κώδικα είναι να ζητήσουμε από το μεταγλωττιστή με τη χρήση κάποιου διακόπτη, να μας τυπώνει εκτός από τα λάθη (errors) όσο το δυνατόν περισσότερες προειδοποιήσεις (warnings). Ο διακόπτης **-Wall** συνδυάζει τα πιο συνηθισμένα είδη προειδοποιήσεων για πιθανά λάθη και συνιστάται να χρησιμοποιείται σε κάθε μεταγλώττιση.

Ας δούμε στη συνέχεια μερικά από τα συνηθισμένα λάθη που κάνουν οι αρχάριοι προγραμματιστές της C:

1) Κάνουμε ένα αντίγραφο του πρώτου μας προγράμματος και στη γραμμή 4 του κώδικα διαγράφουμε το ερωτηματικό της printf. Η αντιγραφή ενός αρχείου στα λειτουργικά συστήματα UNIX/Linux μπορεί να γίνει με την εντολή **cp** (copy):

```
$ cp 1-01.c 1-02.c
```

Ο κώδικας του νέου αρχείου τροποποιείται ως εξής:

```
1:   /* 1-02.c */
2:   #include <stdio.h>

3:   int main(void){
4:     printf ("Hello, world!\n") /* missing semicolon*/
5:     return 0;
6:   }
```

Η μεταγλώττιση του προγράμματος θα εμφανίσει το επόμενο μήνυμα λάθους:

```
$ gcc 1-02.c -o 1-02
1-02.c: In function ‘main’:
1-02.c:5:3: error: expected ‘;’ before ‘return’
$
```

Το λάθος αυτό εντοπίζεται εύκολα, αφού το μήνυμα του μεταγλωττιστή είναι ξεκάθαρο: «Στο αρχείο 1-02.c και στη συνάρτηση “main”, λείπει στη γραμμή 5 το “**;**” πριν από την εντολή return».

Τα μηνύματα που παράγει το GCC έχουν τη μορφή αρχείο:γραμμή:μήνυμα.

2) Στο επόμενο παράδειγμα δεν έχουμε δηλώσει την ακέραια μεταβλητή y:

```
1:   /* 1-03.c */
2:   #include <stdio.h>

3:   int main(void){
4:     int x;
5:     x = y + 5;
6:     return 0;
7:   }
```

Το μήνυμα του μεταγλωττιστή είναι:

```
1-03.c: In function ‘main’:
1-03.c:5: error: ‘y’ undeclared (first use in this
function)
1-03.c:5: error: (Each undeclared identifier is
reported only once
1-03.c:5: error: for each function it appears in.)
```

3) Στο παράδειγμα αυτό, θα κάνουμε κλήση της συνάρτησης **sqrt**, που όπως αναφέραμε υπολογίζει την τετραγωνική ρίζα ενός αριθμού. Στον κώδικα δεν θα συμπεριλάβουμε το αρχείο επικεφαλίδας **math.h**.

```
1:   /* 1-04.c */
2:   #include <stdio.h>

3:   int main(void){
4:     printf("Square root of 9 is %f\n", sqrt(9.0));
5:     return 0;
6:   }
```

Μεταγλωττίζουμε το πρόγραμμα με το διακόπτη **-lm** και στην οθόνη εμφανίζεται η παρακάτω προειδοποίηση:

```
1-04.c: In function ‘main’:
1-04.c:4: warning: incompatible implicit declaration of
built-in function ‘sqrt’
```