1. Εισαγωγή – βασικά στοιχεία προγράμματος

Σύνοψη
Στο κεφάλαιο αυτό αρχικά γίνεται μία εισαγωγή στην έννοια του προγραμματισμού με γλώσσα υψηλού επιπέδου. Παρουσιάζεται η ιστορική εξέλιξη της γλώσσας C και δίνονται τα γενικά χαρακτηριστικά της γλώσσας. Ακολουθεί η περιγραφή των εργαλείων ανάλυσης προβλημάτων προγραμματισμού και των σταδίων υλοποίησης προγράμματος. Στη συνέχεια μελετώνται τα βασικά στοιχεία των προγραμμάτων και το λεξιλόγιο της C και το κεφάλαιο ολοκληρώνεται με την αποτύπωση κανόνων για τη δημιουργία ευανάγνωστων προγραμμάτων.

Λέξεις κλειδιά
γλώσσα προγραμματισμού υψηλού επιπέδου, διαδικαστικός προγραμματισμός, γλώσσα C, πρότυπο γλώσσας, φυσική γλώσσα, διάγραμμα ροής, ψευδοκώδικας, συντάκτης, μεταγλωττιστής, συνδέτης, εκτελέσιμο αρχείο, πηγαίος κώδικας, αντικείμενο αρχείο, συντακτικά και σημασιολογικά σφάλματα, σχόλια, λεξεις κλειδιά, δεσμευμένες λέξεις, αναγνωριστές, main, αρχείο κεφαλίδας

1.1. Εισαγωγή

Αντικείμενο του παρόντος συγγράμματος είναι η εισαγωγή του αναγνώστη στη λογική του προγραμματισμού Η/Υ. Το ενδιαφέρον εστιάζεται στον επονομαζόμενο **διαδικαστικό προγραμματισμό** (procedural programming), βασικά στοιχεία του οποίου είναι η δόμηση του προγράμματος και η επαναλαμβανόμενη χρήση υποπρογραμμάτων, τα οποία είτε επιτελούν εργασίες γενικής φύσης είτε απευθύνονται σε ένα τμήμα του συνολικού προβλήματος. Στόχος είναι η κατανόηση των αρχών του προγραμματισμού και η εμπέδωση της φιλοσοφίας του, έτσι ώστε ο αναγνώστης να προχωρήσει σε άλλες μορφές προγραμματισμού, όπως ο αντικειμενοστραφής προγραμματισμός (object–oriented programming), έχοντας αποκτήσει τις γνώσεις που αποτελούν κοινό τόπο ανάμεσα στα είδη προγραμματισμού.

Στην προσπάθεια αυτή θα χρησιμοποιηθεί ως πλατφόρμα μία γλώσσα προγραμματισμού **υψηλού επιπέδου**, η γλώσσα C. Ο όρος γλώσσα υψηλού επιπέδου υποδηλώνει ότι δεν είναι κατασκευασμένη για να λειτουργεί σε συγκεκριμένη αρχιτεκτονική υπολογιστή, αλλά δύναται να λειτουργήσει σε πληθώρα αρχιτεκτονικών, γεγονός που σημαίνει ότι:

- Οι διάφορες αρχιτεκτονικές υπολογιστών, για να λειτουργήσουν, χρησιμοποιούν ένα σύνολο εντολών, το οποίο χρησιμοποιεί τη γλώσσα μηχανής της εκάστοτε αρχιτεκτονικής. Επομένως, έαν κάποιος προγραμματίσει κάνοντας χρήση της γλώσσας μηχανής μίας αρχιτεκτονικής, τα προγράμματά του δε θα είναι συμβατά με άλλες αρχιτεκτονικές. Το αντιστάθμισμα αυτού του μειονεκτήματος είναι ότι στο παρελθόν, που οι υπολογιστές είχαν λίγη μνήμη και χαμηλή ταχύτητα, ο προγραμματισμός σε γλώσσα μηχανής χειριζόταν αποδοτικότερα τους πόρους του μηχανήματος.
- Για να είναι μία γλώσσα συμβατή με τις γλώσσες μηχανής διάφορων αρχιτεκτονικών, θα πρέπει να λαμβάνει χώρα μεταγλώττιση από τη γλώσσα προγραμματισμού στην εκάστοτε γλώσσα μηχανής. Όντως αυτό συμβαίνει και το λογισμικό που επιτελεί τη συγκεκριμένη εργασία ονομάζεται **μεταγλωττιστής** (compiler).

Με βάση τα παραπάνω, οι γλώσσες προγραμματισμού υψηλού επιπέδου είναι ως επί το πλείστον *μεταγλωττισμένες* γλώσσες, αποσκοπώντας στο να είναι ευανάγνωστες και κατανοητές.

Σε ό,τι αφορά τη C, παρουσιάζει μία σειρά από ενδιαφέροντα και χρήσιμα χαρακτηριστικά:

- Μπορεί να χρησιμοποιηθεί και ως γλώσσα προγραμματισμού χαμηλού επιπέδου, επιτρέποντας άμεση πρόσβαση στους πόρους του υπολογιστή.

- Είναι σχετικά μικρή και εύκολη στην εκμάθηση.
- Υποστηρίζει δομημένο προγραμματισμό.
- Είναι αποτελεσματική, παράγοντας συμπαγή και γρήγορα στην εκτέλεση προγράμματα.
- Έχει φορητότητα, δηλαδή ο κώδικάς της μεταγλωττίζεται από διάφορους μεταγλωττιστές χωρίς να απαιτούνται τροποποιήσεις.
- Μετά από τέσσερις και πλέον δεκαετίες συνεχίζει να αποτελεί μία από τις ευρύτερα χρησιμοποιούμενες γλώσσες προγραμματισμού, γεγονός που έχει δημιουργήσει πολύ μεγάλη εγκατεστημένη βάση εφαρμογών που αναπτύχθηκαν με αυτήν τη γλώσσα και πρέπει να συντηρούνται και να εξελίσσονται.
- Αποτελεί μία εξαιρετική εκκίνηση για την εκμάθηση γλωσσών που στηρίζονται στον αντικειμενοστραφή προγραμματισμό (C++, Java, C#), καθώς ο τελευταίος έχει δανειστεί πολλά χαρακτηριστικά από τη C.

Ωστόσο, η γλώσσα C παρουσιάζει και μία σειρά μειονεκτημάτων, που την καθιστούν απαιτητική στον χειρισμό. Παρέχοντας μεγάλο βαθμό ελευθερίας στον προγραμματιστή και χαρακτηριζόμενη από έλλειψη περιορισμών και μικρό βαθμό ελέγχου λαθών, υποχρεώνει τον προγραμματιστή να είναι ιδιαίτερα προσεκτικός και να χρησιμοποιεί διαδικασίες ελέγχου λαθών, οι οποίες παρέχονται αυτόματα σε άλλες γλώσσες προγραμματισμού. Επιπλέον, σε πολλές περιπτώσεις εισάγονται λάθη που είναι μη ανιχνεύσιμα από τον μεταγλωττιστή και οδηγούν σε καταστάσεις απροσδιοριστίας, δηλαδή μη αναμενόμενες καταστάσεις με μη προσδιορισμένη συμπεριφορά.

1.2. Ιστορική αναδρομή της γλώσσας C

Η C επινοήθηκε το 1972 από τον Dennis Ritchie στα εργαστήρια Bell. Δημιουργήθηκε για να εξυπηρετήσει το λειτουργικό σύστημα Unix, το οποίο έως τότε ήταν γραμμένο σε assembly. Ο δημιουργός του Unix, Ken Thompson, φίλος και συνεργάτης του Ritchie, είχε δημιουργήσει την πρόγονο της C, τη γλώσσα B. Και οι δύο γλώσσες έχουν κοινή καταγωγή από τη γλώσσα BCPL, η οποία είχε αναπτυχθεί από τον Martin Richards κατά το πέρασμά του απο το Τεχνολογικό Ινστιτούτο της Μασσαχουσέττης (MIT) το 1967, στηριζόμενη στη γλώσσα CPL (Cambridge Programming Language) του Πανεπιστημίου του Cambridge. Και οι τρεις γλώσσες κατασκευάστηκαν στο πλαίσιο του προγράμματος MAC και του απογόνου του Multics, τα οποία στόχευαν στην κατανομή των πόρων των υπολογιστών σε πολλούς χρήστες. Τα δύο αυτά προγράμματα, στα οποία συνέπραξαν το MIT, η General Electric και τα εργαστήρια Bell, αποτέλεσαν τη θερμοκοιτίδα πολλών προγραμμάτων λογισμικού, που κυριαρχούν από τη δεκαετία του 1960 έως σήμερα.

Η C, ούσα ευέλικτη και αποδοτική, χρησιμοποιήθηκε αρχικά για τον προγραμματισμό συστημάτων στο Unix. Το 1974 εμφανίστηκε από τον Brian Kernighan το πρώτο γραπτό κείμενο για τη γλώσσα, υπό τον τίτλο “Programming in C: A Tutorial”. Το 1977 έγινε η πρώτη επίσημη τεκμηρίωση της γλώσσας με το βιβλίο “The C Programming Language” από τους Kernighan και Ritchie. Το βιβλίο αυτό αποτέλεσε το «ευαγγέλιο» των προγραμματιστών της C, αποκαλούμενο «Λευκή Βίβλος» ή «πρότυπο K&R».

Με την πάροδο του χρόνου η γλώσσα C άρχισε να χρησιμοποιείται και σε άλλα πεδία εφαρμογών, πέραν του προγραμματισμού συστημάτων. Η εμφάνιση των μεταγλωττιστών της γλώσσας στο MS–DOS και ο μεγάλος αριθμός προγραμμάτων βιβλιοθήκης που κατασκευάστηκαν, οδήγησαν τη γλώσσα στο απόγειό της στα τέλη της δεκαετίας του 1980. Βέβαια, η γλώσσα γνώρισε πολλές αλλαγές, οδηγούμενη τελικά το 1989 στην επονομαζόμενη **ANSI έκδοση**, που την προτυποποίησε (το όνομά της το έλαβε από την επιτροπή του American National Standards Institute). Μία σημαντική ανανέωση έγινε το 1999 με το πρότυπο C99, ενώ το τελευταίο πρότυπο παρουσιάστηκε το 2011. Ωστόσο, τα καινοτόμα στοιχεία του προτύπου C11, όπως η προτυποποίηση του πολυνηματικού προγραμματισμού, δεν έχουν ακόμη ενσωματωθεί στους περισσότερους μεταγλωττιστές της γλώσσας.

Τις τελευταίες δύο δεκαετίες τα ηνία έχει λάβει ο αντικειμενοστρεφής προγραμματισμός και στην κατεύθυνση αυτή αρχικά συνέβαλε η γλώσσα C++, που επινοήθηκε το 1983 από τον Δανό Bjarne Stroustrup στα εργαστήρια της AT&T (το τμήμα των εργαστηρίων Bell που μεταφέρθηκε στην AT&T, όταν η πρώτη διασπάστηκε). Η C++ – ή *C με τάξεις* – μπορεί να θεωρηθεί απόγονος της C, αν και έχει αρκετές διαφορές. Πάντως, θα πρέπει να σημειωθεί ότι, σύμφωνα με μετρήσεις σχετικών οργανισμών για την επαγγελματική και ακαδημαϊκή χρήση των γλωσσών προγραμματισμού, το 2015 η γλώσσα C συνεχίζει να αποτελεί τη γλώσσα στην οποία έχουν γραφτεί οι περισσότερες γραμμές κώδικα.

1.3. Εργαλεία ανάλυσης προβλημάτων

Για την ανάλυση ενός προβλήματος και την κατάρτιση του μοντέλου που θα υλοποιηθεί σε κώδικα έχουν επικρατήσει τα ακόλουθα τρία εργαλεία:

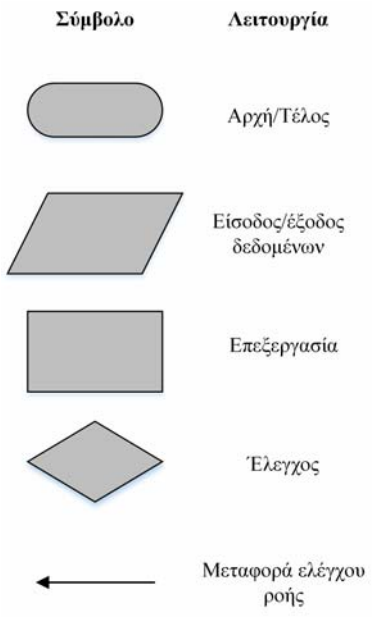
1. Η **φυσική γλώσσα** (natural language), σύμφωνα με την οποία το πρόβλημα αναλύεται σε απλές προτάσεις της καθομιλούμενης γλώσσας, χρησιμοποιώντας το συντακτικό αυτής.

2. Ο ψευδοκώδικας (pseudocode), ο οποίος μετασχηματίζει τη φυσική γλώσσα σε μία σειρά προτάσεων που χρησιμοποιούν το συντακτικό γλώσσας προγραμματισμού, χωρίς να ακολουθούν επακριβώς τον φορμαλισμό συγκεκριμένης γλώσσας.

3. Το λογικό διάγραμμα ή διάγραμμα ροής (flow chart), σύμφωνα με το οποίο απεικονίζεται γραφικά η λύση του προβλήματος, με χρήση ειδικών συμβόλων.

Δεν υπάρχουν γενικοί κανόνες για την υιοθέτηση κάποιου από τα τρία εργαλεία. Η χρήση καθενός εκ των τριών εξαρτάται από τον εκάστοτε προγραμματιστή και το συγκεκριμένο πρόβλημα. Το λογικό διάγραμμα χρησιμοποιείται ευρύτατα, όταν συνεργάζονται ομάδες προγραμματιστών, γιατί παρέχει μία σειρά από πλεονεκτήματα, καθώς προσδίδει μεγαλύτερη ευχέρεια *(α)* στην παραστατική ανάλυση του προβλήματος, *(β)* στην τεκμηρίωση του προγράμματος, *(γ)* στην αποδοτική συντήρηση του προγράμματος. Ωστόσο, έχει περιορισμούς, καθόσον *(α)* σύνθετα προγράμματα οδηγούν σε σύνθετα και δύσχρηστα διαγράμματα ροής και *(β)* τροποποιήσεις στα προγράμματα μπορεί να απαιτήσουν επανασχεδιασμό ολόκληρου του διαγράμματος.

Τα σύμβολα που χρησιμοποιούνται στο διάγραμμα ροής παρουσιάζονται στο **Σχήμα 1.1**:



Σχήμα 1.1 *Τα σύμβολα του διαγράμματος ροής*

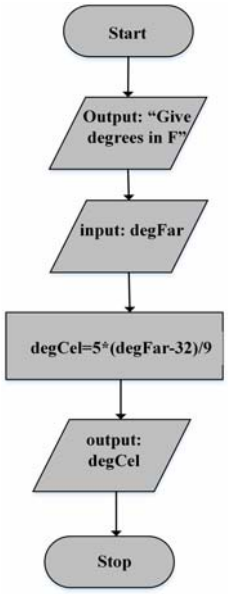
1.3.1. Παράδειγμα

Μία ενδεικτική περιγραφή του τρόπου εφαρμογής των ανωτέρω εργαλείων θα γίνει με τη βοήθεια του ακόλουθου προβλήματος: Να μετατραπούν οι βαθμοί Farenheit σε βαθμούς Κελσίου, χρησιμοποιώντας την εξίσωση μετασχηματισμού $C = 5 \cdot (F - 32) / 9$.

1. Με χρήση φυσικής γλώσσας

Ζήτησε από τον χρήστη τη θερμοκρασία σε βαθμούς F
Διάβασε την τιμή που δίνει ο χρήστης
Αποθήκευσε την τιμή σε θέση αποθήκευσης που ονομάζεται `degFar`
Υπολόγισε τους βαθμούς C με χρήση μαθηματικής σχέσης
Αποθήκευσε το αποτέλεσμα σε θέση αποθήκευσης που ονομάζεται `degCel`
Τύπωσε το περιεχόμενο της `degCel`

2. Με χρήση διαγράμματος ροής



Σχήμα 1.2 *Το διάγραμμα ροής του παραδείγματος 1.3.1*

3. Με χρήση ψευδοκώδικα

```
print "enter degrees in Farenheit"
read degFar
degCel = (degFar - 32) * 5 / 9
print degCel
```

1.4. Στάδια ανάπτυξης προγράμματος

Το υπολογιστικό πρόγραμμα είναι μία ακολουθία εντολών, με τις οποίες ο υπολογιστής εκτελεί μία συγκεκριμένη εργασία και επιλύει ένα δοθέν πρόβλημα. Η υλοποίηση ενός προγράμματος– ο επονομαζόμενος και *κύκλος δημιουργίας προγράμματος*– περιλαμβάνει τέσσερα στάδια:

1. Η συγγραφή του πηγαίου κώδικα (source code). Στο βήμα αυτό χρησιμοποιείται ένας **συντάκτης κειμένου** (text editor) για τη συγγραφή του κώδικα. Συνήθως, χρησιμοποιείται ο ενσωματωμένος συντάκτης της γλώσσας C. Το αποτέλεσμα είναι ένα αρχείο κειμένου, αναγνώσιμο από οποιονδήποτε συντάκτη (notepad, wordpad, πρόγραμμα επεξεργασίας κειμένου κ.λπ.), το οποίο έχει κατάληξη `.c`.

2. Η μεταγλώττιση του πηγαίου κώδικα (compilation). Η διαδικασία της μεταγλώττισης εκτελείται από τον **μεταγλωττιστή** (compiler) και παράγεται ένα αρχείο που ονομάζεται **αρχείο αντικειμένου** (object file) και περιέχει τον κώδικα σε γλώσσα μηχανής. Στο στάδιο αυτό ανιχνεύονται τα **συντακτικά σφάλματα** (syntax errors), τα οποία είναι σφάλματα που οφείλονται σε παραβίαση των συντακτικών κανόνων και αναφέρονται υπό μορφή λίστας (οπότε μπορούν να διορθωθούν, προτού εκτελεστεί το πρόγραμμα). Εάν δεν υπάρχουν συντακτικά σφάλματα, το αρχείο που προκύπτει έχει το όνομα του αρχείου του πηγαίου κώδικα και κατάληξη **.obj**.

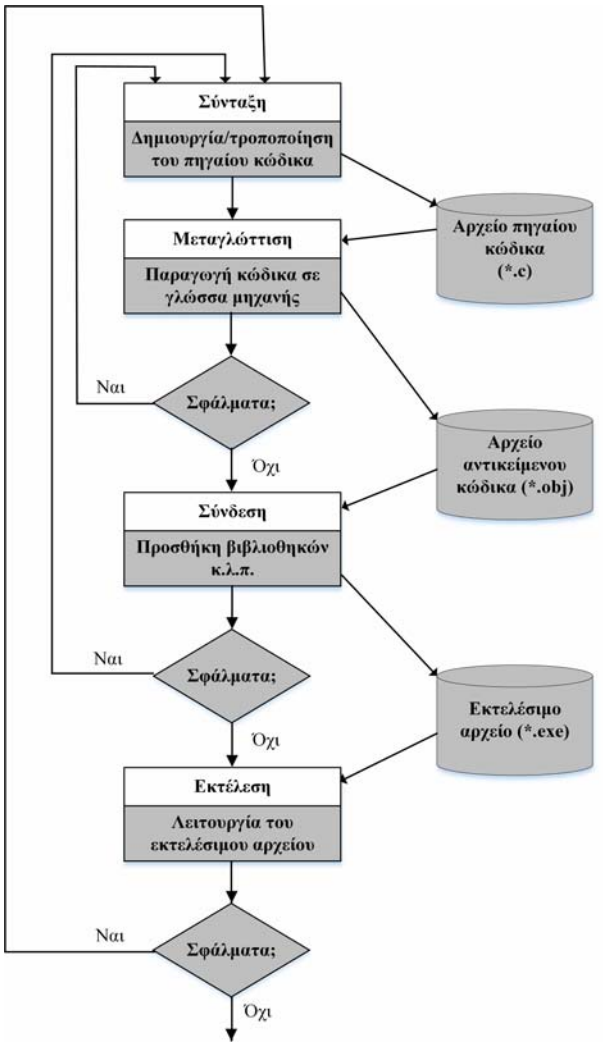
3. Η σύνδεση του αντικειμένου με βιβλιοθήκες. Στο τρίτο στάδιο επιτελείται η διεργασία της σύνδεσης (linking), η οποία είναι πλήρως αυτοματοποιημένη και γίνεται με χρήση του **συνδέτη** (linker). Το αποτέλεσμα της σύνδεσης είναι η δημιουργία του εκτελέσιμου κώδικα ή η αναφορά τυχόν προβλημάτων, όπως για παράδειγμα η αδυναμία εντοπισμού μίας συνάρτησης ή μίας εξωτερικής μεταβλητής. Ο εκτελέσιμος κώδικας αποθηκεύεται σε αρχείο που έχει το όνομα του πηγαίου κώδικα και επέκταση **.exe**. Πλέον το πρόγραμμα μπορεί να εκτελεστεί όπως ένα οποιοδήποτε εκτελέσιμο αρχείο του υπολογιστή.

Ο συνδέτης έχει τη δυνατότητα να συνδέσει περισσότερα του ενός αρχεία αντικειμένου. Επιπλέον, αναζητά σε βιβλιοθήκες τα σώματα των συναρτήσεων που ο προγραμματιστής χρησιμοποίησε στο πρόγραμμά του (λεπτομέρειες στην §1.5 και σε επόμενα κεφάλαια).

4. Η εκτέλεση του προγράμματος. Στο τελευταίο στάδιο εκτελείται ή «τρέχει» το αρχείο **.exe** και ελέγχεται η ορθή λειτουργία του προγράμματος. Τα σφάλματα που ανακύπτουν στο στάδιο αυτό ονομάζονται **σημασιολογικά σφάλματα** (semantic errors). Οφείλονται σε κακή σχεδίαση της λύσης του προβλήματος και, δυστυχώς, αναγνωρίζονται στον χρόνο εκτέλεσης. Χαρακτηριστικό σφάλμα εκτέλεσης είναι η διαίρεση αριθμού με το μηδέν, ενέργεια που δεν ανιχνεύεται ως λανθασμένη κατά τη μεταγλώττιση.

Σε περίπτωση σφάλματος, ο προγραμματιστής επιστρέφει στο πρώτο στάδιο, όπου κάνει τις διορθώσεις, και επαναλαμβάνει τα υπόλοιπα στάδια, έως ότου το πρόγραμμα λειτουργήσει επιτυχώς.

Στο **Σχήμα 1.3** αποτυπώνονται εποπτικά τα στάδια υλοποίησης προγράμματος:



Σχήμα 1.3 *Τα στάδια υλοποίησης προγράμματος*

1.5. Βασικά στοιχεία προγράμματος

Έστω το ακόλουθο απλό πρόγραμμα:

```
/*****************************************************
 This program prints out the sentence "This is a test."
 *****************************************************/

#include <stdio.h>

int main()
{
  printf( "This is a test.\n" );
  return 0;
}  // End of main
```

Η καρδιά ενός προγράμματος της γλώσσας C είναι η λέξη **main**. Αντιστοιχεί στο κύριο τμήμα του κώδικα (κύρια *συνάρτηση* του προγράμματος, γι' αυτό και εφεξής θα αναφέρεται ως **main()**) και χρησιμοποιείται για να γνωστοποιήσει το σημείο έναρξης της εκτέλεσης του προγράμματος. Μετά τη **main()** ακολουθεί το εισαγωγικό αριστερό άγκιστρο (**{**) και κατόπιν οι προτάσεις του προγράμματος. Η **main()** και μαζί το πρόγραμμα τερματίζουν με το καταληκτικό δεξί άγκιστρο (**}**). Στο παρόν πρόγραμμα η **main()** αποτελείται από μία πρόταση κλήσης της συνάρτησης **printf()**, η οποία ανήκει στη βασική βιβλιοθήκη συναρτήσεων της C, καθώς και από την πρότυπη τελευταία πρόταση των προγραμμάτων της γλώσσας C **return 0;**. Η βιβλιοθήκη συναρτήσεων περιλαμβάνει τον κώδικα πρότυπων συναρτήσεων και αποτελείται από μία σειρά αρχείων, τα οποία έχουν την κατάληξη **h**. Τα αρχεία αυτά ονομάζονται **αρχεία κεφαλίδας** (header files) και περιλαμβάνουν συναρτήσεις συναφούς λειτουργίας. Δηλώνονται πριν από τη **main()**, με χρήση της *οδηγίας προς τον προεπεξεργαστή* (preprocessor directive) **include** ως εξής:

```
#include <όνομα_αρχείου_κεφαλίδας.h>
```

Στην παρούσα περίπτωση το αρχείο κεφαλίδας είναι το **stdio.h** (**st**an**d**ard **i**nput–**o**utput), το οποίο περιλαμβάνει συναρτήσεις σχετιζόμενες με τις συσκευές εισόδου (π.χ. πληκτρολόγιο) και εξόδου (π.χ. οθόνη).

Η συνάρτηση **main()** εκτός από προτάσεις και κλήσεις σε συναρτήσεις βιβλιοθήκης μπορεί να καλεί και άλλες συναρτήσεις, οι οποίες δημιουργούνται από τον προγραμματιστή. Μία συνάρτηση είναι ένα σύνολο προτάσεων με ένα δεδομένο όνομα, όπως είναι η **main()** ή η **printf()**. Οι προτάσεις αποτελούν το **σώμα** (body) της συνάρτησης και περικλείονται σε άγκιστρα. Λεπτομερής ανάλυση του συντακτικού και της λειτουργίας των συναρτήσεων θα δοθεί στο Κεφάλαιο 4.

Όλες οι προτάσεις τελειώνουν με **ερωτηματικό** (**;**) (semicolon), το οποίο ονομάζεται **σύμβολο του τερματιστή προτάσεων** (statement terminator symbol). Εξαίρεση αποτελούν οι οδηγίες προς τον προεπεξεργαστή, όπως είναι η **include**, οι οποίες αρχίζουν πάντοτε με **δίεση** (**#**) και δεν τελειώνουν με ερωτηματικό**.**

Η έξοδος του προγράμματος είναι η εμφάνιση στην οθόνη της πρότασης: **This is a test.**

Οι τρεις πρώτες γραμμές του ανωτέρω προγράμματος είναι **σχόλια** (comments) και δεν αποτελούν τμήμα του κώδικα, καθώς δεν λαμβάνονται υπόψη από τον μεταγλωττιστή. Το σχόλιο είναι κείμενο ανάμεσα σε **/*** και ***/** και μπορεί να τοποθετηθεί οπουδήποτε μέσα στο πρόγραμμα. Μπορεί να επεκταθεί σε περισσότερες από μία γραμμές. Σε περίπτωση που πρέπει να τοποθετηθεί ένα σχόλιο σε κάποιο σημείο μίας γραμμής και να μην επεκταθεί σε άλλη γραμμή, χρησιμοποιείται το σύμβολο **//** και ακολούθως τοποθετείται το σχόλιο, όπως συμβαίνει στο τέλος του ανωτέρω προγράμματος (το κείμενο **end of main** είναι σχόλιο).

Τα σχόλια πρέπει να χρησιμοποιούνται αφειδώς, γιατί καθιστούν τον κώδικα ευανάγνωστο και συνεισφέρουν στην επεξήγηση δυσνόητων σημείων. Είναι θεμιτό να ευθυγραμμίζονται τα σύμβολα των σχολίων και να μην τοποθετούνται ποτέ σχόλια μέσα σε σχόλια (ένθετα σχόλια), γιατί μπορεί να δημιουργηθεί πρόβλημα σε μερικούς μεταγλωττιστές.

Παρακάτω παρουσιάζονται μερικές περιπτώσεις σωστών και λανθασμένων σχολίων:

(α)
```
/* Αυτό /* το σχόλιο */ είναι λανθασμένο */
```
(β)
```
/*
      Αυτό το
        σχόλιο
είναι σωστό
        */
```

Παρατηρήσεις:

1. Στο παρόν σύγγραμμα ο όρος **εντολή** (command) αφορά *σε κάθε συντακτική οντότητα που είτε μεταβάλλει κάποια τιμή είτε επιτρέπει τη ροή πληροφορίας από και προς το εξωτερικό περιβάλλον*. Υπ' αυτήν την έννοια, οι συναρτήσεις θεωρούνται εντολές. Έτσι, π.χ. η συνάρτηση **`printf`** είναι μία εντολή, καθώς μεταφέρει πληροφορία στο εξωτερικό περιβάλλον.

2. Η γλώσσα C διαχωρίζει τα κεφαλαία γράμματα από τα μικρά (case sensitive). Η εντολή **`Printf`** δεν είναι ίδια με την **`printf`.** Όλες οι εντολές στη C γράφονται με μικρά γράμματα!

3. Η σωστή στηλοθεσία είναι πολύ σημαντική, καθώς καθιστά τον κώδικα ευανάγνωστο.

4. Η συνάρτηση **`printf()`** ανήκει στις **μορφοποιούμενες** συναρτήσεις εισόδου– εξόδου. Ονομάζεται μορφοποιούμενη, γιατί δίνει τη δυνατότητα στον χρήστη να μορφοποιήσει την έξοδό της, δυνάμενη να εκτυπώσει μεταβλητές διαφόρων τύπων και με διάφορους τρόπους, χρησιμοποιώντας κατάλληλα σύμβολα. Δυαδική της **`printf()`** είναι η **`scanf()`**, η οποία λαμβάνει πληροφορία από την είσοδο (πληκτρολόγιο).

Η πρόταση **`printf( "This is a test.\n" );`** καλεί την **`printf()`** για να τυπωθεί το καθορισμένο κείμενο. Τα **ορίσματα εισόδου** (input arguments) περικλείονται από παρενθέσεις και προσδιορίζουν το προς εκτύπωση κείμενο και τη μορφή με την οποία θα εκτυπωθεί. Τέλος, το σύμβολο **`\n`**, που ανήκει στις **ακολουθίες διαφυγής**, σημαίνει «μετακινήσου στην επόμενη γραμμή». Λεπτομερής περιγραφή της λειτουργίας των συναρτήσεων εισόδου– εξόδου δίνεται στο επόμενο κεφάλαιο.

5. Πέραν της **`include`**, μία σημαντική οδηγία προς τον προεπεξεργαστή είναι η **`define`**, η οποία αντιστοιχίζει ένα όνομα σε μία σταθερά ή σε μία σειρά χαρακτήρων (εκτελεί *μακροαντικατάσταση*). Οποτεδήποτε εμφανίζεται το όνομα μέσα στον κώδικα, αντικαθίσταται αυτόματα με την τιμή της σταθεράς ή τη συμβολοσειρά. Για παράδειγμα, εάν χρησιμοποιηθεί η λέξη **TRUE** στη θέση της τιμής **1** και η λέξη **FALSE** στη θέση της τιμής **0**, θα δοθούν δύο **`define`** ως εξής:

```
#define TRUE 1
#define FALSE 0
```

Εάν αντικατασταθεί ολόκληρη φράση, μπορεί να εμφανιστεί στην οθόνη με χρήση της **`printf`**:

```
#define TITLOS  "C Programming \ne-book\n"
printf( TITLOS );
```

Το αποτέλεσμα είναι:

```
C Programming
e-book
```

Μία συνηθισμένη χρήση της **`define`** είναι για τον καθορισμό του μεγέθους στοιχείων, όπως είναι η διάσταση ενός πίνακα, τα οποία μπορεί να χρησιμοποιηθούν επανειλημμένα μέσα στον κώδικα. Επιπρόσθετα, μέσω της **`define`** μπορούν να οριστούν εντολές, που ονομάζονται **μακροεντολές**. Τέλος, όπως θα αναλυθεί στο Κεφάλαιο 10, η **`define`** εμπλέκεται στην επονομαζόμενη *υπό συνθήκη μεταγλώττιση*.

1.6. Λεξιλόγιο της γλώσσας C

Από τα αναφερθέντα στην προηγούμενη παράγραφο γίνεται φανερό ότι η λειτουργία της γλώσσας C στηρίζεται σε ένα λεξιλόγιο. Το λεξιλόγιο της C περιλαμβάνει τέσσερις μείζονες κατηγορίες λέξεων:

1. Δεσμευμένες λέξεις
2. Λέξεις κλειδιά

3. Τελεστές
4. Αναγνωριστές

1.6.1. Δεσμευμένες λέξεις

Οι δεσμευμένες λέξεις (reserved words) χρησιμοποιούνται από τη γλώσσα C κατά τρόπο αποκλειστικό και πρέπει να αποφεύγεται η χρήση τους ως ονόματα. Αποτελούνται από:

- Ονόματα συναρτήσεων πρότυπης βιβλιοθήκης (runtime function names), όπως **`printf`**, **`abs`** κ.λπ.
- Μακροονόματα (macro names). Είναι ονόματα που περιέχονται σε αρχεία κεφαλίδας για ορισμό μακροεντολών, π.χ. **`EOF`**, **`INT_MAX`**.
- Ονόματα τύπων (type names). Είναι ονόματα τύπων σε ορισμένα αρχεία κεφαλίδας, π.χ. **`time_t`**, **`va_list`**.
- Ονόματα εντολών προεπεξεργαστή (preprocessor). Είναι ονόματα που χρησιμοποιεί προεπεξεργαστής της C και έχουν προκαθορισμένη σημασία, π.χ. **`include`**, **`define`**.
- Ονόματα που αρχίζουν με τον χαρακτήρα υπογράμμισης **`_`** και έχουν δεύτερο χαρακτήρα τον ίδιο ή κεφαλαίο γράμμα, π.χ. **`_DATE_`**, **`_FILE_`**.

1.6.2. Λέξεις κλειδιά

Οι λέξεις κλειδιά (keywords) είναι λεκτικές μονάδες, οι οποίες είτε μόνες τους είτε με άλλες λεκτικές μονάδες χαρακτηρίζουν κάποια γλωσσική κατασκευή. Π.χ. η λέξη **`int`** αναπαριστά τον ακέραιο τύπο δεδομένων και το ζεύγος **`if-else`** χρησιμοποιείται στον έλεγχο ροής προγράμματος.

Οι λέξεις κλειδιά, αν και είναι ένας περιορισμός των γλωσσών, αυξάνουν την αναγνωσιμότητα και αξιοπιστία των προγραμμάτων, ενώ ταυτόχρονα επιταχύνουν τη διαδικασία της μεταγλώττισης. Λέξεις κλειδιά, όπως **`if`**, **`else`**, **`for`**, **`case`**, **`while`**, **`do`**, έχουν γίνει κοινά αποδεκτές, διευκολύνοντας την εκμάθηση των γλωσσών προγραμματισμού.

Στον **Πίνακα 1.1** παρατίθενται οι λέξεις κλειδιά της γλώσσας C:

`auto`	**`break`**	**`case`**	**`char`**	**`const`**	**`continue`**	**`default`**	**`do`**
`double`	**`else`**	**`enum`**	**`extern`**	**`float`**	**`for`**	**`goto`**	**`if`**
`int`	**`long`**	**`register`**	**`return`**	**`short`**	**`signed`**	**`sizeof`**	**`static`**
`struct`	**`switch`**	**`typedef`**	**`union`**	**`unsigned`**	**`void`**	**`volatile`**	**`while`**

Πίνακας 1.1 *Λέξεις κλειδιά της γλώσσας C*

1.6.3. Αναγνωριστές

Οι αναγνωριστές (identifiers) είναι λεκτικές μονάδες που κατασκευάζει ο προγραμματιστής. Αυτές οι λεκτικές μονάδες χρησιμοποιούνται συνήθως ως ονόματα που ο προγραμματιστής δίνει σε δικές του κατασκευές, όπως μεταβλητές, σταθερές, συναρτήσεις και δικούς του τύπους δεδομένων. Ένα όνομα προσδιορίζει μοναδιαία, από το σύνολο των κατασκευών του προγράμματος, την κατασκευή στην οποία αποδόθηκε, εξ ου και το όνομα αναγνωριστής. Περισσότερα στοιχεία για τους αναγνωριστές σημειώνονται στο Κεφάλαιο 2.

1.7. Κανόνες δημιουργίας ευανάγνωστων προγραμμάτων

- Θα πρέπει να αποφεύγονται ονόματα ενός χαρακτήρα, όπως **`i`**, **`j`**, **`x`**, **`y`** (εκτός από ειδικές περιπτώσεις που θα αναφερθούν στα επόμενα κεφάλαια).

- Το όνομα που χρησιμοποιείται θα πρέπει να είναι ενδεικτικό του τι αναπαριστά ή διαχειρίζεται η μεταβλητή. Συγκεκριμένα:

    1. Η μεταβλητή που αναπαριστά τον όγκο θα μπορούσε να ονομαστεί **`volume`** και η μέγιστη τιμή της **`max_volume`** ή **`maxVolume`**.

    2. Η συνάρτηση που εμφανίζει στην οθόνη πληροφορίες για τη λειτουργία του προγράμματος μπορεί να λάβει το ενδεικτικό όνομα **`printProgramInfo`**.

- Σε ολόκληρο τον κώδικα ενός προγράμματος θα πρέπει να διατηρείται ο ίδιος φορμαλισμός στην ονοματοδοσία. Κατά συνέπεια, εάν χρησιμοποιηθεί η σύμβαση σύζευξης δύο ή περισσότερων λέξεων σε ένα όνομα μεταβλητής μέσω του χαρακτήρα υπογράμμισης **`_`** (π.χ. **`string_name`**), δεν θα πρέπει να χρησιμοποιηθεί άλλος τρόπος σύζευξης (π.χ. **`stringName`**) αλλά να διατηρηθεί ο αρχικός (δηλαδή **`string_name)`**.

- Θα πρέπει να χρησιμοποιούνται μικρά γράμματα για ονόματα μεταβλητών και κεφαλαία γράμματα για χρήση συνωνύμων μέσω της εντολής προεπεξεργαστή **`define`**.

Ερωτήσεις αυτοαξιολόγησης

(1) Η γλώσσα C αποτελεί:
(α) Γλώσσα προγραμματισμού υψηλού επιπέδου
(β) Γλώσσα μηχανής
(γ) Γλώσσα assembly
(δ) Τίποτε από τα προηγούμενα

(2) Ποια είναι η σωστή σειρά εκτέλεσης των ακόλουθων βημάτων;
(α) i. Μεταγλώττιση και δημιουργία του αρχείου αντικειμένου
ii. Δημιουργία του εκτελέσιμου κώδικα
iii. Συγγραφή του πηγαίου κώδικα
iv. Προεπεξεργασία και ενσωμάτωση των αρχείων κεφαλίδας
(β) i. Συγγραφή του πηγαίου κώδικα
ii. Προεπεξεργασία και ενσωμάτωση των αρχείων κεφαλίδας
iii. Μεταγλώττιση και δημιουργία του αντικείμενου κώδικα
iv. Δημιουργία του εκτελέσιμου κώδικα
(γ) i. Προεπεξεργασία και ενσωμάτωση των αρχείων κεφαλίδας
ii. Συγγραφή του πηγαίου κώδικα
iii. Μεταγλώττιση και δημιουργία του αρχείου αντικειμένου
iv. Δημιουργία του εκτελέσιμου κώδικα
(δ) i. Μεταγλώττιση και δημιουργία του αρχείου αντικειμένου
ii. Συγγραφή του πηγαίου κώδικα
iii. Προεπεξεργασία και ενσωμάτωση των αρχείων κεφαλίδας
iv. Δημιουργία του εκτελέσιμου κώδικα

(3) Ποιο από τα ακόλουθα σχόλια είναι σωστό;
(α)
```
/* Σχόλιο σχόλιο σχόλιο σχόλιο
```
(β)
```
/* Σχόλιο σχόλιο /* σχόλιο σχόλιο */
```
(γ)
```
/* Σχόλιο
   σχόλιο  σχόλιο
   σχόλιο  */
```
(δ)
```
/* Σχόλιο σχόλιο σχόλιο σχόλιο    //
```

(4) Το όνομα **`typedef`** αποτελεί:
(α) Λέξη κλειδί

(β) Τελεστή
(γ) Αναγνωριστή
(δ) Δεσμευμένη λέξη

Βιβλιογραφία κεφαλαίου

Θραμπουλίδης, Κ. (2002), *Διαδικαστικός Προγραμματισμός - C (Τόμος Α)*, 2η έκδοση, Εκδόσεις Τζιόλα.

Χατζηγιαννάκης, Ν. (2012), *Η Γλώσσα C σε Βάθος*, 4η Έκδοση, Εκδόσεις Κλειδάριθμος.

Deitel, H. & Deitel, P. (2014), *C Προγραμματισμός*, 7η έκδοση, Εκδόσεις Γκιούρδα.

Gabrielli, M. & Martini, S. (2010), *Programming Languages: Principles and Paradigms*, Springer.

Horton, I. (2006), *Beginning C – from Novice to Professional*, 4th ed., Apress.

Kernighan, B. & Ritchie, D. (1990), *Η γλώσσα προγραμματισμού C*, Εκδόσεις Κλειδάριθμος.

Lohr, S. (2001), *Go To*, Basic Books.