ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

Δομές Δεδομένων (Εργ.)
Ακ. Έτος 2018-19
Διδάσκων: Ευάγγελος Σπύρου

Επανάληψη Α' μέρος

1. Στόχος του εργαστηρίου

Στόχος του πρώτου εργαστηρίου είναι η επανάληψη βασικών γνώσεων στη γλώσσα C. Όλες οι δομές δεδομένων που θα διδαχθούν στο εργαστήριο, θα αναπτυχθούν σε C. Για το λόγο αυτό, απαιτείται εξοικείωση με τις βασικές έννοιες προγραμματισμού που έχουν ήδη διδαχθεί.

2. Εισαγωγή στη γλώσσα C

Η γλώσσα C δημιουργήθηκε στις αρχές της δεκαετίας 1970 από τον Dennis Richie. Αποτελεί μια γλώσσα υψηλού επιπέδου και παραμένει ιδιαίτερα δημοφιλής. Είναι μια ευέλικτη γλώσσα και μπορεί να χρησιμοποιηθεί για την ανάπτυξη εφαρμογών σε διάφορα πεδία. Εμφανίζει αρκετά πλεονεκτήματα, όπως π.χ., ότι τα προγράμματα που είναι γραμμένα σε C εκτελούνται ιδιαίτερα γρήγορα, σε σχέση με άλλες γλώσσες προγραμματισμού, έχει μεταφέρσιμο κώδικα, υποστηρίζει τον δομημένο προγραμματισμό και βρίσκεται κοντά στο υλικό. Υπάρχουν αρκετές έτοιμες συναρτήσεις που μπορεί να χρησιμοποιηθούν από τον προγραμματιστή, καθώς και πολλοί μεταγλωττιστές, για διάφορα λειτουργικά συστήματα, αρκετοί από τους οποίους διανέμονται δωρεάν. Φυσικά, έχει και κάποια μειονεκτήματα, συγκρινόμενη με άλλες γλώσσες προγραμματισμού. Είναι σχετικά δύσκολη στην εκμάθηση, αρκετά λάθη του προγραμματιστή δεν ανιχνεύονται από τον μεταγλωττιστή και δεν είναι αντικειμενοστρεφής.

3. Δημιουργία προγράμματος στη C

Η δημιουργία ενός προγράμματος σε C περνάει από διάφορα στάδια. Αρχικά, γίνεται η συγγραφή του προγράμματος, στη συνέχεια η μεταγλώττιση (compile) και η σύνδεση του μεταγλωττισμένου κώδικα με αυτόν των συναρτήσεων βιβλιοθήκης που χρησιμοποιούνται (link). Τέλος, το πρόγραμμα εκτελείται (run). Συνήθως χρησιμοποιείται ένα περιβάλλον ανάπτυξης (όπως π.χ., το Dev-C++), στο οποίο υποστηρίζονται όλες οι παραπάνω λειτουργίες μέσω ενός απλού γραφικού περιβάλλοντος. Υπενθυμίζουμε ότι ο κώδικας του προγράμματος μπορεί να γραφτεί με οποιονδήποτε επεξεργαστή κειμένου και πρέπει να αποθηκεύεται σε αρχεία `.c`. Αν το πρόγραμμα μεταγλωττιστεί επιτυχώς, παράγεται ένα αρχείο αντικειμένου (object file) `.o`, το οποίο και περιέχει τον κώδικα του προγράμματος μεταφρασμένο σε γλώσσα μηχανής.

4. Το πρώτο πρόγραμμα σε C

Το πρώτο πρόγραμμα σε C είναι συνήθως το παρακάτω:

```
#include <stdio.h>
int main(void)
{
     printf("Hello, World!");
     return 0;
```

```
}
```

Υπενθυμίζουμε τη σημασία της κάθε γραμμής: Αρχικά, η οδηγία `#include` υποχρεώνει το μεταγλωττιστή να συμπεριλάβει τα περιεχόμενα του αρχείου `stdio.h` στον κώδικα του προγράμματος. Το αρχείο αυτό περιέχει πληροφορίες για συναρτήσεις βιβλιοθήκης και παρέχεται με το μεταγλωττιστή. Ονομάζεται αρχείο επικεφαλίδας (header file). Συγκεκριμένα, το `stdio.h` περιέχει δηλώσεις συναρτήσεων που σχετίζονται με την εμφάνιση και το διάβασμα δεδομένων (output/input).

Κάθε πρόγραμμα σε C πρέπει υποχρεωτικά να έχει μια συνάρτηση `main()`, η οποία και αποτελεί την κύρια συνάρτηση κάθε προγράμματος. Καλείται αυτόματα, όταν εκτελεστεί το πρόγραμμα. Η εκτέλεση του προγράμματος αρχίζει με την πρώτη της εντολή και τερματίζει με την τελευταία, εκτός κι αν κληθεί νωρίτερα κάποια εντολή εξόδου, π.χ., η `return`. Η ειδική λέξη `int` που προηγείται του ονόματος της `main()` σημαίνει ότι πρέπει να επιστραφεί κάποιος ακέραιος αριθμός, όταν αυτή τερματίσει.

5. Η συνάρτηση `printf()`

Η `printf()` είναι μια συνάρτηση βιβλιοθήκης της C, η οποία και χρησιμοποιείται για την εμφάνιση δεδομένων στην οθόνη. Δέχεται μια μεταβλητή λίστα παραμέτρων. Η πρώτη παράμετρος είναι ένα αριθμητικό μορφοποίησης (format string), δηλαδή μια ακολουθία χαρακτήρων μέσα σε διπλά εισαγωγικά, η οποία και καθορίζει τον τρόπο με τον οποίο θα εμφανιστούν τα δεδομένα στην οθόνη. Οι επόμενες παράμετροι είναι προαιρετικές. Αν υπάρχουν, η printf() εμφανίζει το αποτέλεσμά τους στην οθόνη. Το αλφαριθμητικό μορφοποίησης μπορεί να περιέχει ακολουθίες διαφυγής, προσδιοριστικά μετατροπής και απλούς χαρακτήρες. Οι τελευταίοι εμφανίζονται στην οθόνη χωρίς να υποστούν καμία μετατροπή. Αποτελεί ευθύνη του προγραμματιστή να καθορίσει ένα έγκυρο αλφαριθμητικό μορφοποίησης.

Οι ακολουθίες διαφυγής φαίνονται στον παρακάτω πίνακα:

Ακολουθία Διαφυγής	**Μετάφραση**
`\a`	Ηχητικό σήμα
`\b`	Διαγραφή χαρακτήρα (backspace)
`\f`	Αλλαγή σελίδας
`\n`	Αλλαγή γραμμής
`\r`	Επαναφορά δρομέα στην αρχή της γραμμής
`\t`	Οριζόντιος στηλοθέτης (tab)
`\v`	Κατακόρυφος στηλοθέτης
`\\`	Ανάστροφη κάθετος
`\'`	Μονό εισαγωγικό
`\"`	Διπλά εισαγωγικά
`\?`	Λατινικό ερωτηματικό

Άσκηση 1.1

Να γραφεί ένα πρόγραμμα το οποίο να χρησιμοποιεί μία μόνο `printf()` για να εμφανίσει την ακόλουθη έξοδο.

```
*   *
  *
*   *
```

```
#include <stdio.h>
int main(void)
{
     printf("*   *\n  * \n*   *\n");
     return 0;
}
```

6. Προσδιοριστικά μετατροπής

Ένα προσδιοριστικό μετατροπής αρχίζει με το χαρακτήρα % και ακουλουθείται από έναν ή περισσότερους χαρακτήρες με ειδική σημασία. Στην απλή μορφή, το % ακουλουθείται από έναν από τους παρακάτω χαρακτήρες:

Χαρακτήρας Μετατροπής	Σημασία
`c`	Εμφάνιση του χαρακτήρα που αντιστοιχεί σε απρόσημη ακέραια τιμή
`d` ή `i`	Εμφάνιση ενός ακεραίου
`u`	Εμφάνιση ενός απρόσημου ακεραίου
`f`	Εμφάνιση ενός πραγματικού αριθμού, εξ ορισμού ακρίβεια 6 δεκαδικών ψηφίων
`s`	Εμφάνιση χαρακτήρων ενός αλφαριθμητικού
`e` ή `E`	Εμφάνιση ενός πραγματικού αριθμού σε επιστημονική μορφή
`g` ή `G`	Χρησιμοποιείται `%e` ή `%E` αν ο εκθέτης είναι μικρότερος από -4 ή μεγαλύτερος από ή ίσος με την ακρίβεια. Αλλιώς χρησιμοποιείται `%f`.
`p`	Εμφάνιση της τιμής ενός δείκτη
`x` ή `X`	Εμφάνιση ενός απρόσημου ακεραίου σε δεκαεξαδική μορφή. Με το `%x` τα δεκαεξαδικά ψηφία (A-F) εμφανίζονται πεζά, με το `%X` κεφαλαία
`o`	Εμφάνιση απρόσημου αριθμού σε οκταδική μορφή
`n`	Δεν εκτυπώνεται κάτι. Το αντίστοιχο όρισμα είναι δείκτης σε ακέραιο, στον οποίο θα αποθηκευθεί ο αριθμός των χαρακτήρων που έχουν εκτυπωθεί μέχρι τώρα
`%`	Εμφάνιση του χαρακτήρα `%`

Άσκηση 1.2

Τρέξτε το παρακάτω πρόγραμμα και προσπαθήστε να κατανοήσετε τη χρήση των προσδιοριστικών μετατροπής:

```
#include <stdio.h>
int main(void)
{
     int len;

     printf("%c\n", 'w');
     printf("%d\n", -100);
     printf("%f\n", 1.56);
     printf("%s\n", "some text");
     printf("%e\n", 100.25);
     printf("%g\n", 0.0000123);
     printf("%X\n", 15);
     printf("%o\n", 14);
     printf("test%n\n", &len);
     printf("%d%%\n", 100);

     return 0;
}
```

7. Τύποι μεταβλητών στη C

Η γλώσσα C υποστηρίζει αρκετούς τύπους μεταβλητών, οι οποίοι και συνοψίζονται στον ακόλουθο πίνακα, για ένα 32bit υπολογιστικό σύστημα:

Τύπος	**Συνηθισμένο Μέγεθος (bytes)**	**Εύρος τιμών (min-max)**	**Ψηφία ακρίβειας**
char	1	-128...127	
short int	2	-32.768...32.767	
int	4	-2.147.483.648 ... 2.147.483.647	
long int	4	-2.147.483.648 ... 2.147.483.647	
float	4	Μικρότερη θετική τιμή: $1.17*10^{-38}$ Μεγαλύτερη θετική τιμή: $3.4*10^{38}$	6
double	8	Μικρότερη θετική τιμή: $2.2*10^{-308}$ Μεγαλύτερη θετική τιμή: $1.8*10^{308}$	15
long double	8, 10, 12, 16		
unsigned char	1	0...255	
unsigned short int	2	0...65535	
unsigned int	4	0... 4.294.967.295	
unsigned long int	4	0...4.294.967.295	

Θυμίζουμε ότι οι μεταβλητές δηλώνονται ως εξής:

```
τύπος_δεδομένων όνομα_μεταβλητής;
```

Για να μάθουμε το σωστό μέγεθος μιας μεταβλητής, καθώς αυτό μπορεί να διαφέρει από σύστημα σε σύστημα, χρησιμοποιείται ο τελεστής `sizeof`, τον οποίο θα δούμε αργότερα.

Άσκηση 1.3
Βρείτε τα λάθη στο παρακάτω πρόγραμμα και να τα διορθώσετε, ώστε το πρόγραμμα να εκτελείται και να εμφανίζει την τιμή της μεταβλητής `m`.

```
#include <stdio.h>
int mein(void);
{
     int m;
     a = 10
     m = 2a+100
     print(%f\n", M);
     return0;
)
```

Άσκηση 1.4
Βρείτε τα λάθη στο παρακάτω πρόγραμμα και να τα διορθώσετε, ώστε το πρόγραμμα να εκτελείται και να εμφανίζει τις τιμές των μεταβλητών `a` , `b` και `c`.

```
$include <studio.h>
INT main(void)
[
     int b = a/2; a = 10
     float c = 4,5 + a;
     printf("%d %d\n", a, c, b);
     retern 0;
]
```

Άσκηση 1.5
Να γραφεί ένα πρόγραμμα το οποίο να εκχωρεί δύο αρνητικές τιμές σε δύο ακέραιες μεταβλητές και να χρησιμοποιεί αυτές τις μεταβλητές για να εμφανίσει τις αντίστοιχες θετικές.

Άσκηση 1.6
Να συμπληρώσετε τα κενά, με κατάλληλο τρόπο, έτσι ώστε το πρόγραμμα να εμφανίζει την ακόλουθη έξοδο:

```
21
 21
15
25%
A
 a
10
77
077
63
```

```
#include <stdio.h>
int main(void)
```

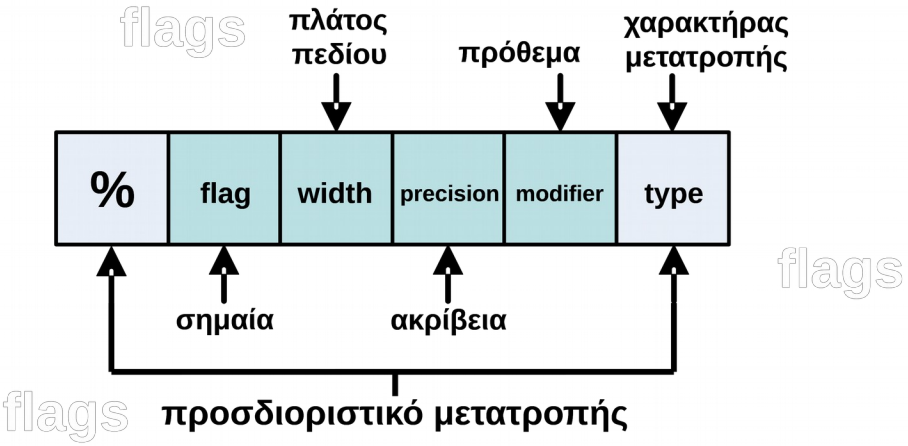
```
{
      int x = 21, y = 0xa, z = 077;

      printf("__________\n", x);
      printf("__________\n", x);
      printf("__________\n", x);
      printf("__________\n", x);
      printf("__________\n", y);
      printf("__________\n", y);
      printf("__________\n", y);
      printf("__________\n", z);
      printf("__________\n", z);
      printf("__________\n", z);
      return 0;
}
```

Άσκηση 1.7

Να γραφεί ένα πρόγραμμα το οποίο να εκχωρεί δύο θετικές πραγματικές τιμές σε δύο πραγματικές μεταβλητές και να εμφανίζει το ακέραιο μέρος του πηλίκου τους και το υπόλοιπο της διαίρεσής τους με τη μορφή δεκαδικού αριθμού. Για παράδειγμα, αν εκχωρηθούν οι τιμές `7.2` και `5.4` , το πρόγραμμα να εμφανίζει `1` και `1.8` , αφού 7.2 = (1×5.4)+1.8 .

9. Εμφάνιση μεταβλητών στην `printf()`

Τα ονόματα των μεταβλητών εισάγονται στην `printf()` μετά το δεξί διπλό εισαγωγικό. Αν είναι περισσότερες από μία, πρέπει να διαχωρίζονται μεταξύ τους με κόμμα. Ο μεταγλωττιστής αντιστοιχίζει ένα-προς-ένα από αριστερά προς τα δεξιά τα ονόματα των μεταβλητών με τα προσδιοριστικά μετατροπής. Κάθε προσδιοριστικό μετατροπής θα πρέπει να ταιριάζει με τον τύπο της μεταβλητής, αλλιώς το πρόγραμμα θα εμφανίσει μια τυχαία τιμή! Προσοχή: ο μεταγλωττιστής δεν ελέγχει αν ο αριθμός των προσδιοριστικών μετατροπής είναι ίσος με τον αριθμό των μεταβλητών. Στην απλή μορφή του, ένα προσδιοριστικό μετατροπής αρχίζει με το χαρακτήρα `%` και ακολουθείται από κατάλληλο χαρακτήρα μετατροπής. Όμως μπορεί να περιέχονται ανάμεσά τους εώς και 4 προαιρετικά πεδία. Δείτε το παρακάτω σχήμα:



Η σημασία της ακρίβειας εξαρτάται από τον χαρακτήρα μετατροπής:

- `d`, `i`, `o`, `u`, `x`, `X`: καθορίζει τον ελάχιστο αριθμό ψηφίων που θα εκτυπωθούν
- `e`, `E`, `f`: καθορίζει τον αριθμό των δεκαδικών ψηφίων

- `g`, `G`: καθορίζει τον μέγιστο αριθμό των σημαντικών ψηφίων
- `s`: καθορίζει τον μέγιστο αριθμό των χαρακτήρων που θα εκτυπωθούν

Η ακρίβεια δηλώνεται με μια τελεία που ακολουθείται είτε από έναν ακέραιο, είτε από τον χαρακτήρα * και έναν ακέραιο στη λίστα των μεταβλητών. Το παρακάτω πρόγραμμα εμφανίζει την τιμή μιας πραγματικής μεταβλητής με διαφορετικές ακρίβειες. Η εξ ορισμού επιλογή είναι να εμφανίζονται έξι δεκαδικά ψηφία. Η τιμή που εμφανίζεται στρογγυλοποιείται προς τα πάνω ή προς τα κάτω, ανάλογα με το αν η τιμή του πρώτου ψηφίου που αποκόπτεται είναι μεγαλύτερη ή όχι από το 4, αντίστοιχα.

```
#include<stdio.h>
int main(void)
{
     float a = 1.2365;

     printf("%f\n", a);
     printf("%.2f\n", a);
     printf("%.*f\n", 3, a);
     printf("%.f\n", a);

     return 0;
}
```

Υπενθυμίζεται ότι όταν χρησιμοποιούμε μια πραγματική μεταβλητή σε μαθηματικές εκφράσεις, είναι ασφαλέστερος ο τύπος `double`.

Όταν εμφανίζουμε την τιμή μιας αριθμητικής μεταβλητής, μπορούμε να καθορίσουμε το συνολικό πλήθος των χαρακτήρων που θα εμφανιστούν, είτε εισάγοντας έναν ακέραιο που να δηλώνει το πλάτος του πεδίου είτε τον χαρακτήρα * και έναν ακέραιο στη λίστα των μεταβλητών που να το δηλώνει. Αν η τιμή της μεταβλητής χρειάζεται λιγότερους χαρακτήρες από το δηλωμένο πλάτος, τότε στην έξοδο προστίθενται κενοί χαρακτήρες από αριστερά προς τα δεξιά, μέχρι να συμπληρωθεί το συνολικό πλήθος των χαρακτήρων. Αν χρειάζεται περισσότερους, το πλάτος του πεδίου δε λαμβάνεται υπόψη και η τιμή της μεταβλητής εμφανίζεται με όσους χαρακτήρες χρειάζεται. Δείτε π.χ., το παρακάτω πρόγραμμα και προσπαθήστε να κατανοήσετε την έξοδο που παράγεται.

```
#include <stdio.h>
int main(void)
{
     int a = 100;
     float b = 1.2365;

     printf("%10d\n", a);
     printf("%10f\n", b);
     printf("%10.3f\n", b);
     printf("%*.3f\n", 6, b);
     printf("%2d\n", a);
     printf("%6f\n", b);
}
```

Όσον αφορά, το πρόθεμα, στην εμφάνιση ενός ακεραίου, το γράμμα `h` δηλώνει ότι ο ακέραιος είναι short, το γράμμα `l` ότι είναι long. Με τα `e`, `E`, `f`, `g`, `G` το γράμμα `L` δηλώνει ότι ο ακέραιος είναι `double`.

Τέλος, οι σημαίες χρησιμοποιούνται για περαιτέρω μορφοποίηση των εμφανιζόμενων τιμών. Δείτε τον πίνακα που ακολουθεί.

Σημαία	Σημασία
-	Η έξοδος στο πεδίο πλάτους στοιχίζεται αριστερά (εξ ορισμού, η έξοδος στοιχίζεται δεξιά)
+	Προσθέτει το πρόσημο + μπροστά από τις θετικές τιμές
Κενός χαρακτήρας	Προσθέτει τον κενό χαρακτήρα μπροστά από τις θετικές τιμές
#	Προσθέτει το 0 μπροστά από τις οκταδικές τιμές και το 0x ή το 0X μπροστά από τις δεκαεξαδικές τιμές
0	Προσθέτει όσα μηδενικά χρειάζονται μπροστά από την εμφανιζόμενη τιμή, ώστε να καλυφθεί το πεδίο πλάτους

Δείτε π.χ., το παρακάτω πρόγραμμα και προσπαθήστε να κατανοήσετε την έξοδο που παράγεται.

```
#include<stdio.h>
int main(void)
{
     int a = 12;
     printf("%-4d\n", a);
     printf("%+4d\n", a);
     printf("% d\n", a);
     printf("%#0x\n", a);
     printf("%#o\n", a);
     printf("%04d\n", a);

     return 0;
}
```

10. Είσοδος δεδομένων

Η συνάρτηση `scanf()` χρησιμοποιείται για το διάβασμα δεδομένων από ένα ρεύμα εισόδου το οποίο ονομάζεται `stdin` (standard input stream). Εξ ορισμού, συνδέεται με το πληκτρολόγιο. Η `scanf()` δέχεται μια μεταβλητή λίστα παραμέτρων. Παρόμοια με την `printf()`, η πρώτη παράμετρος είναι ένα αλφαριθμητικό μορφοποίησης, ενώ οι επόμενες προαιρετικές παράμετροι είναι οι διευθύνσεις της μνήμης των μεταβλητών στις οποίες θα εκχωρηθούν τα δεδομένα που θα εισάγει ο χρήστης. Συνήθως, το αλφαριθμητικό μορφοποίησης περιέχει μόνο απλά προσδιοριστικά μετατροπής. Οι χαρακτήρες μετατροπής που χρησιμοποιούνται στη `scanf()` είναι αυτοί που είδαμε για την `printf()`. Για παράδειγμα, για το διάβασμα ενός ακεραίου χρησιμοποιείται το `%d`. Όπως και με την `printf()`, το προσδιοριστικό μετατροπής πρέπει να αντιστοιχεί σε μια

μεταβλητή και να ταιριάζει με τον τύπο της. Π.χ., για να διαβάσουμε έναν ακέραιο από το πληκτρολόγιο και να τον αποθηκεύσουμε στη μεταβλητή `i`, θα γράφαμε το παρακάτω:

```
int i;
scanf(“%d”, &i);
```

Ο τελεστής μεταβλητής & τοποθετείται πριν το όνομα της μεταβλητής και χρησιμοποιείται για να αποθηκευθεί ο αριθμός που θα εισάγει ο χρήστης στη διεύθυνση μνήμης της μεταβλητής. Αν θέλαμε να διαβάσουμε δύο μεταβλητές διαφορετικού τύπου, θα τροποποιούσαμε το προηγούμενο παράδειγμα ως εξής:

```
int i;
float j;
scanf(“%d%f”, &i, &j);
```

Άσκηση 1.8
Να γραφεί ένα πρόγραμμα το οποίο να διαβάζει έναν χαρακτήρα, έναν ακέραιο και έναν πραγματικό αριθμό και να τους εμφανίζει. Χρησιμοποιείστε μία `scanf()`.

```
#include<stdio.h>
int main(void)
{
     char ch;
     int i;
     float f;

     printf("Enter character, int and float:");
     scanf("%c%d%f", &ch, &i, &f);
     printf("\nChar:%c\tInt:%d\tFloat:%f", ch, i, f);

     return 0;
}
```

Άσκηση 1.9
Να γραφεί ένα πρόγραμμα το οποίο να διαβάζει δύο ακεραίους και να εμφανίζει το άθροισμα, τη διαφορά, το γινόμενο και το αποτέλεσμα της διαίρεσής τους.

```
#include<stdio.h>
int main(void)
{
     int i, j;

     printf("Enter two integers. The second should not be 0:");
     scanf("%d%d", &i, &j);
```

```
    printf("Sum:%d\tDiff:%d\tProd:%d\tDiv:%f",  i+j,  i-j,  i*j,
(double)i/j);

    return 0;
}
```

Προσέξτε το casting που κάναμε στην περίπτωση της διαίρεσης, προκειμένου το αποτέλεσμα να εμφανιστεί σωστά. Τι θα γινόταν αν το παραλείπαμε;

Άσκηση 1.10
Να γραφεί ένα πρόγραμμα το οποίο να διαβάζει τις τιμές τριών προϊόντων και να εμφανίζει τη μέση τιμή τους.

Άσκηση 1.11
Να γραφεί ένα πρόγραμμα το οποίο να διαβάζει την ακτίνα ενός κύκλου και να εμφανίζει το εμβαδό και την περίμετρο του κύκλου.

Άσκηση 1.12
Να γραφεί ένα πρόγραμμα το οποίο να διαβάζει έναν θετικό πραγματικό και να εμφανίζει τον προηγούμενο και τον επόμενο ακέραιο.

Άσκηση 1.13
Να γραφεί ένα πρόγραμμα το οποίο να διαβάζει δύο θετικούς πραγματικούς και να εμφανίζει το άθροισμα των ακέραιων και των δεκαδικών τμημάτων τους. Για παράδειγμα, αν ο χρήστης εισάγει τις τιμές `1.23` και `9.56`, το πρόγραμμα να εμφανίζει `10` και `79`.

Άσκηση 1.14
Να γραφεί ένα πρόγραμμα το οποίο να διαβάζει έναν τετραψήφιο θετικό ακέραιο και έναν μονοψήφιο θετικό ακέραιο. Στη συνέχεια το πρόγραμμα να μετατρέπει τον τετραψήφο αριθμό σε πενταψήφιο, τοποθετώντας στη μέση τον μονοψήφιο. Για παράδειγμα, αν ο χρήστης εισάγει τις τιμές `1234` και `5`, ο αριθμός να μετατρέπεται σε `12534`.

11. Τελεστές
Έχοντας μέχρι τώρα δει πώς μπορούμε να δηλώνουμε και να χρησιμοποιούμε μεταβλητές σε απλές εκφράσεις, στη συνέχεια θα δούμε πώς μπορούμε να δημιουργήσουμε σύνθετες εκφράσεις με τη βοήθεια των τελεστών που παρέχει η C.

Ο τελεστής = χρησιμοποιείται για την απόδοση τιμής σε μια μεταβλητή. Για παράδειγμα, με την εντολή `a=10;` η τιμή της μεταβλητής `a` γίνεται ίση με `10`, ενώ με την εντολή `a=k;` η τιμή της μεταβλητής `a` γίνεται ίση με την τιμή της μεταβλητής `k`. Αν ο τελεστής = χρησιμοποιείται διαδοχικά σε μια εντολή εκχώρησης, τότε η τελική τιμή εκχώρησης αποθηκεύεται σε όλες τις μεταβλητές. Για παράδειγμα με την εντολή:

```
int a, b, c;
a = b = c = 10;
```

Οι τιμές των μεταβλητών a, b, c γίνονται ίσες με 10. Ο τελεστής = έχει δεξιά συσχέτιση, άρα οι αναθέσεις γίνονται από δεξιά προς τα αριστερά. Άρα πρώτα το c γίνεται 10, μετά το b γίνεται ίσο με το c, και μετά το a γίνεται ίσο με το b, άρα επίσης 10.

Μια εντολή εκχώρησης π.χ., a = 10 παράγει ένα αποτέλεσμα που είναι ίσο με την τιμή του a μετά την εκχώρηση. Έτσι, για παράδειγμα:

```
a = 10;
c = a + (b = a+10);
```

Το αποτέλεσμά της εκχώρησης b = a+10 είναι 20 και η τιμή του c γίνεται ίση με 30. Ωστόσο, συστήνεται οι εκχωρήσεις να γίνονται ξεχωριστά και να παίρνουμε πιο ευανάγνωστο και καλύτερα ελεγχόμενο κώδικα.

Οι αριθμητικοί τελεστές +, −, *, / χρησιμοποιούνται για την εκτέλεση των γνωστών μαθηματικών πράξεων. Όταν και οι δύο τελεστέοι είναι ακέραιοι, ο τελεστής / παρέχει το πηλίκο της ακέραιας διαίρεσης. Άρα, αποκόπτει το δεκαδικό μέρος του πηλίκου. Π.χ., αν θεωρήσουμε τις ακέραιες μεταβλητές a, b με τιμές 3 και 2, αντίστοιχα, τότε το αποτέλεσμά της πράξης a/b είναι 1 και όχι 1.5. Αν όμως ένας από τους δύο τελεστέους είναι πραγματική μεταβλητή ή σταθερά, το δεκαδικό μέρος δεν αποκόπτεται. Για παράδειγμα, το αποτέλεσμα της πράξης a/5.0 είναι 0.6, γιατί η τελεία σε μια σταθερά υποδηλώνει ότι η σταθερά είναι πραγματικός αριθμός. Για να υπολογίσουμε το υπόλοιπο μιας διαίρεσης ακεραίων χρησιμοποιούμε τον τελεστή %. Άρα το αποτέλεσμα της πράξης a%b είναι 1. Προσοχή: αν η τιμή του δεξιού τελεστέου στην πράξη διαίρεσης ή υπολοίπου είναι 0, τότε το αποτέλεσμα είναι απροσδιόριστο και το πρόγραμμα ενδέχεται να τερματιστεί.

Οι προηγούμενοι τελεστές ονομάζεται δυαδικοί (binary) γιατί απαιτούν δύο τελεστέους. Υπάρχουν όμως και οι μοναδιαίοι (unary) τελεστές, οι οποίοι εφαρμόζονται σε έναν τελεστέο. Π.χ.:

```
int a = +20; // το + δεν έχει κάποια επίδραση
a = -10; // το - χρησιμοποιείται σαν μοναδιαίος τελεστής
int b = -(a-5); /* το εσωτερικό - χρησιμοποιείται σαν δυαδικός τελεστής, το εξωτερικό σαν μοναδιαίος */
-a; // Η εκτέλεση της εντολης δεν επηρεάζει την τιμή του a
```

Η τιμή του b γίνεται τελικά ίση με 15, ενώ του a παραμένει −10.

Ο τελεστής ++ εισάγεται πριν ή μετά από την τιμή ενός τελεστέου και αυξάνει την τιμή του κατά ένα. Φυσικά, ο τελεστέος πρέπει να είναι μια τροποποιήσιμη τιμή, π.χ., μια μεταβλητή. Στο παρακάτω παράδειγμα:

```
int a = 4;
a++; // η τιμή του a αυξάνεται κατά ένα
++a; // η τιμή του a αυξάνεται κατά ένα
(a+10)++; // λάθος!!!
```

Η τιμή του a αυξάνεται δύο φορές κατά ένα και γίνται ίση με 6. Όταν ο τελεστής ++ εισάγεται μετά το όνομα της μεταβλητής, τότε χρησιμοποιείται η τρέχουσα τιμή της μεταβλητής και μετά αυτή αυξάνεται κατά ένα. Π.χ.:

```
int a = 4, b;
b = a++;
```

πρώτα αποθηκεύεται στη μεταβλητή b η τρέχουσα τιμή του a και έπειτα η τιμή του a αυξάνεται. Άρα η τιμή του b θα είναι 4 και του a θα είναι 5. Όταν όμως ο τελεστής ++ εισάγεται πριν από το όνομα της μεταβλητής, τότε πρώτα αυξάνεται η τιμή της μεταβλητής κατά ένα και μετά αυτή χρησιμοποιείται. Π.χ.:

```
int a = 4, b;
b = ++a;
```

πρώτα αυξάνεται η τιμή του a κατά ένα και έπειτα το b γίνεται ίσο με την (νέα) τιμή του a. Άρα και οι δύο μεταβλητές θα έχουν την τιμή 5. Ο τελεστής -- λειτουργεί με τον ίδιο τρόπο όπως και ο ++, μόνο που μειώνει την τιμή της μεταβλητής κατά ένα. Δείτε το ακόλουθο παράδειγμα:

```
#include<stdio.h>
int main(void)
{
     int a = 4, b;

     b = a--;
     printf("a=%d \t b=%d\n", a, b);

     b = --a;
     printf("a=%d \t b=%d\n", a, b);

     return 0;
}
```

Οι τελεστές σύγκρισης >, >=, <, <=, !=, == χρησιμοποιούνται για τη σύγκριση των τιμών που έχουν δύο εκφράσεις. Πιο αναλυτικά:

- a>10: χρησιμοποιείται για να ελέγξουμε αν η τιμή της μεταβλητής a είναι μεγαλύτερη από το 10
- a>=10: χρησιμοποιείται για να ελέγξουμε αν η τιμή της μεταβλητής a είναι μεγαλύτερη ή ίση από το 10
- a<10: χρησιμοποιείται για να ελέγξουμε αν η τιμή της μεταβλητής a είναι μικρότερη από το 10
- a<=10: χρησιμοποιείται για να ελέγξουμε αν η τιμή της μεταβλητής a είναι μικρότερη ή ίση από το 10

- `a!=10`: χρησιμοποιείται για να ελέγξουμε αν η τιμή της μεταβλητής `a` είναι διαφορετική από το `10`
- `a==10`: χρησιμοποιείται για να ελέγξουμε αν η τιμή της μεταβλητής `a` είναι ίση με το `10`

μια τέτοια έκφραση χαρακτηρίζεται αληθής (true) όταν η τιμή της είναι διαφορετική από το μηδέν. Αν είναι μηδέν χαρακτηρίζεται ψευδής (false). Όταν οι προαναφερθέντες τελεστές σύγκρισης χρησιμοποιούνται σε μια αληθή έκφραση, παράγουν `1`. Όταν η έκφραση είναι ψευδής, το αποτέλεσμα είναι `0`. Δείτε και προσπαθήστε να κατανοήσετε την έξοδο του παρακάτω προγράμματος:

```
#include<stdio.h>
int main(void)
{
     int a = 3, b = 5, c;

     a = (a > 3) + (b <= 5);
     b = (a == 3) + ((b-2) >= 3);
     c = (b != 1);

     printf("%d %d %d \n", a, b, c);
     return 0;
}
```

Οι συνδυαστικοί τελεστές χρησιμοποιούνται για να γράφουν εκφράσεις με πιο σύντομο τρόπο. Οι δύο ακόλουθες εντολές είναι ισοδύναμες:

```
a += 6;
a = a +6;
```

Ομοίως και οι ακόλουθες (προσοχή στην παρένθεση):

```
a *= b+3;
a = a * (b+3);
```

Οι πιο συνηθισμένοι συνδυαστικοί τελεστές είναι οι `+=`, `*=`, `-=`, `/=`, `%=`.

Για την εκτέλεση λογικών πράξεων, χρησιμοποιούνται οι λογικοί τελεστές. Ο τελεστής `&&` χρησιμοποιείται για την εκτέλεση της λογικής πράξης ΚΑΙ (AND). Η τιμή μιας λογικής έκφρασης που περιέχει τον τελεστή `&&` είναι ίση με `1` μόνο εάν όλοι οι όροι της έκφρασης είναι αληθείς. Διαφορετικά η τιμή της είναι ίση με `0`. Προσοχή, αν ο όρος που ελέγχεται σε μια έκφραση με τον τελεστή `&&` έχει ψευδή τιμή, ο μεταγλωττιστής δεν ελέγχει τους υπόλοιπους όρους και θέτει κατευθείαν την τιμή της έκφρασης ίση με `0`. Π.χ., δείτε τι θα τυπώσει το παρακάτω παράδειγμα:

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

```
int main(void)
{
     int a = 10, b = 20, c;

     c = (a > 15) && (++b > 15) ;
     printf("%d %d\n", c, b);

     return 0;
}
```

Αντίστοιχα, ο τελεστής `||` χρησιμοποιείται για την εκτέλεση της λογικής πράξης Ή (OR). Συγκεκριμένα, η τιμή μιας έκφρασης που περιέχει τον τελεστή `||` είναι ίση με `1` εάν έστω και ένας όρος της έκφρασης είναι αληθής, διαφορετικά είναι ίση με `0`. Προσοχή, παρόμοια με τον `&&`, αν ο όρος που ελέγχεται σε μια έκφραση με τον τελεστή `||` είναι αληθής, τότε ο μεταγλωττιστής δεν ελέγχει τους υπόλοιπους όρους και θέτει κατευθείαν την τιμή της συνολικής έκφρασης ίση με `1`. Π.χ., δείτε τι θα τυπώσει το παρακάτω παράδειγμα:

```
#include<stdio.h>
int main(void)
{
     int a = 10, b = 20, c;

     c = (a > 5) && (++b > 15);
     printf("%d %d\n", c, b);

     return 0;
}
```

Θα δούμε περισσότερες ασκήσεις στην επόμενη ενότητα, σε συνδυασμό με έλεγχο προγράμματος. Για την ώρα ας προσπαθήσουμε να υλοποιήσουμε τις παρακάτω ασκήσεις, χωρίς τη χρήση της εντολής `if`. Δείξτε προσοχή στην προτεραιότητα των τελεστών. Π.χ., ο τελεστής `*` έχει μεγαλύτερη προτεραιότητα από τους +, −. Βρείτε και μελετήστε έναν σχετικό πίνακα.

Άσκηση 1.15
Να γραφεί ένα πρόγραμμα το οποίο να διαβάζει δύο ακεραίους και να εμφανίζει το μικρότερο.

Άσκηση 1.16
Να γραφεί ένα πρόγραμμα το οποίο να διαβάζει τον ελάχιστο βαθμό που απαιτείται να πάρει ένας φοιτητής για να επιτύχει στις εξετάσεις, τους βαθμούς τριών φοιτητών και να εμφανίζει πόσοι από αυτούς πέτυχαν στις εξετάσεις.

12. Έλεγχος προγράμματος με χρήση `if`, `if-else`

Εώς τώρα, οι εντολές των προγραμμάτων που έχουμε συναντήσει εκτελούνται από πάνω, προς τα κάτω, με τη σειρά που εμφανίζονται στο πρόγραμμα. Ωστόσο, σε πολλές περιπτώσεις σε ένα

πρόγραμμα, κάποιες εντολές θα πρέπει να εκτελεστούν μόνο αν ισχύουν κάποιες συνθήκες. Η εντολή if επιτρέπει την επιλεκτική εκτέλεση ενός τμήματος κώδικα, ανάλογα με την τιμή μιας συνθήκης. Στην πιο απλή της μορφή συντάσσεται ως εξής:

```
if (συνθήκη)
{
      ... // ομάδα εντολών
}
```

Αν η συνθήκη ειναι αληθής, εκτελούνται οι εντολές ανάμεσα στα άγκιστρα. Π.χ.:

```
int x = 3;
if (x != 0)
{
     printf("Yes\n");
}
```

Αφού το `x` δεν είναι ίσο με `0`, η `if` συνθήκη είναι αληθής και το πρόγραμμα εμφανίζει `Yes`. Αν η συνθήκη δεν είναι αληθής, η ομάδα των εντολών δεν εκτελείται. Έτσι, ο ακόλουθος κώδικας δεν εμφανίζει τίποτα:

```
int  x = -3;
if (x == 0)
{
     printf("Yes\n");
}
```

Υπενθυμίζουμε ότι εάν η ομάδα των εντολών αποτελείται από μια μόνο εντολή, τα άγκιστρα μπορούν να παραλειφθούν. Συνίσταται, ωστόσο να τα χρησιμοποιούμε σε κάθε περίπτωση, για να αποφεύγουμε πιθανά λάθη.

Η εντολή if μπορεί να συμπληρώνεται προαιρετικά με την εντολή else, όπως φαίνεται παρακάτω:

```
if (συνθήκη)
{
      ... // ομάδα εντολών Α
}
else
{
      ... // ομάδα εντολών Β
}
```

Αν η συνθήκη είναι αληθής, εκτελείται η ομάδα εντολών Α, διαφορετικά, εκτελείται η ομάδα εντολών Β. Δείτε π.χ., τι εμφανίζει ο παρακάτω κώδικας:

```
int  x = -3;
```

```
if (x > 0)
{
     printf(“One\n”);
}
else
{
     printf(“Two\n”);
}
```

Στη γενική περίπτωση, μια `if` εντολή μπορεί να περιέχει ένθετες `if` και `else` εντολές. Και αυτές μπορούν με τη σειρά τους να περιέχουν και άλλες κ.ο.κ. Δείτε το παρακάτω πρόγραμμα:

```
#include<stdio.h>
int main(void)
{
     int a = 10, b = 20, c = 30;
     if (a > 5)
     {
          if (b == 20)
               printf("1 ");
          if (c == 40)
               printf("2 ");
          else
               printf("3 ");
     }
     else
          printf("4\n");

     return 0;
}
```

Για να ξεχωρίζετε ποια `if` συνδέεται με ποια `else`, συνίστατι η χρήση αγκίστρων ακόμη και όταν δεν χρειάζεται και φυσικά καλή στοίχιση του κώδικα!

Όταν θέλουμε να ελέγξουμε μια σειρά από συνθήκες, συνήθως χρησιμοποιούμε την ακόλουθη σύνταξη:

```
if (συνθήκη Α)
{
     ... // ομάδα εντολών Α
}
else if(συνθήκη Β)
{
     ... // ομάδα εντολών Β
```

```
}
else if(συνθήκη C)
{
      ... // ομάδα εντολών C
}
...
else
{
      ... // ομάδα εντολών N
}
```

Αν βρεθεί μια συνθήκη που να είναι αληθής, εκτελείται η αντίστοιχη ομάδα εντολών και οι υπόλοιπες `else if` δεν ελέγχονται. Η εκτέλεση του κώδικα στην περίπτωση αυτή συνεχίζει με την εντολή που υπάρχει μετά την τελευταία `else`. Δείτε το παρακάτω πρόγραμμα:

```
#include<stdio.h>
int main(void)
{
     int a;
     printf("Enter number: ");
     scanf("%d", &a);

     if (a == 1)
          printf("One\n");
     else if (a == 2)
          printf("Two\n");
     else
          printf("Other\n");

     printf("End\n");
     return 0;
}
```

Η τελευταία `else` δεν είναι υποχρεωτικό να υπάρχει. Αν δεν υπάρχει και καμία συνθήκη δεν είναι αληθής, τότε πολύ απλά το πρόγραμμα συνεχίζει με την εκτέλεση της επόμενης εντολής μετά την τελευταία `else if`.

Άσκηση 1.17
Να γραφεί ένα πρόγραμμα το οποίο να διαβάζει έναν ακέραιο και να εμφανίζει την απόλυτη τιμή του.

Άσκηση 1.18
Να γραφεί ένα πρόγραμμα το οποίο να διαβάζει τους βαθμούς δύο φοιτητών και να τους εμφανίζει με αύξουσα σειρά.

Άσκηση 1.19
Να γραφεί ένα πρόγραμμα το οποίο να διαβάζει δύο ακεραίους και να εμφανίζει το αποτέλεσμα της σύγκρισης τους, χωρίς τη χρήση της εντολής `else`.

Άσκηση 1.20
Να γραφεί ένα πρόγραμμα το οποίο να διαβάζει δύο πραγματικούς αριθμούς (π.χ. `a`, `b`) και να εμφανίζει τη λύση της εξίσωσης `a * x + b = 0`.

Άσκηση 1.21
Να γραφεί ένα πρόγραμμα το οποίο να διαβάζει τους βαθμούς 3 φοιτητών και να τους εμφανίζει με αύξουσα σειρά.

13. Έλεγχος προγράμματος με χρήση `switch`

Η εντολή ελέγχου switch χρησιμοποιείται εναλλακτικά έναντι της if-else-if δομής, όταν επιθυμούμε να ελέγξουμε μια έκφραση για όλες τις δυνατές τιμές που μπορεί να πάρει και να χειριστούμε την κάθε περίπτωση ξεχωριστά. Στην πιο συνηθισμενη της μορφή, η εντολή switch συντάσσεται ως εξής:

```
switch(έκφραση)
{
      case σταθερά_1:
      //ομάδα εντολών που θα εκτελεστεί αν η τιμή της έκφρασης είναι ίση με τη σταθερά_1
      break;
      case σταθερά_2:
      //ομάδα εντολών που θα εκτελεστεί αν η τιμή της έκφρασης είναι ίση με τη σταθερά_2
      break;
      ...
      case σταθερά_N:
      //ομάδα εντολών που θα εκτελεστεί αν η τιμή της έκφρασης είναι ίση με τη σταθερά_N
      break;

      default:
      //ομάδα εντολών που θα εκτελεστεί αν η τιμή της έκφρασης δεν είναι ίση με καμία από τις
      //προηγούμενες σταθερές
      break;
}
```

Η έκφραση που ελέγχεται πρέπει να είναι ακέραια μεταβλητή ή έκφραση. Οι σταθερές πρέπει να είναι ακέραιες με διαφορετικές τιμές μεταξύ τους. Κατά την εκτέλεση της εντολής `switch`, η τιμή της έκφρασης συγκρίνεται με κάθε σταθερά `case`. Αν βρεθεί μια ίδια τιμή, εκτελούνται οι εντολές του αντίστοιχου `case` και οι υπόλοιπες `case` περιπτώσεις δεν ελέγχονται. Αν δεν βρεθεί, ακολουθούνται οι εντολές που ακολουθούν την `default` περίπτωση. Και στις δύο περιπτώσεις, η

αντίστοιχη εντολή `break` τερματίζει τη `switch` και το πρόγραμμα συνεχίζει με την επόμενη εντολή. Δείτε το παρακάτω πρόγραμμα:

```
#include<stdio.h>
int main(void)
{
     int a;

     printf("Enter number: ");
     scanf("%d", &a);

     switch(a)
     {
          case 1:
               printf("One\n");
          break;

          case 2:
               printf("Two\n");
          break;

          default:
               printf("Other\n");
          break;
     }

     printf("End\n");
     return 0;
}
```

Εκτός από την εντολή break, η εντολή return μπορεί να χρησιμοποιηθεί για την έξοδο από μια switch εντολή. Για παράδειγμα:

```
#include<stdio.h>
int main(void)
{
     int a = 1;

     switch(a)
     {
          case 1:
               printf("One\n");
          return 0;
```

```
          case 2:
               printf("Two\n");
          break;
     }

     printf("End\n");
     return 0;
}
```

Η εντολή `return` τερματίζει όχι μόνο την εκτέλεση της `switch` εντολής, αλλά και την εκτέλεση της συνάρτησης στην οποία ανήκει. Η ύπαρξη `break` σε κάθε `case` δεν είναι υποχρεωτική. Σε περίπτωση που λείπει από την ομάδα εντολών της `case` περίπτωσης που εκτελείται, ο κώδικας συνεχίζει με την εκτέλεση των εντολών που υπάρχουν στις επόμενες `case` συνθήκες, μέχρι να βρεθεί μια εντολή `break`. Δείτε π.χ. το επόμενο πρόγραμμα και προσπαθήστε να κατανοήσετε την έξοδό του.

```
#include<stdio.h>
int main(void)
{
     int a = 1;

     switch(a)
     {
          case 1:
               printf("One\n");

          case 2:
               printf("Two\n");

          case 3:
               printf("Three\n");
          break;

          default:
               printf("Other\n");
          break;
     }

     printf("End\n");
     return 0;
}
```

Σε κάποιες περιπτώσεις, ενδέχεται να επιθυμούμε να εκτελεστεί η ίδια ομάδα εντολών σε δύο ή περισσότερες περιπτώσεις, οπότε και μπορούμε να τις ενώσουμε ως εξής:

```
case σταθερά_1:
case σταθερά_1:
case σταθερά_1:
//ομάδα εντολών που θα εκτελεστεί αν η τιμή της έκφρασης είναι ίση με σταθερά_1 ή  σταθερά_2
//ή  σταθερά_3
break;
```

Άσκηση 1.22
Να γραφεί ένα πρόγραμμα το οποίο να διαβάζει έναν ακέραιο που να αντιστοιχεί σε κάποιον μήνα (από `1` εώς `12`) και να εμφανίζει την εποχή στην οποία ανήκει, με χρήση `switch`.

Το κύριο μειονέκτημα της switch έναντι της if είναι ότι η switch κάνει έλεγχο μόνο για ισότητα, δηλαδή για τιμές της έκφρασης που να είναι ίσες με case σταθερές. Αντίθετα, με την if μπορούμε να κάνουμε οποιαδήποτε σύγκριση. Επιπλέον, στην εντολή switch, η έκφραση πρέπει να είναι ακέραια μεταβλητή ή έκφραση, καθώς και οι case σταθερές πρέπει να είναι ακέραιες. Οι χαρακτήρες θεωρούνται ακέραιοι, συνεπώς μπορούν να χρησιμοποιηθούν. Δεν επιτρέπεται όμως να χρησιμοποιηθούν πραγματικοί αριθμοί ή αλφαριθμητικά. Ωστόσο, όταν ένα τμήμα κώδικα μπορεί να χρησιμοποιηθεί είτε με switch, είτε με if-else-if, ίσως η switch κάνει τον κώδικα πιο ευανάγνωστο.

Άσκηση 1.23
Να γραφεί ένα πρόγραμμα το οποίο να προσομοιώνει τη λειτουργία της αριθμομηχανής. Το πρόγραμμα να διαβάζει το σύμβολο της αριθμητικής πράξης και δύο ακεραίους και να εμφανίζει το αποτέλεσμα της πράξης.

Άσκηση 1.24
Να γραφεί ένα πρόγραμμα το οποίο να εμφανίζει το εμβαδό ενός τετραγώνου ή ενός κύκλου, ανάλογα με την επιλογή του χρήστη. Με την επιλογή `0` να διαβάζει την πλευρά του τετραγώνου και να εμφανίζει το εμβαδό του. Με την επιλογή `1` να διαβάζει την ακτίνα του κύκλου και να εμφανίζει το εμβαδό του.

Άσκηση 1.25
Να γραφεί ένα πρόγραμμα το οποίο να διαβάζει τους βαθμούς τριών φοιτητών στο θεωρητικό και εργαστηριακό μέρος ενός μαθήματος. Ο τελικός βαθμός προκύπτει ως εξής: `lab_grd*0.3 + theory_grd*0.7`. Το πρόγραμμα να εμφανίζει πόσοι φοιτητές πήραν βαθμό μεταξύ `8` και `10`. Να χρησιμοποιήσετε μέχρι τρεις μεταβλητές.

Άσκηση 1.26
Να γραφεί ένα πρόγραμμα το οποίο να διαβάζει τρεις ακεραίους και να εμφανίζει το ζευγάρι με το μεγαλύτερο άθροισμα. Π.χ., αν ο χρήστης εισάγει `10, -8, 17, 5` το πρόγραμμα να εμφανίζει `10+17=27`.

Άσκηση 1.27

Να γραφεί ένα πρόγραμμα το οποίο να διαβάζει τους αριθμητές και παρονομαστές δύο κλασμάτων και έναν ακέραιο που να αντιστοιχεί σε μια μαθηματική πράξη (`1`: πρόσθεση, `2`: αφαίρεση, `3`: πολλαπλασιασμός, `4`: διαίρεση) και να χρησιμοποιεί την εντολή `switch` για να εμφανίσει το αποτέλεσμα της μαθηματικής πράξης.