ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

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

Επανάληψη Β' μέρος

1. Στόχος του εργαστηρίου

Στόχος του δεύτερου μέρους της επανάληψης είναι η υπενθύμιση 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. Ο βρόχος επανάληψης `for`

Η εντολή for είναι μία από τις τρεις εντολές της C που χρησιμοποιούνται για τη δημιουργία επαναληπτικών βρόχων. Η εκτέλεση του κώδικα που περιέχεται σε έναν for βρόχο επαναλαμβάνεται, όσο μια συνθήκη παραμένει αληθής. Η σύνταξή του βρόχου for είναι η εξής:

```
for (αρχική έκφραση; συνθήκη; τελική έκφραση)
{
      //ομάδα εντολών (σώμα του βρόχου), εκτελείται όσο η συνθήκη παραμένει αληθής
}
```

Οι τρεις εκφράσεις μπορούν να είναι οποιαδήποτε έγκυρη έκφραση της C. Η εντολή `for` εκτελείται ως εξής:

1. Η αρχική έκφραση εκτελείται μόνο μια φορά, όταν αρχίζει η εκτέλεση της εντολής for. Συνήθως είναι μια εντολή εκχώρησης, η οποία αρχικοποιεί κάποια μεταβλητή που θα χρησιμοποιηθεί από τις άλλες δύο εκφράσεις.
2. Ελέγχεται η τιμή της συνθήκης. Η συνθήκη είναι συνήθως μια σχεσιακή έκφραση. Αν είναι ψευδής, ο βρόχος τερματίζεται και η εκτέλεση του προγράμματος συνεχίζει με την εντολή που υπάρχει μετά το άγκιστρο κλεισίματος. Αν είναι αληθής, εκτελείται η ομάδα εντολών, η οποία αποκαλείται και "σώμα" του βρόχου.
3. Εκτελείται η τελική έκφραση. Συνήθως, η τελική έκφραση αλλάζει την τιμή κάποιας μεταβλητής που χρησιμοποιείται στη συνθήκη.
4. Τα βήματα 2, 3 επαναλαμβάνονται συνεχώς μέχρι η τιμή ης συνθήκης να γίνει ψευδής.

Π.χ., για να εμφανίσουμε τους αριθμούς από το 0 μέχρι το 4, θα μπορούσαμε να γράψουμε το παρακάτω πρόγραμμα:

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

```
     for (a = 0; a < 5; a++)
     {
          printf("%d", a);
     }
     return 0;
}
```

Προσπαθήστε με βάση τα προηγούμενα να κατανοήσετε τι συμβαίνει όταν εκτελείται ο βρόχος `for`.

Προσοχή, ένα από τα συχνά λάθη που γίνονται είναι η πρόσθεση ενός ερωτηματικού στο τέλος της εντολής `for`. Το ίδιο συμβαίνει συχνά και στην περίπτωση της εντολής `if`. Δοκιμάστε το στο προηγούμενο παράδειγμα και δείτε τι συμβαίνει.

Άσκηση 2.1
Ποια είναι η έξοδος του παρακάτω προγράμματος όταν τιμή του `j` είναι ίση με α) `10` και β) `3`; Επαληθεύστε την απάντησή σας, τρέχοντας το πρόγραμμα.

```
#include<stdio.h>
int main(void)
{
     int i, j;
     for (i = j; i <10; j++)
          printf("One\n");
     return 0;
}
```

Άσκηση 2.2
Ποια είναι η έξοδος του παρακάτω προγράμματος; Επαληθεύστε την απάντησή σας, τρέχοντας το πρόγραμμα.

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

     for(i = 12; i > 2; i-=5)
          printf("%d ", i);
          printf("End = %d\n", i);

     for(j = 5; j >= 0; j--)
          printf("Test\n");
     return 0;
```

```
}
```

Άσκηση 2.3

Να γραφεί ένα πρόγραμμα το οποίο να διαβάζει έναν ακέραιο αριθμό και αν αυτός ανήκει στο [10, 20] να εμφανίζει τόσες φορές όσες και η τιμή του αριθμού τη λέξη `One`, αλλιώς να διαβάζει 10 ακεραίους και να εμφανίζει μόνο τους αρνητικούς από αυτούς.

Η εντολή `break` που είδαμε στη `switch`, μπορεί να χρησιμοποιηθεί για τον άμεσο τερματισμό ενός `for`, `while` ή `do-while` βρόχου. Π.χ., δείτε το παρακάτω πρόγραμμα:

```
#include<stdio.h>
int main(void)
{
     int i;
     for(i = 1; i < 10; i++)
     {
          if(i == 5)
               break;

          printf("%d ", i);
     }
     printf("End= %d\n", i);
     return 0;
}
```

Όσο η τιμή του `i` δεν είναι ίση με `5`, η συνθήκη `if` είναι ψευδής και το πρόγραμμα εμφανίζει τις τιμές του `i` από `1` εώς και `4`. Όταν το `i` γίνει `5`, η εντολή `break` τερματίζει το βρόχο `for` και η εκτέλεση του προγράμματος συνεχίζει με την επόμενη εντολή. Άρα το πρόγραμμα εμφανίζει `End = 5`.

Σε περίπτωση ένθετων εντολών break, κάθε break τερματίζει την εκτέλεση του βρόχου ή της switch στην οποία περιέχεται. Δείτε το παρακάτω παράδειγμα:

```
#include<stdio.h>
int main(void)
{
     int i;
     for(i = 1; i < 10; i++)
     {
          switch(i)
          {
               case 4:
               break;
```

```
                default:
                     if(i == 3)
                          break;
                     else
                          printf("%d", i);
                printf("* ");
                break;
           }

           if(i == 6)
                break;
     }

     return 0;
}
```

Η εντολή `continue` μπορεί να χρησιμοποιηθεί μέσα σε έναν `for`, `while` ή `do-while` βρόχο. Σε αντίθεση με την `break`, η οποία τερματίζει άμεσα την εκτέλεση του βρόχου, η `continue` τερματίζει την τρέχουσα επανάληψη του βρόχου που την περιέχει και προκαλεί την έναρξη της επόμενης επανάληψης. Οι εντολές που βρίσκονται ανάμεσα στην εντολή `continue` και στο τέλος του βρόχου δεν εκτελούνται για την τρέχουσα επανάληψη. Δείτε π.χ., το παρακάτω πρόγραμμα:

```
#include<stdio.h>
int main(void)
{
     int i;
     for(i = 1; i < 10; i++)
     {
          if (i < 5)
               continue;
          printf("%d ", i);
     }
     return 0;
}
```

Άσκηση 2.4
Ποια είναι η έξοδος του παρακάτω προγράμματος;

```
#include<stdio.h>
int main(void)
{
     int i = 0, j = 5;

     for(i > j; i+j == 5; j < 2)
```

```
    {
        printf("One\n");
        i = 4;
        j = 2;
    }
    printf("%d %d\n", i, j);
    return 0;
}
```

Άσκηση 2.5

Να γραφεί ένα πρόγραμμα το οποίο να διαβάζει έναν ακέραιο στο [1, 10] και να εμφανίζει την προπαίδεια του αριθμού. Για παράδειγμα, αν ο χρήστης εισάγει τον αριθμό `5`, το πρόγραμμα να εμφανίζει: `1*5 = 5, 2*5 = 10, … , 10*5 = 50`. Το πρόγραμμα να υποχρεώνει το χρήστη να εισάγει αριθμό που να ανήκει στο [1, 10].

Άσκηση 2.6

Να γραφεί ένα πρόγραμμα το οποίο να διαβάζει τον αριθμό φοιτητών μιας τάξης και τους βαθμούς τους σε μια εξέταση. Το πρόγραμμα να εμφανίζει το μέσο όρο των βαθμών, τον μικρότερο και τον μεγαλύτερο, καθώς και πόσοι φοιτητές έχουν τον ίδιο μεγαλύτερο βαθμό.

3. Ένθετοι βρόχοι

Αν κάποιος επαναληπτικός βρόχος βρίσκεται μέσα σε κάποιον άλλο, κάθε επανάληψη του “εξωτερικού” προκαλεί την ολοκληρωμένη εκτέλεση του “εσωτερικού” (ένθετου) βρόχου. Ακολούθως θα δούμε τους ένθετους `for` βρόχους, ωστόσο τα ίδια θα ισχύουν και στην περίπτωση των ένθετων `while` και `do-while`. Ας δούμε το παρακάτω παράδειγμα:

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

    for(i = 0; i < 2; i++)
    {
        printf("One ");
        for(j = 0; j < i; j++)
            printf("Two ");
    }
    return 0;
}
```

Ας δούμε πώς λειτουργεί ένας ένθετος βρόχος:

- 1η επανάληψη εσωτερικού βρόχου (`i = 0`): Αφού η συνθήκη (`i < 2`) είναι αληθής, το πρόγραμμα εμφανίζει `One`.
  - 1η επανάληψη εσωτερικού βρόχου (`j = 0`): Αφού (`j = 0`) και (`i = 0`), η συνθήκη (`j < i`) δεν είναι αληθής, άρα δεν εκτελείται ο εσωτερικός βρόχος
- 2η επανάληψη εσωτερικού βρόχου (`i = 1`): Η συνθήκη (`i < 2`) είναι αληθής, άρα το πρόγραμμα εμφανίζει `One`.

- 1η επανάληψη εσωτερικού βρόχου (`j = 0`): Αφού `j=0` και `i=1`, συνθήκη (`j < i`) είναι αληθής, άρα το πρόγραμμα εμφανίζει `Two`.
- 2η επανάληψη εσωτερικού βρόχου (`j = 1`): Αφού `j=1` και `i=1`, η συνθήκη (`j < i`) δεν είναι αληθής, άρα τερματίζεται η εκτέλεση του εσωτερικού βρόχου.

- 3η επανάληψη εξωτερικού βρόχου (`i = 2`): Αφού η συνθήκη (`i < 2`) δεν είναι αληθής, τερματίζεται η εκτέλεση του εξωτερικού βρόχου.

Άρα το πρόγραμμα εμφανίζει `One One Two`. Με την λογική, μελετήστε και το παρακάτω πρόγραμμα. Θυμίζουμε ότι η εκτέλεση της εντολής `break` σε έναν βρόχο προκαλεί τον τερματισμό του βρόχου στον οποίο περιέχεται.

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

     for(i = 0; i < 2; i++)
     {
          for(j = 0; j < 2; j++)
          {
               if(i+j == 1)
                    break;
               printf("Two ");
          }
     printf("One ");
     }
     printf("\nVal1= %d Val2= %d\n", i, j);
     return 0;
}
```

Άσκηση 2.7
Ποια είναι η έξοδος του παρακάτω προγράμματος;

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

     for(i = 0; i < 2; i++)
     {
          printf("One ");
          for(i = 0; i < 2; i++)
               printf("Two ");
     }
     printf("Val = %d\n", i);
     return 0;
}
```

Άσκηση 2.8
Να γραφεί ένα πρόγραμμα το οποίο να εμφανίζει τον πίνακα της προπαίδειας.

4. Η εντολη `while`

Η εντολή `while` αποτελεί τον απλούστερο τρόπο για τη δημιουργία επαναληπτικών βρόχων στη C. Η σύνταξή της είναι η εξής:

```
while(συνθήκη)
{
	//Ομάδα εντολών που εκτελείται όσο η συνθήκη παραμένει αληθής
}
```

Όπως και στις περιπτώσεις των `if` και `for`, αν η ομάδα εντολών περιέχει μόνο μια εντολή, τα άγκιστρα δεν είναι απαραίτητα. Όταν εκτελείται μια εντολή `while`, πρώτα ελέγχεται η τιμή της συνθήκης. Αν είναι ψευδής, τότε ο βρόχος δεν εκτελείται. Αν είναι αληθής, εκτελείται η ομάδα εντολών και η συνθήκη ελέγχεται ξανά. Αν είναι ψευδής, τότε ο βρόχος τερματίζεται. Αν όχι, η ομάδα εντολών εκτελείται πάλι. Αυτή η διαδικασία επαναλαμβάνεται εώς ότου η τιμή της συνθήκης γίνει ψευδής. Δείτε π.χ., το ακόλουθο πρόγραμμα, το οποίο εμφανίζει τους αριθμούς από το `10` εώς και το `1`:

```
#include<stdio.h>
int main(void)
{
	int i = 10;

	while(i != 0)
	{
		printf("%d\n", i);
		i--;
	}
	return 0;
}
```

Θα αναρωτηθεί κάποιος γιατί να υπάρχουν δύο τύποι βρόχων, καθώς και πότε χρησιμοποιείται η `while` και πότε η `for`, αφού παράγουν ισοδύναμο κώδικα. Θα λέγαμε καταρχήν ότι είναι θέμα προτίμησης. Επιπλέον, η `while` προτιμάται όταν ο αριθμός των επανάληψεων δεν είναι γνωστός, ή όταν ελέγχεται απευθείας μια συνθήκη, χωρίς αρχικοποίηση κάποιας μεταβλητής ή χρήση βήματος. Αντίθετα, όταν ο αριθμός των επαναλήψεων είναι γνωστός και χρησιμοποιείται κάποιο βήμα, είναι προτιμότερη η χρήση της εντολής `for`, που οδηγεί και σε πιο ευανάγνωστο κώδικα, γενικά. Φυσικά, υπάρχουν και περιπτώσεις που η `for` μπορεί να οδηγήσει σε "άκομψο" κώδικα, όπως π.χ. όταν αλλάζει η τιμή του βήματος μέσα στο σώμα της. Στην περίπτωση αυτή, η `while` θα οδηγούσε σε πιο ευανάγνωστο κώδικα.

Αν η συνθήκη είναι πάντοτε αληθής, ο βρόχος θα εκτελείται συνεχώς, εκτός αν περιέχει κάποια εντολή τερματισμού (π.χ. `break`). Ο παρακάτω βρόχος είναι ατέρμονος:

```
while(1)
	printf("One\n");
```

Άσκηση 2.9
Να γραφεί ένα πρόγραμμα το οποίο να διαβάζει συνεχώς έναν ακέραιο και να τον εμφανίζει στην οθόνη μέχρι ο χρήστης να πατήσει το 0 (το οποίο δεν πρέπει να εμφανίζεται).

Άσκηση 2.10
Ποια είναι η έξοδος του παρακάτω προγράμματος;

```
#include<stdio.h>
int main(void)
{
	int i = -2;

	while(i-6)
	{
		printf("One ");
		i++;
		while(!(i+1))
		{
			printf("Two ");
			i--;
		}
		i = i+2;
	}
	return 0;
}
```

Άσκηση 2.11
Να γραφεί ένα πρόγραμμα το οποίο να διαβάζει έναν ακέραιο και να εμφανίζει από πόσα ψηφία αποτελείται καθώς και το άθροισμα των ψηφίων του. Για παράδειγμα, αν ο χρήστης εισάγει το `1234`, το πρόγραμμα να εμφανίζει `4` και `10`.

Άσκηση 2.12
Να γραφεί ένα πρόγραμμα το οποίο να διαβάζει συνεχώς ακεραίους που ανήκουν στο [0, 255] και να τους εμφανίζει με δυαδική μορφή. Για παράδειγμα αν ο χρήστης εισάγει τον αριθμό `32`, το πρόγραμμα να εμφανίζει `0010000`. Αν ο χρήστης εισάγει ακέραιο εκτός του [0, 255], το πρόγραμμα να τερματίζει.

5. Η εντολη `do-while`
Σε αντίθεση με τους `for` και `while` βρόχους, στους οποίους ο έλεγχος της συνθήκης γίνεται στην αρχή του βρόχου, στον βρόχο `do-while` γίνεται στο τέλος του. Αυτό σημαίνει ότι ο `do-while` εκτελείται πάντα τουλάχιστον μία φορά. Η εντολή `do-while` συντάσσεται ως εξής:

```
do
{
	//ομάδα εντολών που εκτελείται την πρώτη φορά και επαναλαμβάνεται όσο η συνθήκη είναι	//αληθής
} while(συνθήκη);
```

Η εντολή `do-while` εκτελείται ως εξής:

1. Εκτελείται η ομάδα των εντολών
2. Γίνεται έλεγχος της τιμής της συνθήκης. Αν είναι ψευδής, ο βρόχος τερματίζεται. Αν είναι αληθής, επανεκτελείται η ομάδα των εντολών. Το βήμα αυτό επαναλαμβάνεται μέχρι η συνθήκη να γίνει ψευδής.

Προσοχή, η `do-while` πρέπει να τελειώνει με `;`
Το επόμενο πρόγραμμα χρησιμοποιεί την εντολή do-while για να εμφανίσει τους αριθμούς από το 1 εώς και το 10:

```
#include<stdio.h>
int main(void)
{
     int i = 1;
     do
     {
          printf("%d\n", i);
          i++;
     } while ( i <= 10);

     return 0;
}
```

Γενικά ο βρόχος `do-while` χρησιμοποιείται λιγότερο από τους `for` και `while`. Μπορεί πολύ εύκολα να αντικατασταθεί από αυτούς. Μια συνηθισμένη χρήση του είναι για έλεγχο εγκυρότητας των τιμών που εισάγει ο χρήστης.

Άσκηση 2.13
Να γραφεί ένα πρόγραμμα το οποίο να διαβάζει έναν ακέραιο και να εμφανίζει τη λέξη `This` τόσες φορές όσες και η τιμή του ακεραίου, με χρήση `do-while`.

Άσκηση 2.14
Να γραφεί ένα πρόγραμμα το οποίο να διαβάζει συνεχώς ακεραίους και να εμφανίζει τα τετράγωνα των άρτιων αριθμών, μέχρι ο χρήστης να εισάγει αριθμό που να ανήκει στο [10, 20]. Το πρόγραμμα πριν τερματίσει να εμφανίζει πόσους θετικούς, πόσους αρνητικούς και πόσους αριθμούς εισήγαγε ο χρήστης με τιμή στο [300, 500]. Το μηδέν δεν προσμετράται ούτε στους θετικούς, ούτε στους αρνητικούς. Να χρησιμοποιήσετε `do-while`.

Άσκηση 2.15
Να γραφεί ένα πρόγραμμα το οποίο να εμφανίζει όλους τους αριθμούς από το 111 εώς το 999, εκτός από αυτούς που αρχίζουν από 4 ή τελειώνουν σε 6.

Άσκηση 2.16
Να γραφεί ένα πρόγραμμα το οποίο να διαβάζει συνεχώς ακεραίους και να υπολογίζει το άθροισμά τους μέχρι αυτό να ξεπεράσει το 100. Στη συνέχεια το πρόγραμμα να εμφανίζει το άθροισμά τους, καθώς και πόσους αριθμούς εισήγαγε ο χρήστης.

Άσκηση 2.17
Να γραφεί ένα πρόγραμμα το οποίο να διαβάζει συνεχώς ακεραίους μέχρι ο χρήστης να εισάγει την τιμή 0. Τότε το πρόγραμμα να εμφανίζει τον μεγαλύτερο θετικό και το μικρότερο αρνητικό αριθμό που εισήχθηκε. Αν ο χρήστης εισάγει μόνο θετικούς ή μόνο αρνητικούς αριθμούς, το πρόγραμμα να εμφανίζει ανάλογο μήνυμα. Το μηδέν δεν προσμετράται ούτε στους θετικούς, ούτε στους αρνητικούς αριθμούς.

Άσκηση 2.18
Να γραφεί ένα πρόγραμμα το οποίο να διαβάζει 100 ακεραίους και να βρίσκει και να εμφανίζει τις δύο μεγαλύτερες διαφορετικές τιμές.

6. Πίνακες
Οι μεταβλητές που έχουμε χρησιμοποιήσει μέχρι τώρα μπορεί να αποθηκεύσουν μία μόνο τιμή. Αντίθετα, οι πίνακες αποτελούν έναν τύπο μεταβλητής στον οποίο μπορούμε να αποθηκεύσουμε πολλές τιμές του ίδιου τύπου. Ένας πίνακας μπορεί γενικά να είναι πολυδιάστατος. Συνήθως χρησιμοποιούμε μονοδιάστατους και διδιάστατους πίνακες. Θα μπορούσε κανείς να πει ότι ένας πίνακας αποτελεί μια απλή δομή δεδομένων που αποτελείται από στοιχεία του ίδιου τύπου (π.χ. πίνακας ακεραίων, πραγματικών, χαρακτήρων κλπ). Ένας μονοδιάστατος πίνακας δηλώνεται ως εξής:

```
τύπος_δεδομένων όνομα_πίνακα[πλήθος στοιχείων]
```

Το πλήθος στοιχείων του πίνακα (μήκος ή μέγεθος) δηλώνεται από μια σταθερή, ακέραια έκφραση. Π.χ., η ακόλουθη εντολή:

```
int arr[1000];
```

δηλώνει έναν πίνακα από ακεραίους με το όνομα `arr`, οποίος περιέχει `1000` στοιχεία. Προσοχή, το πλήθος των στοιχείων ενός πίνακα δεν επιτρέπεται να αλλάξει κατά την εκτέλεση ενός προγράμματος. Όταν δηλώνεται ένας πίνακας, ο μεταγλωττιστής δεσμεύει ένα τμήμα μνήμης για να αποθηκεύσει τα στοιχεία του, τα οποία και αποθηκεύονται σε διαδοχικές θέσεις μνήμης.

Για να αναφερθούμε σε κάποιο στοιχείο του πίνακα, γράφουμε το όνομα του πίνακα συνοδευόμενο από τον δείκτη θέσης του στοιχείου μέσα σε αγκύλες `[]`. Προσοχή, σε έναν πίνακα με n στοιχεία, το πρώτο στοιχείο αποθηκεύεται στη θέση `[0]`, το δεύτερο στη θέση `[1]` και το τελευταίο στη θέση `[n-1]`. Τα στοιχεία ενός πίνακα μπορούν να χρησιμοποιηθούν με τον ίδιο τρόπο, όπως και οι απλές μεταβλητές.

Το παρακάτω πρόγραμμα δηλώνει έναν πίνακα 5 ακεραίων, θέτει τις τιμές 10, 20, 30, 40, 50 στα στοιχεία του και εμφανίζει στην οθόνη αυτά που έχουν τιμή μεγαλύτερη από 20:

```
#include<stdio.h>
int main(void)
{
     int i, arr[5];
```

```
	arr[0] = 10;
	arr[1] = 20;
	arr[2] = 30;
	arr[3] = 40;
	arr[4] = 50;

	for(i = 0; i < 5; i++)
		if(arr[i] > 20)
			printf("%d\n", arr[i]);

	return 0;
}
```

Προσοχή, η C δεν ελέγχει αν ο δείκτης θέσης ξεπερνάει τα όρια του πίνακα. Είναι ευθύνη του προγραμματιστή να εξασφαλίσει ότι κάτι τέτοιο δε θα συμβεί. Αν συμβεί, η συμπεριφορά του προγράμματος είναι απρόβλεπτη. Δηλαδή, μπορούμε να γράψουμε arr[5] = 60; και ο μεταγλωττιστής δεν θα το εντοπίσει ως λάθος και θα επιτρέψει την εκτέλεση του προγράμματος. Δοκιμάστε το, για να δείτε τη συμπεριφορά της C στο συγκεκριμένο σφάλμα.

Όπως και με τις απλές μεταβλητές, όταν δηλώνεται ένας πίνακας μπορούμε να αρχικοποιήσουμε τα στοιχεία του. Αν δεν τα αρχικοποιήσουμε, αποκτούν τυχαίες τιμές. Αυτό μπορεί να γίνει ως εξής:

```
int arr[4] = {10, 20, 30, 40, 50};
```

Έτσι, οι τιμές των `arr[0]`, `arr[1]`, `arr[2]`, `arr[3]`, `arr[4]` γίνονται ίσες με `10`, `20`, `30`, `40`, `50`, αντίστοιχα.

Άσκηση 2.18
Να γραφεί ένα πρόγραμμα το οποίο να δηλώνει έναν πίνακα 5 πραγματικών αριθμών και με χρήση επαναληπτικού βρόχου να θέτει τις τιμές 1.1, 1.2, 1.3, 1.4, 1.5 στα στοιχεία του. Στη συνέχεια να εμφανίζει τα στοιχεία του πίνακα με αντίστροφη σειρά, δηλαδή από το τελευταίο προς το πρώτο.

```
#include<stdio.h>
int main(void)
{
	int i;
	double arr[5];

	for(i = 0; i < 5; i++)
		arr[i] = 1.1 + (i*0.1);
	for(i = 4; i >= 0; i--)
		printf("%f\n", arr[i]);
	return 0;
}
```

Άσκηση 2.19
Να γραφεί ένα πρόγραμμα το οποίο να διαβάζει 10 ακεραίους και να τους αποθηκεύει σε έναν πίνακα. Στη συνέχεια, να ελέγχει αν ο πίνακας είναι συμμετρικός, δηλαδή, αν το πρώτο στοιχείο του είναι ίσο με το τελευταίο, το δεύτερο με το προτελευταίο κτλ.

```
#include<stdio.h>
#define SIZE 10

int main(void)
{
     int i, a[SIZE];

     for(i = 0; i < SIZE; i++)
     {
          printf("Enter element a[%d]: ", i);
          scanf("%d", &a[i]);
     }

     for(i = 0; i < SIZE/2; i++)
          if(a[i] != a[SIZE-1-i])
          {
               printf("Non symmetric array\n");
               return 0;
          }
     printf("Symmetric array\n");
     return 0;
}
```

Αν το μέγεθος του πίνακα ήταν περιττός αριθμός (π.χ. ίσο με `11`), θα άλλαζε κάτι στην παραπάνω λύση;

Άσκηση 2.20

Να γραφεί ένα πρόγραμμα το οποίο να διαβάζει τους βαθμούς 100 φοιτητών και να τους αποθηκεύει σε έναν πίνακα. Στη συνέχεια, να εμφανίζεται ο μέσος όρος, ο μεγαλύτερος και ο μικρότερος βαθμός, καθώς και οι θέσεις της πρώτης εμφάνισής τους στον πίνακα. Το πρόγραμμα να υποχρεώνει το χρήστη να εισάγει έναν βαθμό στο διάστημα [0, 10].

Άσκηση 2.21

Να γραφεί ένα πρόγραμμα το οποίο να δηλώνει δύο πίνακες 10 ακεραίων και να διαβάζει τις τιμές των στοιχείων τους. Στη συνέχεια, να ελέγχει αν υπάρχουν κοινά στοιχεία στους δύο πίνακες. Αν ναι, να εμφανίζει την τιμή του κάθε κοινού στοιχείου και τη θέση του στους δύο πίνακες. Αλλιώς να εμφανίζει ένα μήνυμα ότι δεν υπάρχουν κοινά στοιχεία.

7. Διδιάστατοι Πίνακες

Οι διδιάστατοι πίνακες της C μοιάζουν με τους γνωστούς μαθηματικούς πίνακες δύο διαστάσεων της άλγεβρας και αποτελούνται και αυτοί από γραμμές και στήλες. Όπως και οι πίνακες μίας διάστασης, έτσι και οι διδιάστατοι αποτελούνται από στοιχεία του ίδιου τύπου. Για να δηλώσουμε έναν διδιάστατο πίνακα πρέπει να δηλώσουμε το όνομα του πίνακα, τον τύπο δεδομένων των στοιχείων του, τον αριθμό των γραμμών και στηλών του:

```
τύπος_δεδομένων όνομα_πίνακα[πλήθος_γραμμών][πλήθος_στηλών]
```

Το πλήθος των στοιχείων του πίνακα είναι προφανώς ίσο με το πλήθος των γραμμών του επί το πλήθος των στηλών του. Για να αναφερθούμε σε κάποιο στοιχείο του πίνακα, γράφουμε το όνομα του πίνακα, συνοδευόμενο από τον δείκτη γραμμής του και τον δείκτη στήλης του, μέσα σε διπλές αγκύλες. Και στην περίπτωση των διδιάστατων η αρίθμηση των δεικτών θέσης γραμμής και στήλης αρχίζουν από το μηδέν. Για παράδειγμα, για να δημιουργήσουμε έναν πίνακα ακεραίων με `3` γραμμές και `4` στήλες:

```
int a[3][4];
```

με στοιχεία τα `a[0][0]`, `a[0][1]`, ..., `a[2][3]`. Η αρχικοποίηση ενός διδιάστατου πίνακα γίνεται με παρόμοιο τρόπο όπως και ενός μονοδιάστατου:

```
int arr[3][3] = {{10, 20, 30}, {40, 50, 60}, {70, 80, 90}};
```

Τα εσωτερικά άγκιστρα μπορούν να παραλειφθούν.

Άσκηση 2.22
Να γραφεί ένα πρόγραμμα το οποίο να διαβάζει οκτώ ακέραιους, να τους αποθηκεύει σε έναν πίνακα `2x4` και να εμφανίζει τα στοιχεία του πίνακα αντίστροφα, δηλαδή ξεκινώντας από το "κάτω-δεξιά" και καταλήγοντας στο "πάνω-αριστερά".

```
#include<stdio.h>

#define ROWS 2
#define COLS 4

int main(void)
{
     int i, j, arr[ROWS][COLS];

     for(i = 0; i < ROWS; i++)
     {
          for(j = 0; j < COLS; j++)
          {
               printf("Enter arr[%d][%d]: ", i, j);
               scanf("%d", &arr[i][j]);
          }
     }
     printf("\nArray elements\n");
     printf("--------------\n");
     for(i = ROWS-1; i >= 0; i--)
          for(j = COLS-1; j >= 0; j--)
               printf("arr[%d][%d] = %d\n", i, j, arr[i][j]);

     return 0;

}
```

8. Χαρακτήρες

Εκτός από τους αριθμητικούς τύπους δεδομένων που έχουμε εώς τώρα χρησιμοποιήσει (`int`, `float`, `double`), θα ασχοληθούμε τώρα με τον αριθμητικό τύπο `char`. Για τη διαχείριση χαρακτήρων (αλλά και αλφαριθμητικών που θα δούμε στο επόμενο εργαστήριο) θα θεωρήσουμε ότι οι χαρακτήρες είναι κωδικοποιημένοι στο πρότυπο ASCII.

Ένας χαρακτήρας στον κώδικα ASCII κωδικοποιείται σαν ακέραιος με τιμή από `0` εώς `255`. Για την αποθήκευση της τιμής του μπορεί να χρησιμοποιηθεί ο τύπος `char`. Όταν αποθηκεύουμε έναν χαρακτήρα σε μια μεταβλητή τύπου `char`, στην πραγματικότητα αποθηκεύεται η ASCII τιμή του χαρακτήρα. Η αποθήκευση γίνεται π.χ. ως εξής:

```
char ch;
ch = 'c';
```

Έτσι η τιμή της μεταβλητής ch γίνεται ίση με την τιμή ASCII του χαρακτήρα `'c'` δηλαδή ίση με `99`. Άρα, οι εντολές `ch = 'c'` και `ch = 99` είναι ισοδύναμες. Συνίσταται η χρήση της πρώτης εντολής, αφενός γιατί οδηγεί σε πιο ευανάγνωστο κώδικα και αφετέρου γιατί δεν εξαρτάται από το σύνολο κωδικοποίησης, άρα οδηγεί σε πιο έυκολα μεταφέρσιμο κώδικα. Προσοχή: όταν χρησιμοποιείται ένας σταθερός χαρακτήρας, πρέπει να περικλείεται από μονές αποστρόφους! Αν στο προηγούμενο παράδειγμα παραλείπαμε τις αποστρόφους, τότε ο μεταγλωττιστής θα θεωρούσε ότι στην εντολή `ch = c;` το `c` θα ήταν μια μεταβλητή.

Για την εμφάνιση ενός χαρακτήρα χρησιμοποιείται όπως έχουμε δει το `%c` ενώ για την εμφάνιση της ASCII τιμής του το `%d`. Δείτε το παρακάτω πρόγραμμα και δοκιμάστε να το τρέξετε.

```
#include <stdio.h>
int main(void)
{
     char ch;
     ch = 'a';
     printf("Char= %c and its ASCII code is %d\n", ch, ch);

     return 0;
}
```

Δείτε και το παρακάτω πρόγραμμα. Τι θα εμφανίσει και γιατί;

```
#include <stdio.h>
int main(void)
{
     char a = 70, b = 40;
     if ('a' > 'b')
          printf("One\n");

     return 0;
}
```

Αφού η C χειρίζεται τους χαρακτήρες σαν ακεραίους, μπορούμε να τους χρησιμοποιήσουμε σε αριθμητικές εκφράσεις. Δείτε το παρακάτω παράδειγμα:

```
char ch = 'c';
int i;
ch++;
ch = 68;
i = ch - 3;
```

Συνηθισμένη αιτία παρεξηγήσεων είναι η διαχείριση χαρακτήρων που αντιστοιχούν σε ψηφία. Π.χ. για να εμφανίσουμε το `2` μπορούμε να γράψουμε `printf("%c", '2');` προσοχή, όχι `2`. Το `'2'` αντιστοιχεί στον ASCII χαρακτήρα με τιμή `50`. Έτσι, θα μπορούσαμε π.χ. να ελέγξουμε εάν ένας χαρακτήρας αντιστοιχεί σε ψηφίο `0-9` ως εξής:

```
if (ch >= '0' && ch <= '9')
```

Άσκηση 2.23
Να γραφεί ένα πρόγραμμα το οποίο να εμφανίζει τα κεφαλαία και τα πεζά γράμματα του λατινικού αλφαβήτου, καθώς και τις ASCII τιμές τους.

```
#include<stdio.h>
int main(void)
{
     int i;
     for(i = 0; i < 26; i++)
          printf("%c(%d) - %c(%d)\n", 'a'+i, 'a'+i, 'A'+i, 'A'+i);

     return 0;
}
```

Άσκηση 2.24
Να γραφεί ένα πρόγραμμα το οποίο να διαβάζει δύο χαρακτήρες και να εμφανίζει τους χαρακτήρες που βρίσκονται μεταξύ τους στο ASCII σύνολο. Για παράδειγμα, αν ο χρήστης εισάγει `af` ή `fa`, το πρόγραμμα να εμφανίζει bcde.

```
#include<stdio.h>
int main(void)
{
     char ch1, ch2;
     printf("Enter characters: ");
     scanf("%c%c", &ch1, &ch2);

     if (ch1 < ch2)
     {
          ch1++;
          while(ch1 != ch2)
          {
               printf("%c", ch1);
               ch1++;
          }
```

```
	}
	else
	{
		ch2++;
		while(ch2 != ch1)
		{
			printf("%c", ch2);
			ch2++;
		}

	}
	return 0;
}
```

Άσκηση 2.25
Να γραφεί ένα πρόγραμμα το οποίο να εμφανίζει όλα τα πεζά γράμματα του λατινικού αλφαβήτου σε μία γραμμή, τα κεφαλαία γράμματα σε μια δεύτερη γραμμή και τους χαρακτήρες που αντιστοιχούν στα ψηφία `0-9` σε μια τρίτη γραμμή. Να χρησιμοποιήσετε μόνο έναν βρόχο `for`.

9. Αλφαριθμητικά
Ένα αλφαριθμητικό (string) στη C είναι μια ακολουθία χαρακτήρων, η οποία πρέπει να τελειώνει με έναν ειδικό χαρακτήρα (τερματικό χαρακτήρα - null character) και προσδιορίζει το τέλος του. Συμβολίζεται με `'\0'`. Μια ακολουθία από κανέναν ή περισσότερους χαρακτήρες που περιέχεται μέσα σε διπλά εισαγωγικά ονομάζεται κυριολεκτικό αριθμητικό. Τα εισαγωγικά χρησιμοποιούνται για την οριοθέτηση του αλφαριθμητικού και δεν αποτελούν μέρος του. Τεχνικά, η C χειρίζεται τα αλφαριθμητικά σαν έναν ανώνυμο πίνακα από χαρακτήρες. Έτσι, ο μεταγλωττιστής, όταν συναντήσει ένα κυριολεκτικό αριθμητικό, δεσμεύει μνήμη για να αποθηκεύσει τους χαρακτήρες του καθώς και τον τερματικό χαρακτήρα. Για παράδειγμα, αν συναντήσει το αλφαριθμητικό `"message"`, θα δεσμεύσει 8 θέσεις μνήμης, 1 για κάθε χαρακτήρα του και μία για τον τερματικό χαρακτήρα.

Για να αποθηκεύσουμε ένα αλφαριθμητικό σε μια μεταβλητή χρησιμοποιούμε έναν πίνακα χαρακτήρων. Σύμφωνα με τα παραπάνω, για ένα αλφαριθμητικό N χαρακτήρων, θα πρέπει να δηλώσουμε έναν πίνακα με N+1 θέσεις. Π.χ., η παρακάτω εντολή δηλώνει τον πίνακα `str` στον οποίο μπορεί να αποθηκευτεί ένα αλφαριθμητικό με εώς 7 χαρακτήρες.

```
char str[8];
```

Όταν δηλώνεται ένας πίνακας, μπορεί να αρχικοποιηθεί με ένα αλφαριθμητικό. Για πάραδειγμα, με την ακόλουθη δήλωση, ο μεταγλωττιστής αντιγράφει τους χαρακτήρες του `"message"` στον πίνακα `str` και προσθέτει τον τερματικό χαρακτήρα. Δηλαδή η τιμή του `str[0]` γίνεται `'m'`, το `str[1]` γίνεται `'e'` κτλ. Η τιμή του τελευταίου στοιχείου γίνεται ίση με τον τον τερματικό χαρακτήρα, δηλαδή με `'\0'`:

```
char str[8] = "message";
```

Αυτή η δήλωση είναι ισοδύναμη με:

```
char str[8] = {'m', 'e', 's', 's', 'a', 'g', 'e', '\0'};
```
Ένας πιο ευέλικτος τρόπος αρχικοποίησης του πίνακα είναι να μη δηλωθεί η διάστασή του και να την υπολογίσει ο μεταγλωττιστής. Π.χ. ως εξής:

```
char str[] = "message";
```

οπότε δεσμεύονται 8 θέσεις μνήμης για τον `str`. Ο τρόπος αυτός είναι πιο γρήγορος και πιο ασφαλής.

Οι κύριες συναρτήσεις που χρησιμοποιούνται για την εμφάνιση αλφαριθμητικών είναι οι `printf()` και `puts()`. Η `printf()` χρησιμοποιεί το προσδιοριστικό `%s` και έναν δείκτη στο αλφαριθμητικό (θα δούμε στο επόμενο εργαστήριο ότι το όνομα του πίνακα χωρίς αγκύλες είναι ένας δείκτης στον πίνακα):

```
#include<stdio.h>
int main(void)
{
     char str[] = "This is text";
     printf("%s\n", str);
     return 0;
}
```

Η `printf()` εμφανίζει τους χαρακτήρες του αλφαριθμητικού από τον πρώτο χαρακτήρα στον οποίο δείχνει ο δείκτης μέχρι να συναντήσει τον τερματικό χαρακτήρα. Προσοχή: αν η `printf()` συναντήσει τον τερματικό χαρακτήρα σε κάποια ενδιάμεση θέση, δεν θα εμφανίσει τους χαρακτήρες μετά από αυτόν. Π.χ., τρέξτε το παρακάτω πρόγραμμα:

```
#include<stdio.h>
int main(void)
{
     char str[] = "sample_text";

     str[4] = '\0';
     printf("%s\n", str);
     return 0;
}
```

Άσκηση 2.26
Να γραφεί ένα πρόγραμμα το οποίο να δημιουργεί ένα αλφαριθμητικό το οποίο να περιέχει όλα τα πεζά και κεφαλαία γράμματα του λατινικού αλφαβήτου.

```
#include<stdio.h>
int main(void)
{
     char str[53];
     int i;

     for (i = 0; i < 26; i++)
     {
```

```
            str[i] = 'a'+i;
            str[26+i] = 'A'+i;
        }
        str[52] = '\0';
        printf("%s\n", str);
        return 0;
}
```

Άσκηση 2.27

Ποια είναι η έξοδος του παρακάτω προγράμματος;

```
#include <stdio.h>
int main(void)
{
        char str[] = "Right'0'Wrong";

        printf("%s\n", str);
        return 0;
}
```