Α' Εξάμηνο
ΕΙΣΑΓΩΓΗ ΣΤΟ ΔΟΜΗΜΕΝΟ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟ

Εργαστήριο – 4η σειρά ασκήσεων.
Κοζάνη, 1 Νοεμβρίου 2007.

Πρόγραμμα p4-1 (μεταγλώττιση και εκτέλεση)
Πολύ απλό το πρώτο, για προθέρμανση! Πριν πατήσετε enter σκεφτείτε τι τιμή θα έχουν οι μεταβλητές που πρόκειται να τυπωθούν.

```
#include <stdio.h>

int main()
{
    int target;
    int *i, *j, *k;

    target = 1;
    i = &target;
    getchar();
    printf("target = %d and *i = %d \n", target, *i);

    k = j = i;
    getchar();
    printf("*j = %d and *k = %d \n", *j, *k);

    while (target != 0)
    {
        printf("Now, give me an integer (zero to quit): ");
        scanf("%d", &target);
        printf("Now *i = %d, *j = %d and *k = %d \n", *i, *j, *k);
    }

    return 0;
}
```

Πρόγραμμα p4-2 (μεταγλώττιση και εκτέλεση)
Ένα πρόγραμμα που έχουμε ξαναδεί, αλλά αυτή τη φορά με χρήση πίνακα: εύρεση μέγιστου και ελάχιστου από 10 αριθμούς που εισάγουμε από το πληκτρολόγιο. Εισάγουμε και ένα νέο στοιχείο: υπολογισμό τυπικής απόκλισης σύμφωνα με τη σχέση $s=\sqrt{(\frac{\sum_{i=0}^{N_{\mathrm{DAT}}}(x_i-\bar{x})^2}{N_{\mathrm{DAT}}})}$ όπου $\bar{x}$ είναι η μέση τιμή.
Δείτε με προσοχή που έχει άγκιστρα { και }, που έχει αγκύλες [*και*] , *όπως και που έχει παρενθέσεις*

(*και*), *για να μην τα μπερδεύετε!*

```
#include <stdio.h>
#include <math.h>
#define NDAT 10

int main()
{
    int i, numbers[NDAT] = {0}, min, max, x;
    float ave = 0.0, stdev = 0.0;

    for (i = 0; i < NDAT; i++)
    {
        printf("Δώσε μου τον αριθμό %d \n", i+1);
        scanf("%d", &numbers[i]);
    }

    min = max = numbers[0];
    for (i = 0; i < NDAT; i++)
    {
        x = numbers[i];
        if(min > x) min = x;
        if(max < x) max = x;
        ave += x;
    }
    printf("Ελάχιστο = %d και μέγιστο = %d \n", min, max);

    ave /= NDAT;
    for (i = 0; i < NDAT; i++)
        stdev += pow(numbers[i] - ave,2);
    stdev = sqrt(stdev/NDAT);
    printf("Μέση τιμή +- τυπική απόκλιση: %f +- %f \n", ave,
stdev);

    return 0;
}
```

Πρόγραμμα p4-3 (τροποποίηση προγράμματος)
Να ξαναγράψετε το προηγούμενο πρόγραμμα αλλά αντί για συμβολισμό πινάκων χρησιμοποήστε συμβολισμό δεικτών (pointers), *παντού εκτός από εκεί όπου δηλώνεται ο πίνακας numbers*. Υπενθυμίζεται ότι το όνομα ενός πίνακα, π.χ. a[NDAT], είναι συγχρόνως και δείκτης στο πρώτο στοιχείο του, a[0]. Επίσης, η παράσταση a+i είναι δείκτης που δείχνει προς το στοιχείο a[i] και τέλος, για να πάρουμε την τιμή της μεταβλητής προς την οποία δείχνει ένας δείκτης προτάσσουμε

τον τελεστή * (βλ. και πρόγραμμα p4-1, πιο πάνω).

Πρόγραμμα p4-4 (μεταγλώττιση και εκτέλεση)
Πρόκειται για πρόγραμμα σχεδόν ίδιο με τα προηγούμενα δύο. Μόνο που δε χρειάζεται πια να περιοριστούμε σε ένα συγκεκριμένο μέγεθος πίνακα. Το ίδιο πρόγραμμα μπορεί να χρησιμοποιηθεί για διαφορετικό αριθμό δεδομένων κάθε φορά. Αυτό επιτυγχάνεται με τη λεγόμενη δυναμική κατανομή μνήμης (dynamic memory allocation).
Η δυναμική κατανομή μνήμης είναι μία τεχνική δέσμευσης ακριβώς όσης μνήμης χρειαζόμαστε κάθε φορά κατά τη διάρκεια της εκτέλεσης του προγράμματος και αποδέσμευσης αυτής της μνήμης όταν δε χρειάζεται άλλο πια. Για να επιτύχουμε αυτού του είδους τη διαχείριση μνήμης χρησιμοποιούμε τις παρακάτω συναρτήσεις που δηλώνονται στο αρχείο stdlib.h (το οποίο επομένως πρέπει να κάνουμ ε include):
`malloc` – δεσμεύει τόσες λέξεις μνήμης όσο το όρισμα που θα της δώσουμε και επιστρέφει ένα δείκτη στην αρχή της δεσμευμένης περιοχής. Αρχικό περιεχόμενο μνήμης: απροσδιόριστο.
`calloc` – παίρνει σαν πρώτο όρισμα αριθμό στοιχείων με τόσο μέγεθος σε λέξεις μνήμης όσο το δεύτερο όρισμα και επιστρέφει ένα δείκτη στην αρχή της δεσμευμένης περιοχής. Αρχικό περιεχόμενο αυτών, η τιμή 0.
`realloc` – παίρνει σαν όρισμα ένα δείκτη που δείχνει σε περιοχή μνήμης δεσμευμένη από εντολή malloc ή calloc και την αλλάζει κατά τόσο όσο ορίζει το δεύτερο όρισμά της, χωρίς να καταστρέψει τα περιεχόμενα στο ήδη δεσμευμένο τμήμα της μνήμης.
`free` – παίρνει σαν όρισμα το δείκτη μιας δεσμευμένης περιοχής της μνήμης και την αποδεσμεύει.
Το παρακάτω πρόγραμμα διαφέρει από το p4-2 στα εξής σημεία:

- νέα εντολή συμπερίληψης αρχείου stdlib.h
  #include <stdlib.h>
- Δεν υπάρχει η συμβολική σταθερά NDAT. Τώρα δηλώνεται μία ακέραια μεταβλητή n για τον αριθμό μεταβλητών που θέλουμε να αποθηκεύσουμε.
- Με το όνομα numbers δηλώνεται ένας δείκτης σε ακέραιο.
- Ζητείται και διαβάζεται ο αριθμός n των αριθμών που θα δοθούν και δεσμεύεται μνήμη που αντιστοιχεί σε n ακέραιους. Χρησιμοποιείται η συνάρτηση calloc, ενώ για το μέγεθος της μνήμης που αντιστοιχεί σε μία ακέραιη μεταβλητή χρησιμοποιείται η συνάρτηση sizeof. Αυτή σαν όρισμα μπορεί να πάρει όχι μόνο μια μεταβλητή αλλά και έναν τύπο π.χ. int, float, char και επιστρέφει πόση μνήμη χρειάζεται για την αποθήκευση αυτού.
- Στο τέλος αποδεσμεύεται η μνήμη με τη βοήθεια της συνάρτησης free, αν και στο συγκεκριμένο πρόγραμμα δεν είναι απαραίτητο.

Σημείωση: Στο παρακάτω πρόγραμμα χρησιμοποιείται συμβολισμός πινάκων αλλά μπορεί να χρησιμοποιηθεί εξίσου και ο ισοδύναμος συμβολισμός δεικτών όπως στο πρόγραμμα p4-3.

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

int main()
{
    int i, *numbers, n = 0, min, max, x;
    float ave = 0.0, stdev = 0.0;

    printf("Πόσους αριθμούς θα βάλεις;\n");
    scanf("%d", &n);
```

```
    numbers = (int *) calloc(n, sizeof(int));
    for (i = 0; i < n; i++)
    {
        printf("Δώσε μου τον αριθμό %d \n", i+1);
        scanf("%d", &numbers[i]);
    }

    min = max = numbers[0];
    for (i = 0; i < n; i++)
    {
        x = numbers[i];
        if(min > x) min = x;
        if(max < x) max = x;
        ave += x;
    }
    ave /= n;
    printf("Ελάχιστο = %d και μέγιστο = %d \n", min, max);

    for (i = 0; i < n; i++)
        stdev += pow(numbers[i] - ave,2);
    stdev = sqrt(stdev/n);
    printf("Μέση τιμή +- τυπική απόκλιση: %f +- %f \n", ave,
stdev);
    free(numbers);

    return 0;
}
```

Πρόγραμμα p4-5 (μεταγλώττιση και εκτέλεση)
Αυτό το μικρό πρόγραμμα δείχνει το συμβολισμό δεικτών όπως εφαρμόζεται σε πίνακες δύο διαστάσεων. Αν φανταστούμε έναν πίνακα a[m][n] σαν έναν πίνακα m γραμμών και κάθε γραμμή σαν έναν πίνακα n στοιχείων, τότε είναι σα να έχουμε αντί για κάθε γραμμή ένα δείκτη σε αυτή. Έτσι, χρειαζόμαστε δύο φορές τον τελεστή περιεχομένου * για να αποκτήσουμε πρόσβαση στις τιμές των στοιχείων του πίνακα: μία φορά για να εντοπίσουμε την αρχική γραμμή του πίνακα a και μια φορά για να βρούμε το αρχικό στοιχείο αυτής της γραμμής.

```
#include <stdio.h>

int main()
{
    int i, j, k, a[2][2] = {{1, 2}, {3, 4}};

    k = 0;
```

```
    for (i = 0; i < 2; i++)
        for (j = 0; j < 2; j++, k++)
            printf("Γραμμή %d στήλη %d: στοιχείο %d με τιμή %d και
ξανά: %d \n", i, j, k, a[i][j], **a+k);
}
```

Δοκιμάστε να το τροποποιήσετε για να δουλεύει με πίνακα τριών διαστάσεων (για μέγεθος 2 σε κάθε διάσταση θα έχει συνολικά 8 στοιχεία).