Υποχρεωτικό Μάθημα 3ου Εξαμήνου
Η/Υ 240: Δομές Δεδομένων
Χειμερινό Εξάμηνο Ακ. Έτους 2008-2009
Τμήμα Επιστήμης Υπολογιστών
Πανεπιστήμιο Κρήτης

1ο Φροντιστήριο

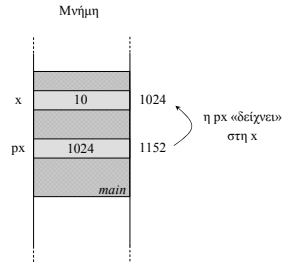
Διδάσκουσα:
Φατούρου Παναγιώτα
faturu [at] csd.uoi.gr

Βοηθός:
Κοσμάς Ελευθέριος
ekosmas [at] csd.uoi.gr

Δείκτες (Pointers)

Παράδειγμα:

```
void main (void)
{
    int x;
    int *px;

    x = 10;
    px = &x;

    printf ("x = %d", x);
    printf ("px = %u", px);
    printf ("*px = %d", *px);
}
```



Έξοδος παραδείγματος:

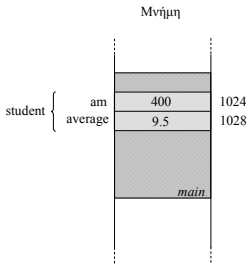
```
x = 10;
px = 1024;
*px = 10;
```

- Χρησιμοποιούμε:
  - **&x** : Για να μάθουμε την διεύθυνση μνήμης στην οποία διατηρείται η x
  - ***px** : Για να αποκτήσουμε την τιμή της μεταβλητής στην οποία δείχνει ο δείκτης px.

Δομές (Structs)

Παράδειγμα:

```
struct s {
    int am;
    float average;
};

void main (void)
{
    struct s student;

    student.am = 400;
    student.average = 9.5;
    printf ("A.M.:%d -- M.O.: %f",
          student.am, student.average);
}
```



Έξοδος παραδείγματος:

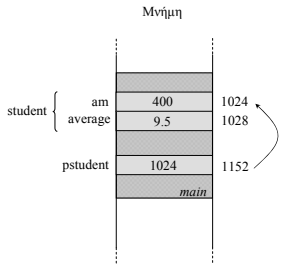
```
A.M.: 400 -- M.O.: 9.5
```

- Χρησιμοποιούμε **x**.<όνομα στοιχείου> : για να προσπελάσουμε τα στοιχεία μιας δομή που περιγράφεται από την **μεταβλητή x**

Δομές (Structs)

Παράδειγμα:

```
struct s {
    int am;
    float average;
};

void main (void)
{
    struct s student;
    struct s *pstudent;

    pstudent = &student;
    pstudent->am = 400;
    pstudent->average = 9.5;
    printf ("A.M.:%d -- M.O.: %f",
          pstudent->am, student->average);
}
```



Έξοδος παραδείγματος:

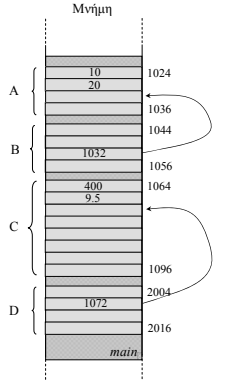
```
A.M.: 400 -- M.O.: 9.5
```

- Χρησιμοποιούμε **px** -> <όνομα στοιχείου> : για να προσπελάσουμε τα στοιχεία μιας δομή που περιγράφεται από τον **δείκτη px**

Πίνακες (Arrays)

Παράδειγμα:

```
struct s {
    int am;
    float average;
};

void main (void)
{
    int A[4];
    int *B[4];
    struct s C[4];
    struct s *D[4];

    A[0] = 10;
    A[1] = 20;

    B[2] = &A[2];

    C[0].am = 400;
    C[0].average = 9.5;

    D[1] = &C[1];
}
```

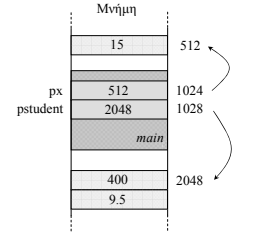


Malloc

- Επιτρέπει τη ***δυναμική*** (κατά την εκτέλεση του προγράμματος) δέσμευση μνήμης
  - παίρνει ως όρισμα το **μέγεθος της μνήμης** που θα δεσμευθεί
    - χρήση της συνάρτησης **sizeof**, π.χ. sizeof(int), sizeof(char), …
  - επιστρέφει έναν δείκτη στην αρχή της δεσμευμένης μνήμης (ή null σε περίπτωση αποτυχίας)
    - ο δείκτης αυτός είναι τύπου **void** και πρέπει να μετατραπεί στον κατάλληλο τύπο (casting)

Παράδειγμα:

```
void main (void)
{
    int *px;
    struct s *pstudent;

    px = (int *) malloc (sizeof(int));

    pstudent = (struct s *) malloc
              (sizeof(struct s));

    *px = 15;
    pstudent -> am = 400;
    pstudent -> average = 9.5;
}
```

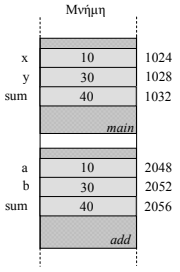


Συναρτήσεις

- Παίρνουν κάποια ορίσματα, εκτελούν κάποια λειτουργία και επιστρέφουν μία τιμή
- Είναι αναγκαίο να ορίζουμε τη συνάρτηση ή το *πρωτότυπό* της πριν τη συνάρτηση main του προγράμματός μας
- Κάθε συνάρτηση εκτελείται στο **δικό της χώρο μνήμης**

Παράδειγμα:

```
int add (int a, int b);

void main (void)
{
    int x = 10, y = 30;
    int sum = add (x+y);
    printf ("x+y = %d\n", sum);
}

int add (int a, int b)
{
    int sum = a+b;
    return (sum);
}
```



Έξοδος παραδείγματος:

```
x+y = 40
```


Εμβέλεια μεταβλητών

- Οι μεταβλητές ενός προγράμματος διακρίνονται σε δύο κατηγορίες:
  - καθολικές μεταβλητές: δηλώνονται στην αρχή κάθε προγράμματος πριν τις συναρτήσεις και μπορούν να προσπελάσονται από όλες τις συναρτήσεις.
  - τοπικές ή εσωτερικές μεταβλητές κάθε συνάρτησης

Παράδειγμα:

```
int a = 100;        // Μία καθολική μεταβλητή

void f1 (void);
void f2 (void);

void main (void)
{
    int b = 4;      // Μία τοπική μεταβλητή
    printf ("Η a στην main έχει τιμή: %d", a);
    f1();
    f2();
}

void f1 (void)
{
    printf ("Η a στην f1 έχει τιμή: %d", a);
}

void f2 (void)
{
    int a = 300;    // Μία τοπική μεταβλητή
    printf ("Η a στην f2 έχει τιμή: %d", a);
}
```

Έξοδος παραδείγματος:

```
Η a στην main έχει τιμή: 100
Η a στην f1 έχει τιμή: 100
Η a στην f2 έχει τιμή: 300
```


Παράδειγμα ανταλλαγής τιμών

- Έστω ότι θέλουμε να υλοποιήσουμε ένα πρόγραμμα που ανταλλάσσει τις τιμές δύο ακεραίων μεταβλητών

1ος Αλγόριθμος:

```
void main (void)
{
    int x = 5, y = 10;
    printf ("Πριν την ανταλλαγή x = %d, y = %d", x, y);
    x = y;
    y = x;
    printf ("Μετά την ανταλλαγή x = %d, y = %d", x, y);
}
```

Έξοδος 1ου Αλγορίθμου:

```
Πριν την ανταλλαγή: x = 5, y = 10
Μετά την ανταλλαγή: x = 10, y = 10
```

Ο 1ος αλγόριθμος είναι λανθασμένος

2ος Αλγόριθμος:

```
void main (void)
{
    int x = 5, y = 10;
    printf ("Πριν την ανταλλαγή x = %d, y = %d", x, y);
    temp = x;
    x = y;
    y = temp;
    printf ("Μετά την ανταλλαγή x = %d, y = %d", x, y);
}
```

Έξοδος 2ου Αλγορίθμου:

```
Πριν την ανταλλαγή: x = 5, y = 10
Μετά την ανταλλαγή: x = 10, y = 5
```

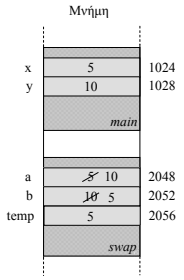
Ο 2ος αλγόριθμος είναι σωστός

Παράδειγμα ανταλλαγής τιμών – Με χρήση συνάρτησης

- **Πέρασμα με τιμή** των ορισμάτων της συνάρτησης

3ος Αλγόριθμος:

```
void swap (int a, int b);

void main (void)
{
    int x = 5, y = 10;
    printf ("Πριν την ανταλλαγή x = %d, y = %d", x, y);
    swap (x,y);
    printf ("Μετά την ανταλλαγή x = %d, y = %d", x, y);
}

void swap (int a, int b)
{
    int temp;
    temp = a;
    a = b;
    b = temp;
}
```



Έξοδος 3ου Αλγορίθμου:

```
Πριν την ανταλλαγή: x = 5, y = 10
Μετά την ανταλλαγή: x = 5, y = 10
```

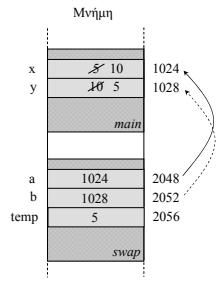
Ο 3ος αλγόριθμος είναι λανθασμένος

Παράδειγμα ανταλλαγής τιμών – Με χρήση συνάρτησης

- **Πέρασμα με αναφορά** των ορισμάτων της συνάρτησης

4ος Αλγόριθμος:

```
void swap (int *a, int *b);

void main (void)
{
    int x = 5, y = 10;
    printf ("Πριν την ανταλλαγή x = %d, y = %d", x, y);
    swap (&x,&y);
    printf ("Μετά την ανταλλαγή x = %d, y = %d", x, y);
}

void swap (int *a, int *b)
{
    int temp;
    temp = *a;
    *a = *b;
    *b = temp;
}
```



Έξοδος 4ου Αλγορίθμου:

```
Πριν την ανταλλαγή: x = 5, y = 10
Μετά την ανταλλαγή: x = 10, y = 5
```

Ο 4ος αλγόριθμος είναι σωστός

Παράδειγμα ανταλλαγής τιμών – Με χρήση συνάρτησης

- Χρήση καθολικών μεταβλητών

5ος Αλγόριθμος:

```
int x = 5, y = 10;
void swap (void);

void main (void)
{
    printf ("Πριν την ανταλλαγή x = %d, y = %d", x, y);
    swap ();
    printf ("Μετά την ανταλλαγή x = %d, y = %d", x, y);
}

void swap (void)
{
    int temp;
    temp = x;
    x = y;
    y = temp;
}
```

Έξοδος 5ου Αλγορίθμου:

```
Πριν την ανταλλαγή: x = 5, y = 10
Μετά την ανταλλαγή: x = 10, y = 5
```

Ο 5ος αλγόριθμος είναι σωστός

Συμπέρασμα

- Η ενημέρωση της τιμής μιας μεταβλητής x μέσω μιας συνάρτησης f μπορεί να πραγματοποιηθεί με έναν από τους παρακάτω τρόπους:
  - Ορισμός της x ως καθολικής μεταβλητής
    - **Κακή** προγραμματιστική επιλογή όταν έχουμε πολλές μεταβλητές που τροποποιούνται μέσω συναρτήσεων
  - Πέρασμα της x με αναφορά στη συνάρτηση f
  - Πέρασμα της x με τιμή στη συνάρτηση f, τροποποίησή της και επιστροφή της νέας τιμής
    - Παράδειγμα:

```
int f (int a)
{
        int b = a+100;
        return (b);
}

void main (void)
{
        int x = 100;
        printf ("Πριν την ενημέρωση x = %d", x);
        x = f(x);   <-- Αυτό δημιουργεί πρόβλημα;
        printf ("Μετά την ενημέρωση x = %d", x);
}
```

Έξοδος παραδείγματος:

```
Πριν την ενημέρωση x = 100
Μετά την ενημέρωση x = 200
```


Ερώτημα

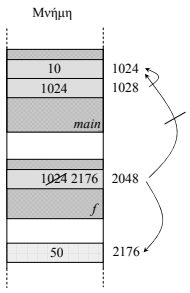
- Πώς μπορώ να ενημερώσω την τιμή μιας μεταβλητής x που είναι **δείκτης** σε κάποια άλλη μεταβλητή με τη χρήση μιας συνάρτησης f;
- Απάντηση: Με οποιονδήποτε από τους τρόπους που παρουσιάστηκαν στην προηγούμενη διαφάνεια.
- Χρειάζεται προσοχή εάν χρησιμοποιηθεί η μέθοδος του περάσματος με αναφορά της x στην f.
  - Θα πρέπει το όρισμα της f να οριστεί καταλλήλως και να είναι **δείκτης σε δείκτη**.
  - (δίνονται παραδείγματα στις επόμενες διαφάνειες ...)

Παράδειγμα 1

- Έστω ότι θέλουμε να ενημερώσουμε την τιμή μιας μεταβλητής px που είναι **δείκτης** σε κάποια άλλη μεταβλητή με τη χρήση μιας συνάρτησης f

1ος Αλγόριθμος: *Ο 1ος αλγόριθμος είναι λανθασμένος*

```
Void f (int *a)
{
    a = (int *) malloc (sizeof(int));
    *a = 50;
}

void main (void)
{
    int x = 10;
    int *px = &x;
    printf ("Πριν την ενημέρωση px=%u, *px=%d", px, *px);
    f(px);
    printf ("Μετά την ενημέρωση px=%u, *px=%d", px, *px);
}
```



Έξοδος 1ου Αλγορίθμου:

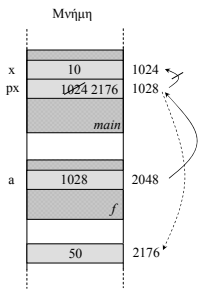
```
Πριν την ενημέρωση: px=1024, *px=10
Μετά την ενημέρωση: px=1024, *px=10
```

- **Ερώτηση: Με ποιο τρόπο περνάμε την μεταβλητή px στη συνάρτηση f; (με τιμή ή με αναφορά;)**

Παράδειγμα 2 – χρησιμότητα δείκτη σε δείκτη

2ος Αλγόριθμος:

```
Void f (int **a);

void main (void)
{
    int x = 10;
    int *px = &x;
    printf ("Πριν την ενημέρωση px=%u, *px = %d", px, *px);
    f(&px);
    printf ("Μετά την ενημέρωση px=%u, *px = %d", px, *px);
}

void f (int **a)
{
    *a = (int *) malloc (sizeof(int));
    **a = 50;
}
```



Έξοδος 2ου Αλγορίθμου:

```
Πριν την ενημέρωση: px=1024, *px=10
Μετά την ενημέρωση: px=2176, *px=50
```

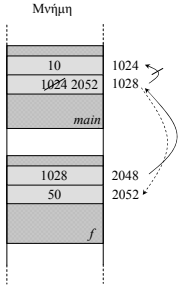
Ο 2ος αλγόριθμος είναι σωστός

- **Ερώτηση: Με ποιο τρόπο περνάμε την μεταβλητή px στη συνάρτηση f; (με τιμή ή με αναφορά;)**

Παράδειγμα 3 – χρησιμότητα της malloc

3ος Αλγόριθμος: *Ο 3ος αλγόριθμος είναι λανθασμένος*

```
Void f (int **a)
{
    int y = 50;
    *a = &y;
}

void main (void)
{
    int x = 10;
    int *px = &x;
    printf ("Πριν την ενημέρωση px=%u, *px=%d", px, *px);
    f(&px);
    printf ("Μετά την ενημέρωση px=%u, *px=%d", px, *px);
}
```



Έξοδος 3ου Αλγορίθμου:

```
Πριν την ενημέρωση: px=1024, *px=10
ΣΦΑΛΜΑ!
```

- Μετά το τέλος της f, ο χώρος μνήμης της y αποδίδεται στο σύστημα επειδή περιέχεται στο χώρο μνήμης της f
- Με τη χρήση της malloc αποφεύγετε το πρόβλημα αυτό, διότι δεσμεύεται χώρος μνήμης που **δεν ανήκει** σε κάποια συνάρτηση ή γενικότερα σε κάποιο πρόγραμμα.

Ολοκληρωμένο Παράδειγμα

- Έστω ότι η γραμματεία του Τμήματος Επιστήμης Υπολογιστών, επιθυμεί να διατηρήσει διάφορα στοιχεία για τους προπτυχιακούς φοιτητές.
  - Θεωρούμε πως επιθυμεί να διατηρήσει το Α.Μ. (am) και το μέσο όρο βαθμολογίας (average) κάθε φοιτητή
  - Για την αποθήκευση των στοιχείων κάθε φοιτητή χρησιμοποιείται η εξής δομή:

```
struct s {
      int am;
      float average;
};
```

  - Τα στοιχεία όλων των φοιτητών διατηρούνται στον πίνακα CSD.
- Ο πίνακας CSD θα μπορούσε να είναι ένας πίνακας από δομές struct s (δηλαδή `struct s CSD[MAX_STUDENTS]`).
  - Η λύση αυτή έχει το μειονέκτημα πως εάν η δομή struct s περιέχει πολλά στοιχεία, τότε σπαταλάτε άσκοπα μεγάλο μέρος της μνήμης είτε στον CSD αποθηκεύεται ένας φοιτητής είτε MAX_STUDENTS φοιτητές
- Μια καλύτερη λύση είναι ο CSD να περιέχει δείκτες προς δομές struct s s (δηλαδή `struct s *CSD[MAX_STUDENTS]`) και να δεσμεύεται χώρος για μία δομή struct s μόνο όταν αυτό απαιτείται (π.χ. κατά την εισαγωγή ενός νέου φοιτητή)

Ολοκληρωμένο Παράδειγμα

- Στη συνέχεια θα παρουσιαστεί κώδικας για την υλοποίηση ενός συστήματος διατήρησης των δεδομένων των φοιτητών του Τμήματος Επιστήμης Υπολογιστών, το οποίο θα υποστηρίζει τις εξής λειτουργίες:
  - Εισαγωγή νέου φοιτητή (`InsertNewStudent`)
  - Αναζήτηση στοιχείων φοιτητή (`SearchInfosForStudent`)
  - Ενημέρωση μέσου όρου φοιτητή (`UpdateInfosOfStudent`)
  - Διαγραφή κάποιου φοιτητή (`DeleteStudent`)

  και θα παρέχει κατάλληλο μενού για την προσπέλασή τους

Ολοκληρωμένο Παράδειγμα

```
#include <stdio.h>
#define MAX_STUDENTS 1000

struct s {                      // Δομή διατήρησης των στοιχείων κάθε φοιτητή
    int am;
    float average;
};

int students_count = 0;         // Μετρητής του πλήθους των καταχωρημένων φοιτητών

void InsertNewStudent (float average);
void SearchInfosForStudent (int am);
void UpdateInfosOfStudent (int am, float average);
void DeleteStudent (int am);

void main (void)
{
    struct s *CSD[MAX_STUDENTS];        // Θεωρούμε πως αρχικά όλοι οι δείκτες είναι null
    int choice = 0, am;
    float average;

    while (choice != 5) {
          printf ("Διάλεξε ένα από τα παρακάτω: \n");       // Προβολή μενού στο χρήστη
          printf ("1. Εισαγωγή νέου φοιτητή \n");
          printf ("2. Αναζήτηση στοιχείων φοιτητή \n");
          printf ("3. Ενημέρωση στοιχείων φοιτητή \n");
          printf ("4. Διαγραφή φοιτητή \n");
```


Ολοκληρωμένο Παράδειγμα

```
          printf ("5. Εξοδος \n");
          scanf ("%d", &choice);
          switch (choice) {             // Ανάλογα με την επιλογή του χρήστη ...
                  case 1 :  printf ("Δώσε τον μέσο όρο του νέου φοιτητή: ");
                            scanf ("%d", &average);
                            InsertNewStudent (average);
                  break;
                  case 2 :  printf ("Δώσε το Α.Μ. του φοιτητή: ");
                            scanf ("%d", &am);
                            SearchInfosForStudent (am);
                  break;
                  case 3 :  printf ("Δώσε το Α.Μ. του φοιτητή: ");
                            scanf ("%d", &am);
                            printf ("Δώσε τον νέο μέσο όρο του φοιτητή");
                            scanf ("%d", &average);
                            UpdateInfosOfStudent (am, average);
                  break;
                  case 4 :  printf ("Δώσε το Α.Μ. του φοιτητή: ");
                            scanf ("%d", &am);
                            DeleteStudent (am);
                  break;
                  case 4 :  printf ("Δώσε το Α.Μ. του φοιτητή: ");
                            scanf ("%d", &am);
                            DeleteStudent (am);
                  break;
          }
    }
}
```


Ολοκληρωμένο Παράδειγμα

```
void InsertNewStudent (float average)
{
    CSD[students_count] = (struct s *) malloc(sizeof(struct s));
    CSD[students_count]->am = students_count;
    CSD[students_count]->float = average;
    students_count ++;
}

void SearchInfosForStudent (int am)
{
    if (CSD[am] != null)
          printf ("Ο μέσος όρος του φοιτητή με Α.Μ.:%d είναι: \n", am, CSD[am]->average);
    else  printf ("Δεν υπάρχει φοιτητής με Α.Μ.:%d\n", am);
}

void UpdateInfosOfStudent (int am, float average)
{
    if (CSD[am] != null)
          CSD[am]->average = average;
}

void UpdateInfosOfStudent (int am, float average)
{
    if (CSD[am] != null)
          free (CSD[am]);
          CSD[am] = null;
}
```


Αναδρομή

- Η C επιτρέπει σε μια συνάρτηση να καλεί τον εαυτό της, κάτι το οποίο καλείται **αναδρομή**

Παράδειγμα:

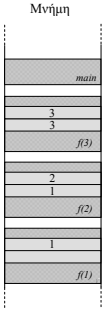
```
int f (int n);

void main (void)
{
    printf ("res = %d", f(3));
}

void f (int *n)
{
    int t;
    if (n == 1) return 1;
    t = f(n-1);
    return (t+n);
}
```

Εκτέλεση:

```
f(3);
  if (3==1) Ψευδές
  f(2);
      if (2==1) Ψευδές
      f(1);
          if (1==1) Αληθές
          return(1);
      t = 1;
      return (1+2);
  t = 3;
  return (3+3);
```

Έξοδος Παραδείγματος:

```
res = 6
```



- Παρατηρούμε ότι κάθε κλήση της συνάρτησης f, έχει διαφορετικές μεταβλητές n και t