Υποχρεωτικό Μάθημα 3ου Εξαμήνου
ΗΥ 240: Δομές Δεδομένων
Χειμερινό Εξάμηνο Ακ. Έτους 2011-2012
Τμήμα Επιστήμης Υπολογιστών
Πανεπιστήμιο Κρήτης

Εισαγωγή στον Προγραμματισμό (με τη C)

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

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

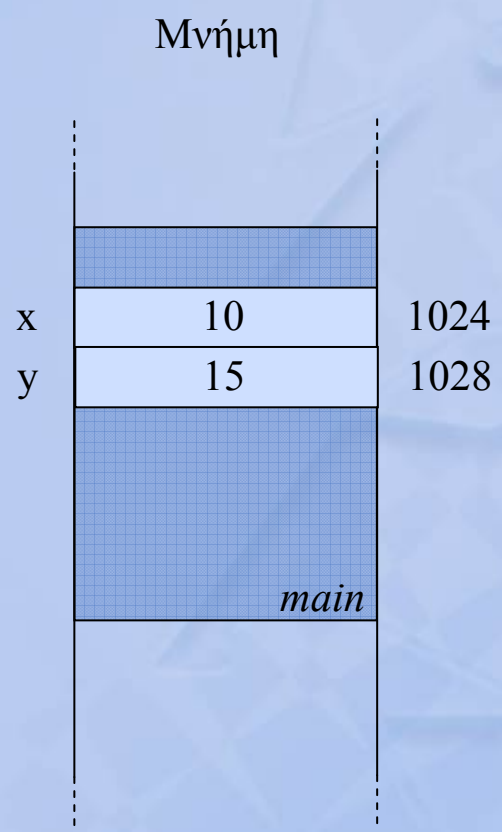
1ο Μέρος

- Μεταβλητές
- Δομές (Structs)
- Πίνακες (Arrays)
- if … else if … else
- break
- switch … case
- Συναρτήσεις
- Εμβέλεια μεταβλητών
- Παράδειγμα

Μεταβλητές

Παράδειγμα:

```
void main (void)
{
    int x;
    int y;

    x = 10;
    y = x+5;

    printf ("x = %d", x);
    printf ("y = %d", y);
}
```



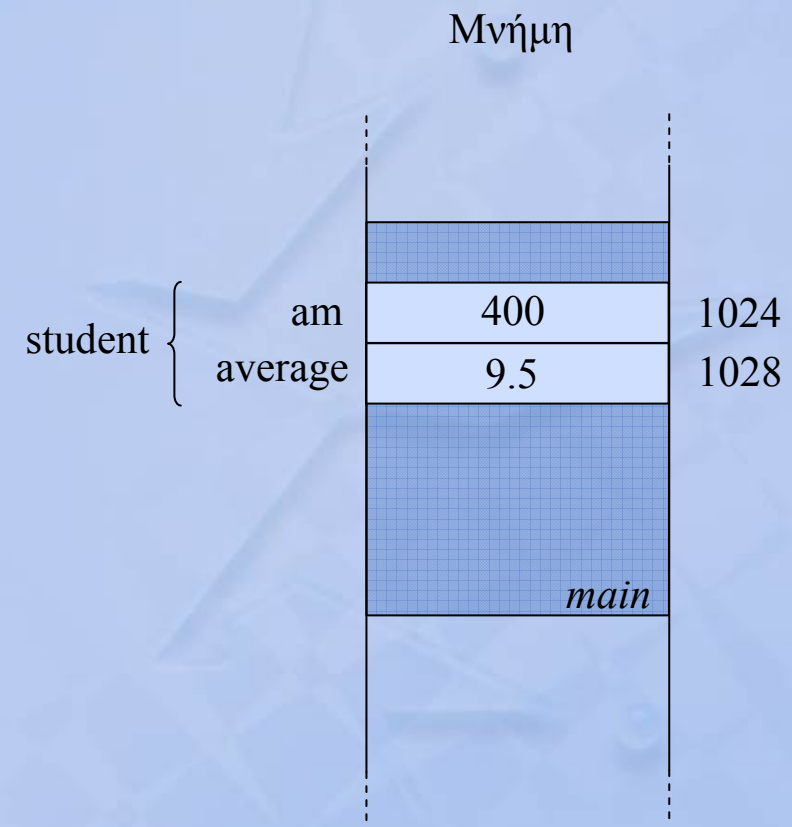
Έξοδος παραδείγματος:

```
x = 10;
y = 15;
```

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:

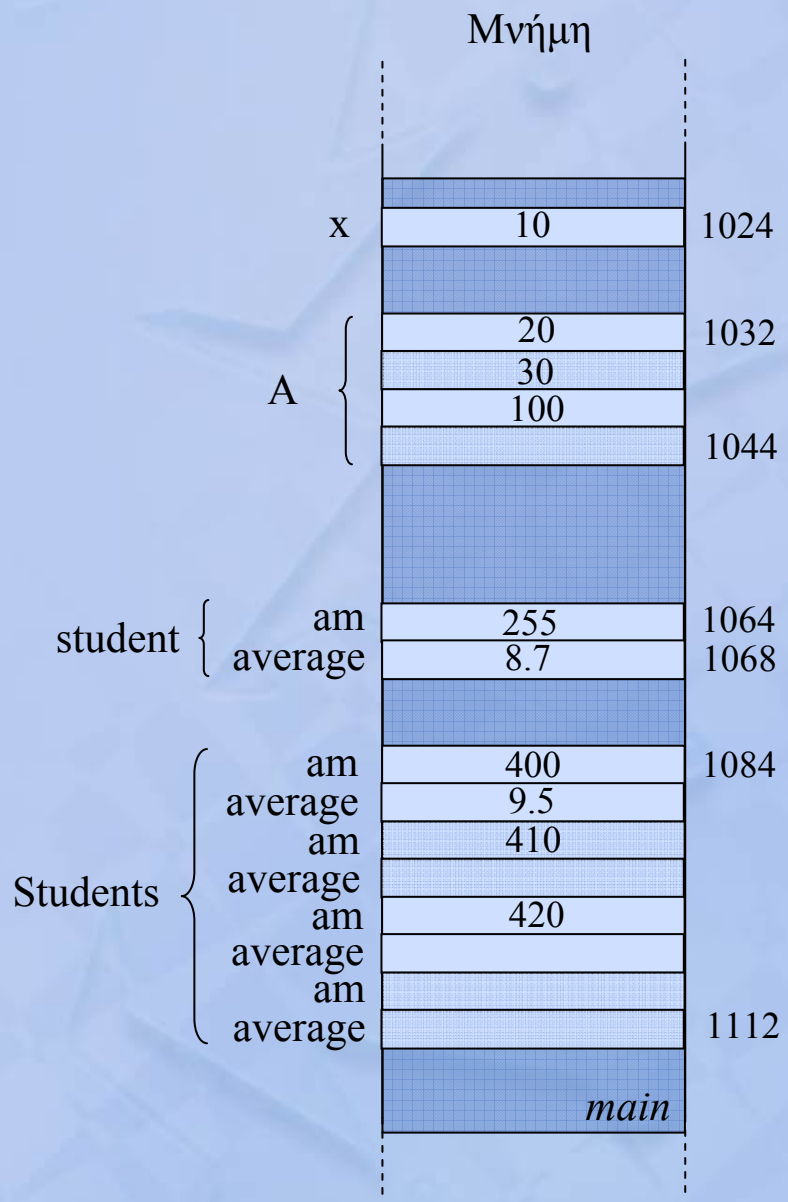
```
A.M.: 400 -- M.O.: 9.5
```

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

Πίνακες (Arrays)

Παράδειγμα:

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

void main (void)
{
    int x;
    int A[4];
    struct s student;
    struct s Students[4];

    x = 10;

    A[0] = 20;
    A[1] = 30;
    A[2] = 10*x;

    student.am = 255;
    student.average = 8.7;

    Students[0].am = 400;
    Students[0].average = 9.5;
    Students[1].am = 410;
    Students[2].am = 420;
}
```



If … else if … else

Έστω ότι θέλουμε να υλοποιήσουμε ένα πρόγραμμα το οποίο θα επιτρέπει στο χρήστη

- να προσθέτει,
- να αφαιρεί,
- και να πολλαπλασιάζει

2 ακεραίους.

```
void main (void)
{
    int num1, num2;
    int choice;

    printf ("Δώσε τον πρώτο ακέραιο: \n");
    scanf ("%d", &num1);

    printf ("Δώσε τον δεύτερο ακέραιο: \n");
    scanf ("%d", &num2);

    printf ("Ποια πράξη θέλεις να εκτελέσεις; ");
    printf ("1 = Πρόσθεση");
    printf ("2 = Αφαίρεση");
    printf ("3 = Πολλαπλασιασμός");
    printf ("Επέλεξε: \n");
    scanf ("%d",&choice);

    if (choice == 1)
          printf ("%d+%d=%d\n",num1,num2,num1+num2);
    else if (choice == 2)
          printf ("%d-%d=%d\n",num1,num2,num1-num2);
    else if (choice == 3)
          printf ("%d*%d=%d\n",num1,num2,num1*num2);
    else
          printf ("Λάθος Επιλογή");
}
```

Break

- Η εκτέλεση της εντολής **break** έχει ως αποτέλεσμα την ***έξοδο*** του προγράμματος ***από τον βρόγχο*** στον οποίο αυτή περιέχεται.
  - βρόγχοι for, do…while, while.

```
#define N 1000;

void main (void)
{
    int counter = 0;
    int A[N];

    /* Αύξηση του μετρητή counter έως το 100 */
    while (1) {
       counter = counter + 1;
       if (counter == 100)
          break;
    }

// Εύρεση της θέσης του στοιχείου 10 στον πίνακα A
    for (i=0; i<N; i++) {
       if (A[i] == 10) {
          printf ("Το 10 είναι στη θέση %d", i);
          break;
       }
    }
}
```

```
void main (void)
{
    int counter1 = 0, counter2 = 0;

    while (1) {
       counter1++;
       if (counter1 == 100)
          break;

       counter2 = 0;
       while (1) {
          counter2+=2;
          if (counter2 == 200)
             break;
       }
    }
}
```

If … else if … else

Έστω ότι θέλουμε να υλοποιήσουμε ένα πρόγραμμα το οποίο θα επιτρέπει στο χρήστη

- να προσθέτει,
- να αφαιρεί,
- και να πολλαπλασιάζει

2 ακεραίους.

```
void main (void)
{
    int num1, num2;
    int choice;

    while (1) {
      printf ("Δώσε τον πρώτο ακέραιο: \n");
      scanf ("%d", &num1);

      printf ("Δώσε τον δεύτερο ακέραιο: \n");
      scanf ("%d", &num2);

      printf ("Ποια πράξη θέλεις να εκτελέσεις; ");
      printf ("1 = Πρόσθεση");
      printf ("2 = Αφαίρεση");
      printf ("3 = Πολλαπλασιασμός");
      printf ("4 = Έξοδος");
      printf ("Επέλεξε: \n");
      scanf ("%d",&choice);

      if (choice == 1)
          printf ("%d+%d=%d\n",num1,num2,num1+num2);
      else if (choice == 2)
          printf ("%d-%d=%d\n",num1,num2,num1-num2);
      else if (choice == 3)
          printf ("%d*%d=%d\n",num1,num2,num1*num2);
      else if (choice ==4)
          break;
      else
          printf ("Λάθος Επιλογή");
    }
}
```

Switch … case

Έστω ότι θέλουμε να υλοποιήσουμε ένα πρόγραμμα το οποίο θα επιτρέπει στο χρήστη

- να προσθέτει,
- να αφαιρεί,
- και να πολλαπλασιάζει

2 ακεραίους.

```
void main (void)
{
    int num1, num2;
    int choice = 0;

    while (choice!=4) {
      printf ("Δώσε τον πρώτο ακέραιο: \n");
      scanf ("%d", &num1);

      printf ("Δώσε τον δεύτερο ακέραιο: \n");
      scanf ("%d", &num2);

      printf ("Ποια πράξη θέλεις να εκτελέσεις; ");
      printf ("1 = Πρόσθεση");
      printf ("2 = Αφαίρεση");
      printf ("3 = Πολλαπλασιασμός");
      printf ("4 = Έξοδος");
      printf ("Επέλεξε: \n");
      scanf ("%d",&choice);

      switch (choice) {
          case 1: printf ("%d+%d=%d\n",num1,num2,num1+num2);
          break;
          case 2: printf ("%d-%d=%d\n",num1,num2,num1-num2);
          break;
          case 3: printf ("%d*%d=%d\n",num1,num2,num1*num2);
          break;
          case 4: printf ("Εξοδος από το πρόγραμμα");
          break;
          default: printf ("Λάθος Επιλογή");
          break;
      }
    }
}
```

Συναρτήσεις

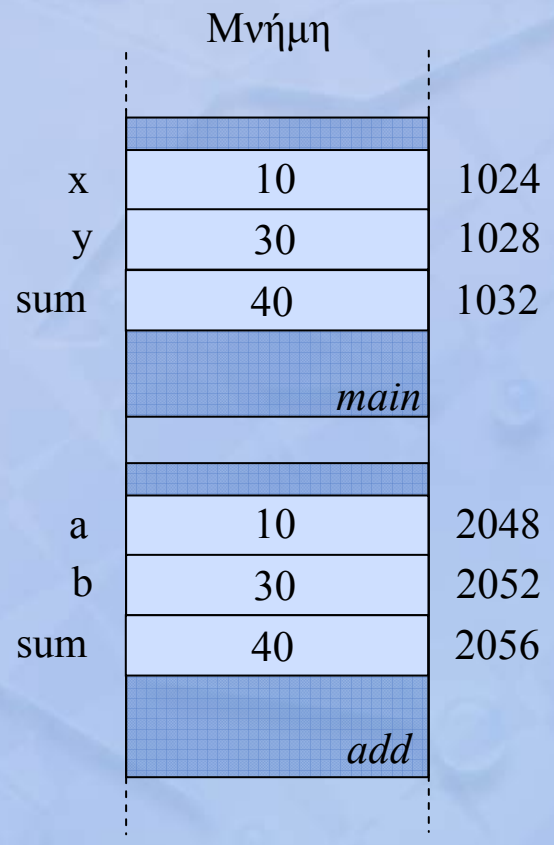
- Παίρνουν κάποια ορίσματα, εκτελούν κάποια λειτουργία και επιστρέφουν μία τιμή
- Είναι αναγκαίο να ορίζουμε τη συνάρτηση ή το *πρωτότυπό* της πριν τη συνάρτηση main του προγράμματός μας
- Κάθε συνάρτηση εκτελείται στο **δικό της χώρο μνήμης**

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);
    f2();
    f1();
}

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
```

Ολοκληρωμένο Παράδειγμα - Χρήση Πινάκων και Δομών

- Έστω ότι η γραμματεία του Τμήματος Επιστήμης Υπολογιστών, επιθυμεί να διατηρήσει διάφορα στοιχεία για τους προπτυχιακούς φοιτητές.
  - Θεωρούμε πως επιθυμεί να διατηρήσει το A.M. (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 είναι ένας πίνακας από δομές struct s, δηλαδή **`struct s CSD[MAX_STUDENTS]`**
  - *Ποια είναι η μορφή του πίνακα αυτού στη μνήμη;*

Ολοκληρωμένο Παράδειγμα - Χρήση Πινάκων και Δομών

- 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`)

  και θα παρέχει κατάλληλο μενού για την προσπέλασή τους

Ολοκληρωμένο Παράδειγμα - Χρήση Πινάκων και Δομών

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

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

struct s CSD[MAX_STUDENTS];     // Υποθέτουμε πως αρχικά τα πεδία am και average
                                // κάθε στοιχείου του CSD είναι -1
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 main (void)
{
    int choice = 0, am;
    float average;

    while (choice != 4) {
          printf ("Ποια λειτουργία θέλεις να εκτελέσεις;\n");    // Προβολή μενού στο χρήστη
          printf ("1. Εισαγωγή νέου φοιτητή \n");
          printf ("2. Αναζήτηση στοιχείων φοιτητή \n");
          printf ("3. Ενημέρωση στοιχείων φοιτητή \n");
```

Ολοκληρωμένο Παράδειγμα - Χρήση Πινάκων και Δομών

```
        printf ("4. Εξοδος \n");
        printf ("Επέλεξε: \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 A.M. του φοιτητή: ");
                          scanf ("%d", &am);
                          SearchInfosForStudent (am);
                break;
                case 3 :  printf ("Δώσε το A.M. του φοιτητή: ");
                          scanf ("%d", &am);
                          printf ("Δώσε τον νέο μέσο όρο του φοιτητή");
                          scanf ("%d", &average);
                          UpdateInfosOfStudent (am, average);
                break;
                case 4 : printf ("Εξοδος από το πρόγραμμα");
                break;
                default : printf ("Λάθος επιλογή");
                break;
        }
    }
} // End of main
```

Ολοκληρωμένο Παράδειγμα - Χρήση Πινάκων και Δομών

```
void InsertNewStudent (float average)
{
    CSD[students_count].am = students_count;
    CSD[students_count].float = average;
    printf ("Ο νέος φοιτητής εισήχθη επιτυχώς με Α.Μ. %d\n", students_count);

    students_count ++;
}

void SearchInfosForStudent (int am)
{
    if (CSD[am].am != -1)
          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].am != -1)
          CSD[am].average = average;
}
```

Παρατηρήσεις:

- Σε κάθε συνάρτηση θα έπρεπε να εξασφαλίζουμε πως δεν βγαίνουμε **εκτός των ορίων του πίνακα**

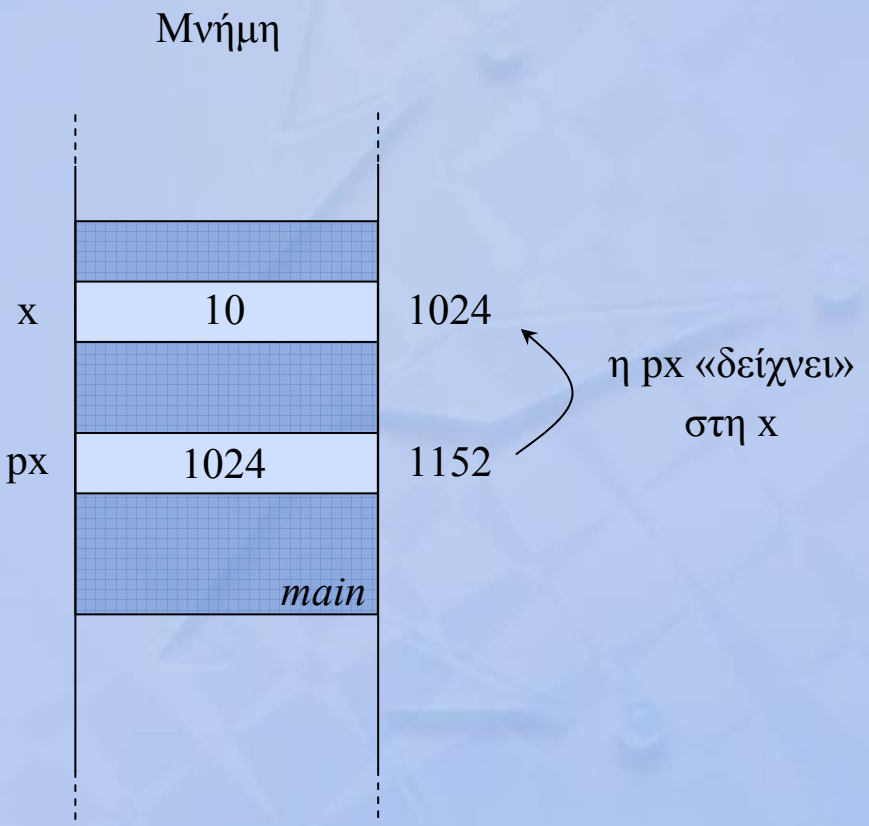
2ο Μέρος

- Δείκτες (Pointers)
- Δομές (Structs)
- Πίνακες (Arrays)
- Malloc
- free
- Παράδειγμα

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:

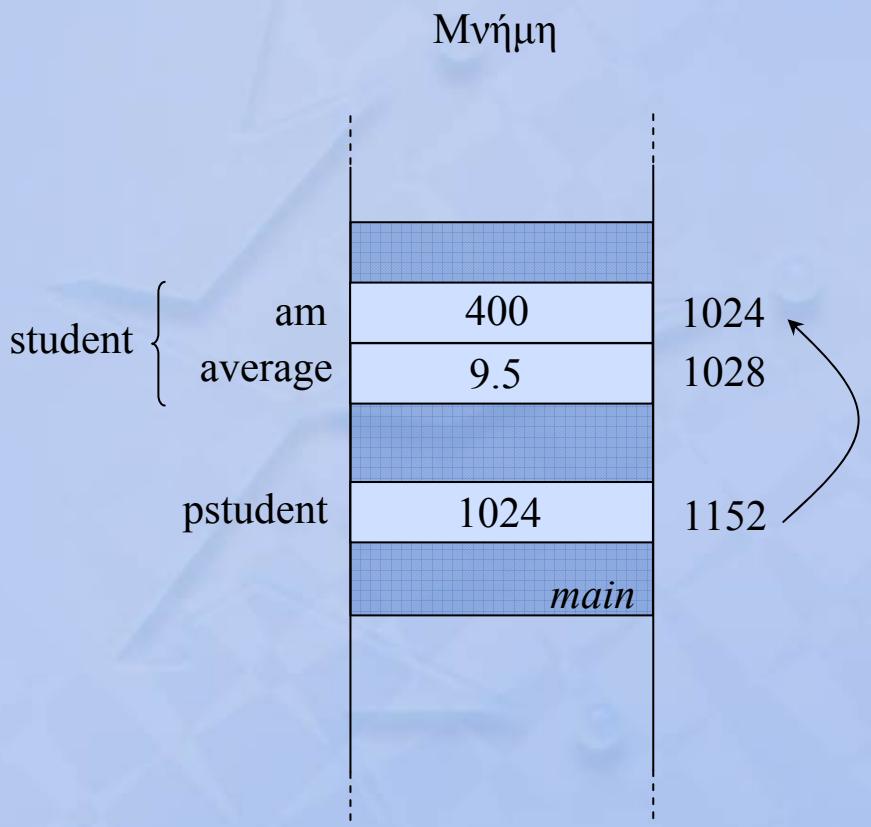
```
x = 10;
px = 1024;
*px = 10;
```

- 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;
    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:

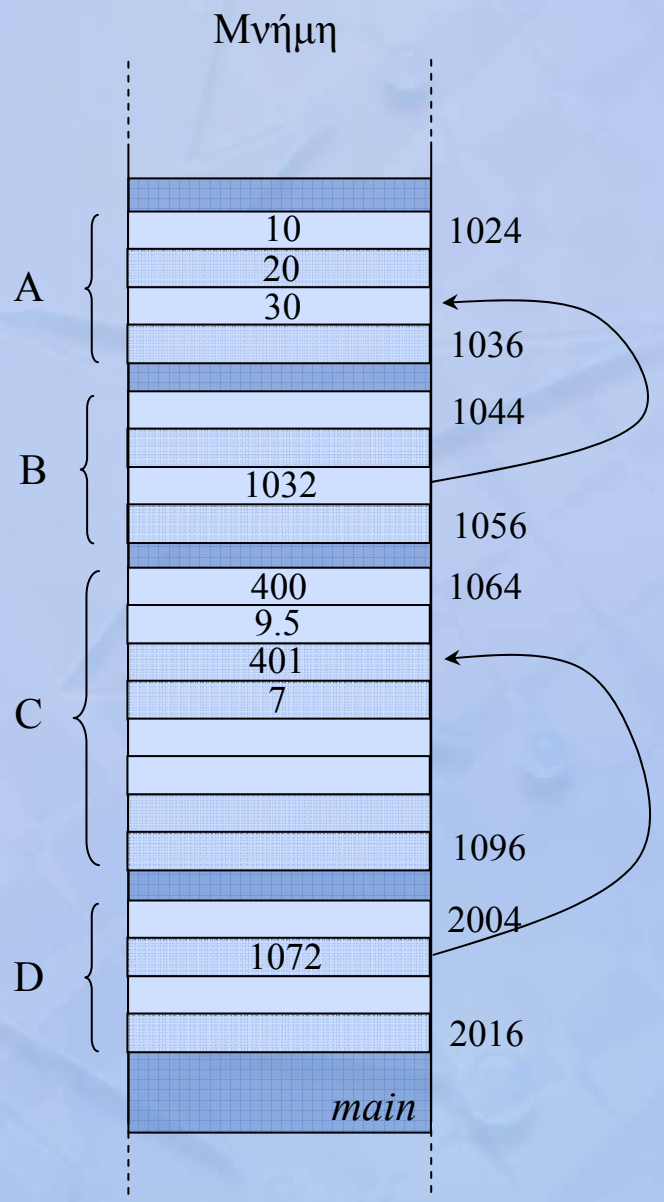
```
A.M.: 400 -- M.O.: 9.5
```

- Χρησιμοποιούμε **px** -> <όνομα στοιχείου> : για να προσπελάσουμε τα στοιχεία μιας δομής 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];
    *B[2] = 30;

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

    D[1] = &C[1];
    D[1]->am = 401;
    D[1]->average = 7;
}
```

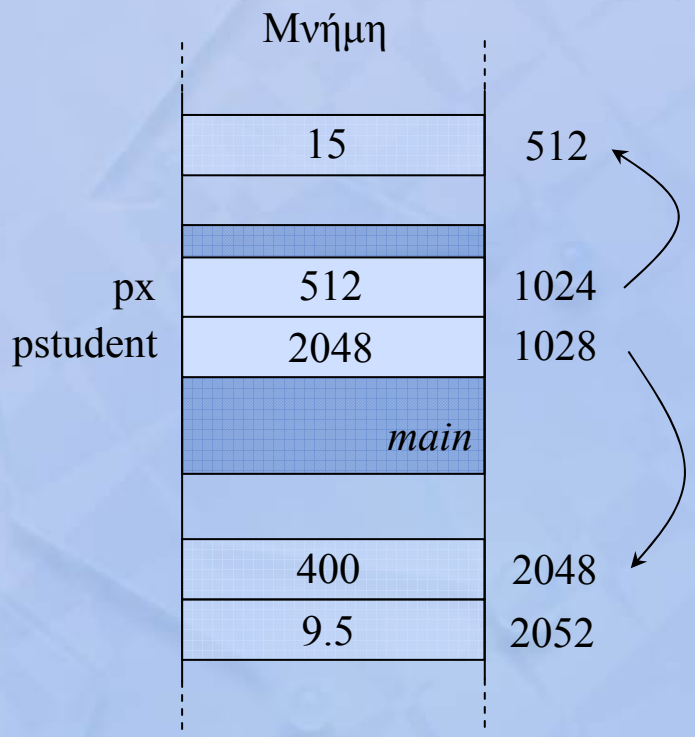


Malloc

- 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**)
  - ο δεσμευμένος χώρος μνήμης **δεν ανήκει** σε κάποια συνάρτηση ή πρόγραμμα

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;
}
```

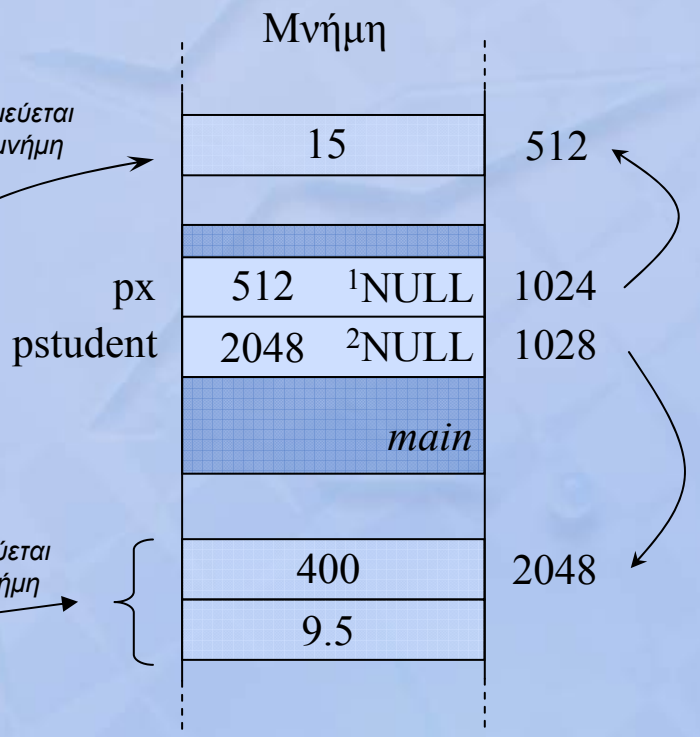


Free

- Η συνάρτηση αυτή χρησιμοποιείται για την ***επιστροφή*** στο σύστημα της μνήμης που έχει δεσμευθεί με τη χρήστη της συνάρτησης malloc
- Παίρνει ως όρισμα έναν **δείκτη** στη μνήμη που επιθυμούμε να αποδεσμεύσουμε
  - ο δείκτης αυτός είναι ίδιος με τον δείκτη που επιστράφηκε από τη malloc

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;

    free (px);
    free (pstudent);

    px=NULL;[1]
    pstudent=NULL;[2]
}
```



Σημαντικό: Μια καλή πρακτική είναι μετά την απελευθέρωση της μνήμης στην οποία δείχνει κάποιος δείκτης ***p*** να αναθέσουμε στον ***p*** την τιμή **NULL**

Ολοκληρωμένο Παράδειγμα - Χρήση Πινάκων, Δεικτών σε Δομές

- Έστω ότι η γραμματεία του Τμήματος Επιστήμης Υπολογιστών, επιθυμεί να διατηρήσει διάφορα στοιχεία για τους προπτυχιακούς φοιτητές.
  - Θεωρούμε πως επιθυμεί να διατηρήσει το A.M. (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)
  - *Ποια είναι η μορφή του πίνακα αυτού;*

Ολοκληρωμένο Παράδειγμα - Χρήση Πινάκων, Δεικτών σε Δομές

- 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 φοιτητή (`DeleteStudent`)**

  και θα παρέχει κατάλληλο μενού για την προσπέλασή τους

Ολοκληρωμένο Παράδειγμα - Χρήση Πινάκων, Δεικτών σε Δομές

```
#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, struct s **StudentsArray);
void SearchInfosForStudent (int am, struct s **StudentsArray);
void UpdateInfosOfStudent (int am, float average, struct s **StudentsArray);
void DeleteStudent (int am, struct s **StudentsArray);

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

    while (choice != 5) {
          printf ("Ποια λειτουργία θέλεις να εκτελέσεις;\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 - Χρήση Πινάκων, Δεικτών σε Δομές

```
printf ("5. Εξοδος \n");
printf ("Επέλεξε: \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, CSD);
          break;
          case 2 :  printf ("Δώσε το A.M. του φοιτητή: ");
                    scanf ("%d", &am);
                    SearchInfosForStudent (am, CSD);
          break;
          case 3 :  printf ("Δώσε το A.M. του φοιτητή: ");
                    scanf ("%d", &am);
                    printf ("Δώσε τον νέο μέσο όρο του φοιτητή");
                    scanf ("%d", &average);
                    UpdateInfosOfStudent (am, average, CSD);
          break;
          case 4 :  printf ("Δώσε το A.M. του φοιτητή: ");
                    scanf ("%d", &am);
                    DeleteStudent (am, CSD);
          break;
          case 5 : printf ("Εξοδος από το πρόγραμμα");
          break;
          default : printf ("Λάθος επιλογή");
          break;
} } } // End of main
```

Ολοκληρωμένο Παράδειγμα - Χρήση Πινάκων, Δεικτών σε Δομές

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

     printf ("Ο νέος φοιτητής εισήχθη επιτυχώς με Α.Μ. %d\n", students_count);

     students_count ++;
}

void SearchInfosForStudent (int am, struct **StudentsArray)
{
     if (StudentsArray[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, struct **StudentsArray)
{
     if (StudentsArray[am] != null)
             StudentsArray[am]->average = average;
}

void DeleteStudent (int am, float average, struct **StudentsArray)
{
     if (StudentsArray[am] != null)
            free (StudentsArray[am]);
            StudentsArray[am] = null;
}
```

3ο Μέρος

- Συναρτήσεις
- Εμβέλεια μεταβλητών
- Παράδειγμα Ανταλλαγής τιμών
  - με χρήση συνάρτησης
- Χρησιμότητα δείκτη σε δείκτη
- Χρησιμότητα της Malloc
- Αναδρομή

Συναρτήσεις

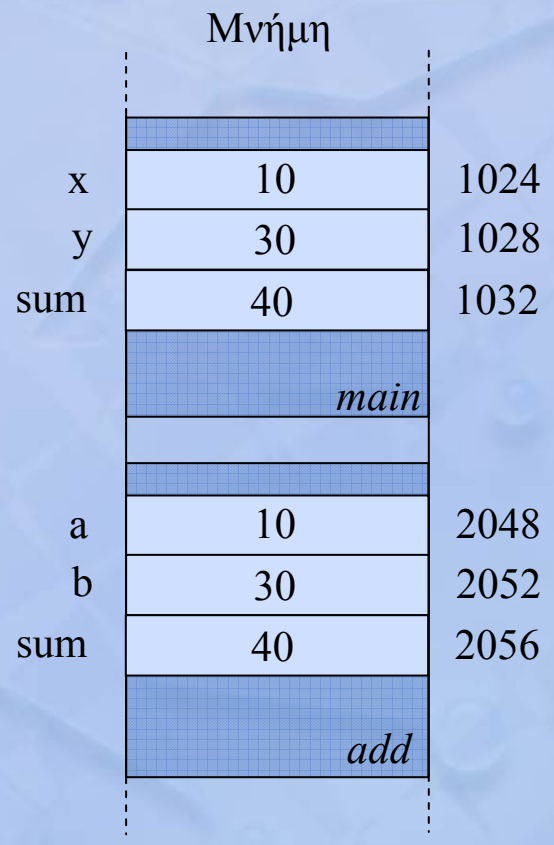
- Παίρνουν κάποια ορίσματα, εκτελούν κάποια λειτουργία και επιστρέφουν μία τιμή
- Είναι αναγκαίο να ορίζουμε τη συνάρτηση ή το *πρωτότυπό* της πριν τη συνάρτηση main του προγράμματός μας
- Κάθε συνάρτηση εκτελείται στο **δικό της χώρο μνήμης**

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
```

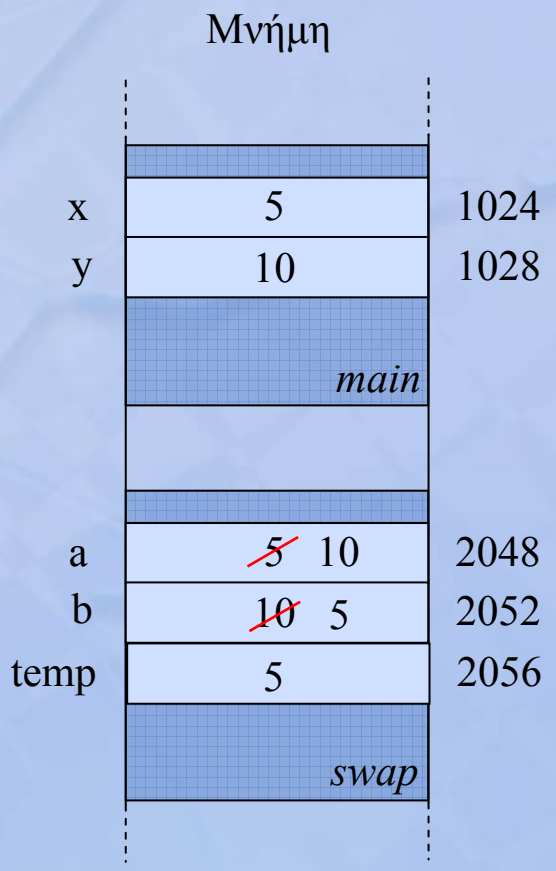
Ο 2ος αλγόριθμος είναι σωστός

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
```

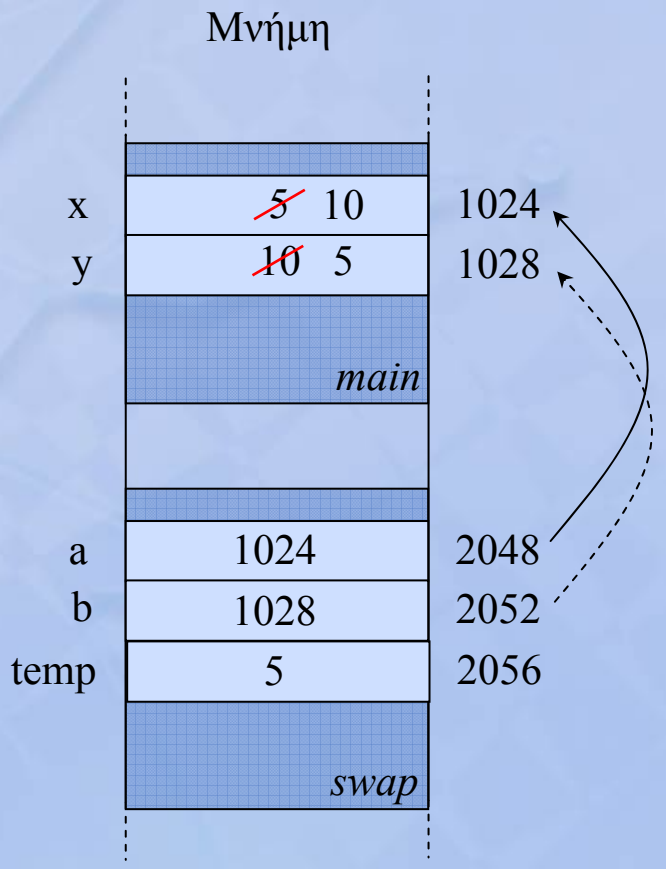
Ο 3ος αλγόριθμος είναι λανθασμένος

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);
        printf ("Μετά την ενημέρωση x = %d", x);
}
```

Αυτό δημιουργεί πρόβλημα;

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.

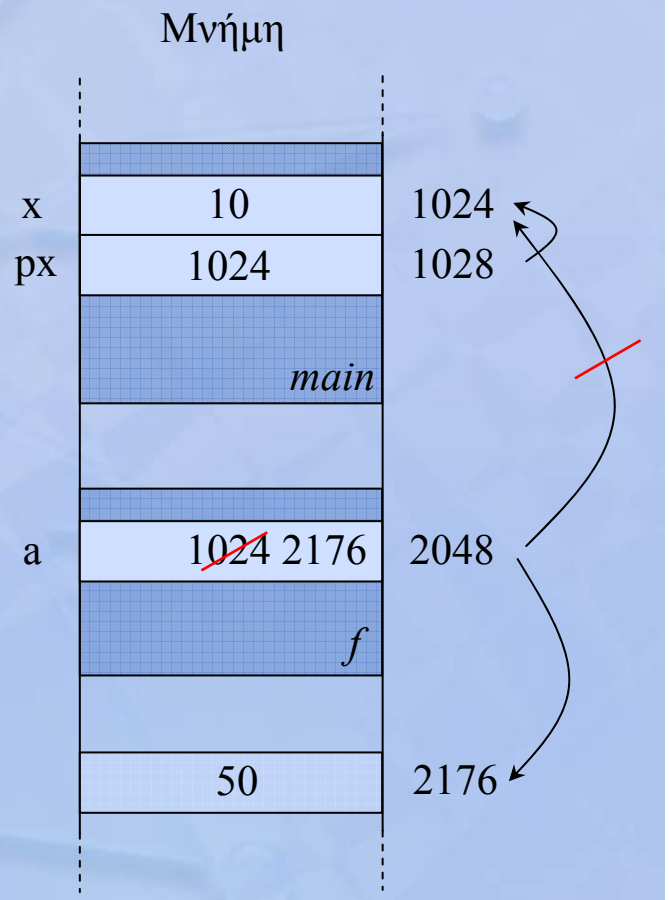
- Χρειάζεται προσοχή εάν χρησιμοποιηθεί η μέθοδος του περάσματος με αναφορά της x στην f.
  - Θα πρέπει το όρισμα της f να οριστεί καταλλήλως και να είναι **δείκτης σε δείκτη**.
  - (παρουσιάζονται παραδείγματα στις επόμενες διαφάνειες ...)

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:

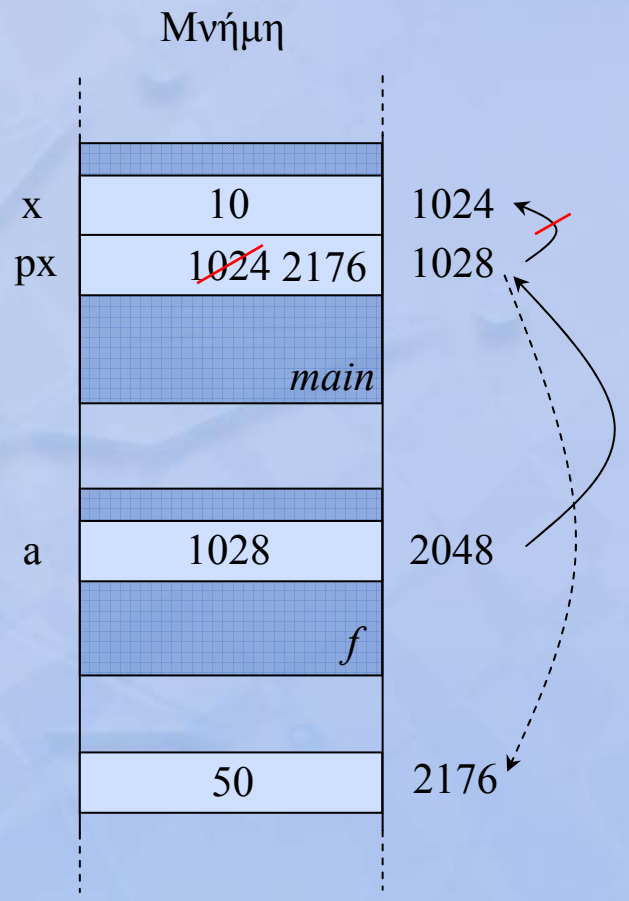
```
Πριν την ενημέρωση: px=1024, *px=10
Μετά την ενημέρωση: px=1024, *px=10
```

- **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

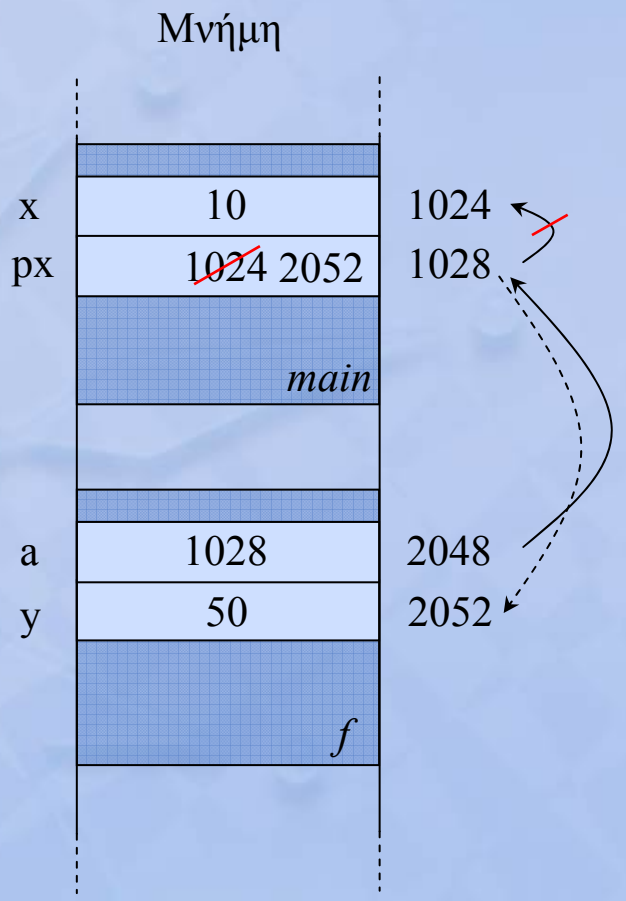
- **Ερώτηση: Με ποιο τρόπο περνάμε την μεταβλητή px στη συνάρτηση f; (με τιμή ή με αναφορά;)**

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.

Αναδρομή

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

Παράδειγμα:

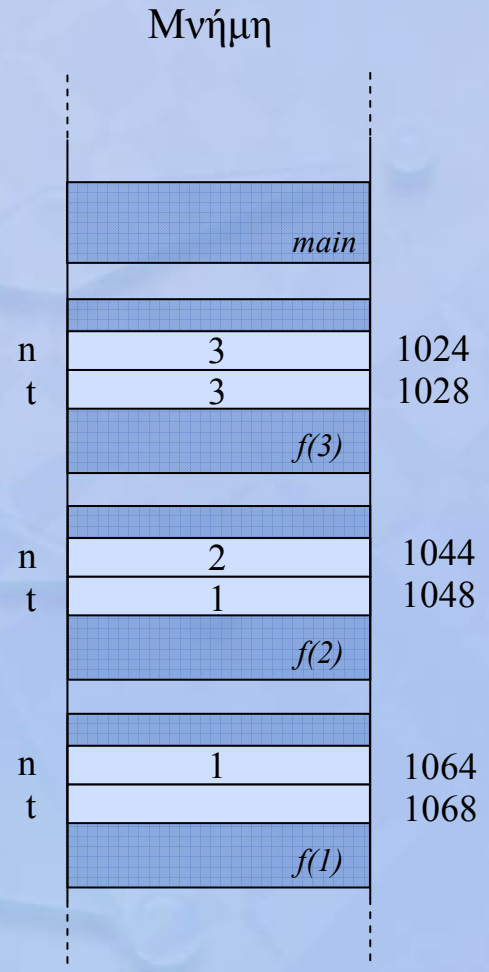
```
int f (int n);

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

int 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