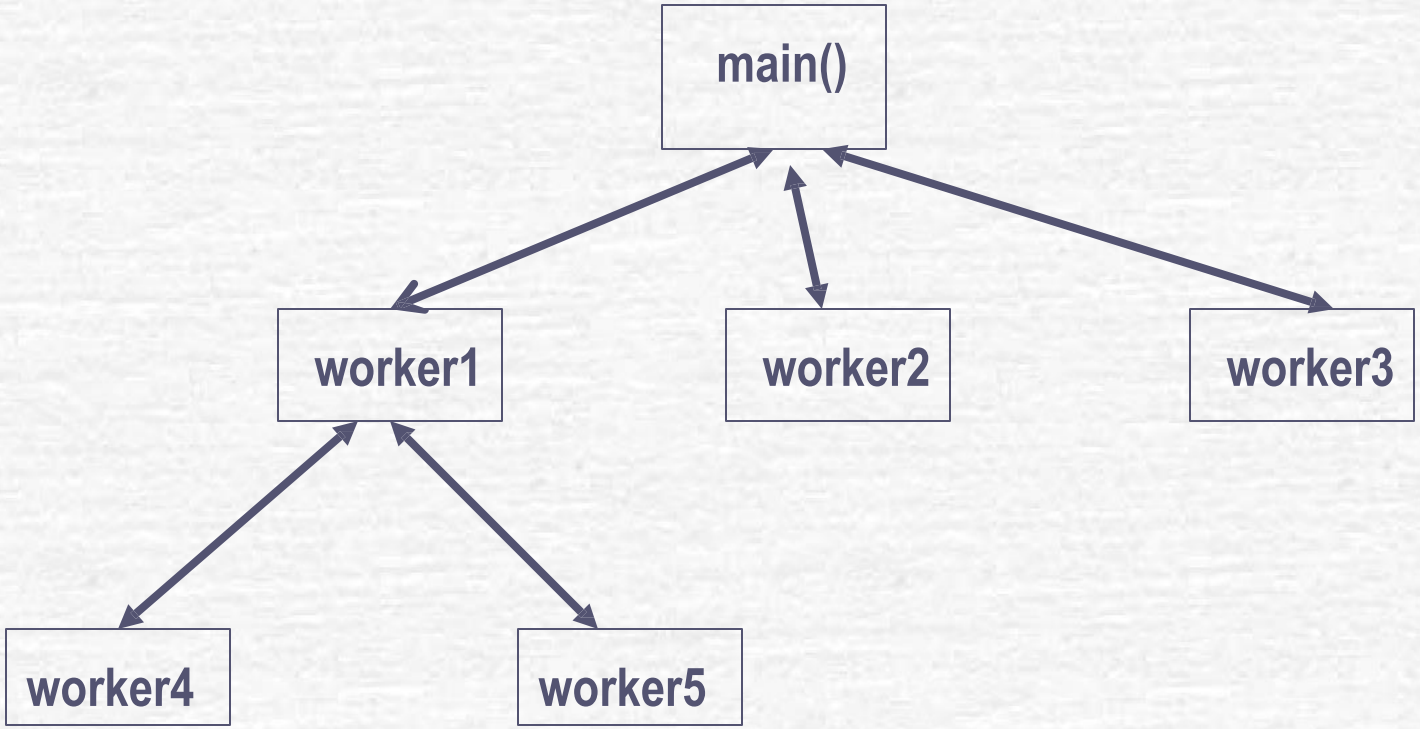
Τμήμα Μηχανικών Η/Υ & Πληροφορικής
Πανεπιστήμιο Πατρών

Εισαγωγή στον Προγραμματισμό Υπολογιστών

Η γλώσσα προγραμματισμού C

Συναρτήσεις (Functions)



Υποπρογράμματα

- Αποτελούν τον τρόπο εφαρμογής (υλοποίησης) της τμηματοποίησης σε ένα πρόγραμμα.
- Κάθε υποπρόγραμμα είναι ένα αυτόνομο τμήμα (μικρό πρόγραμμα)
- Ο συνδυασμός των υποπρογραμμάτων συνιστά το (αρχικό) πρόγραμμα.
- Πλεονεκτήματα
  - Αποφυγή επαναλήψεων
  - Αύξηση επαναχρησιμοποίησης
  - Βελτίωση αναγνωσιμότητας
  - Ευκολότερη συντήρηση

Συναρτήσεις στη C

- Πρόγραμμα C = σύνολο συναρτήσεων
- Η συνάρτηση main αντιπροσωπεύει το κυρίως πρόγραμμα, δηλ. τον τρόπο με τον οποίο συνδυάζονται οι υπόλοιπες συναρτήσεις για τη λύση του προβλήματος.
- Μέρη συνάρτησης
  - Κεφαλίδα = η διεπαφή της συνάρτησης (όνομα, είσοδος, έξοδος)
  - Σώμα = υλοποίηση/ορισμός της συνάρτησης
- Προτάσεις συνάρτησης
  - Δήλωση – Κλήση - Ορισμός

Δήλωση Συνάρτησης

- Προσδιορίζεται η διεπαφή, ο τρόπος αναφοράς στη συνάρτηση:

  <τύπος> <όνομα – συν> ([<παράμετροι>]); Όπου

  <παράμετροι>:=<τύπος1> [<όνομ-παρ1>], …, <τύποςN> [<όνομ-παρN]

  τυπικά ορίσματα (ή είσοδοι)

  τυπικά αποτελέσματος (εξόδου)

- Αν η συνάρτηση δεν επιστρέφει κάποια τιμή, τότε χρησιμοποιείται σαν τύπος αποτελέσματος η λέξη κλειδί void.

Παραδείγματα

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

int min (int, int);

double exp ( double m, int n);

void swap (int a, int b);

char *getptr (char * str, char ch);

char *getptr (char str [ ], char ch);
```

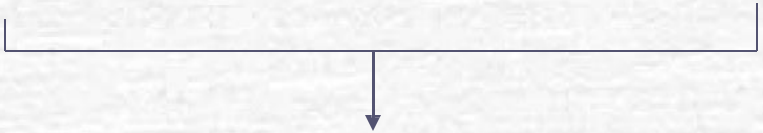
Παράδειγμα

```
#include <stdio.h>
int square(int y); /* αρχέτυπο συνάρτησης */
int main ( )
{
    int x;
    for (x=1; x<=10; x++)
            printf("%d", square(x));
    printf("\n");
    return 0;
}
/* ορισμός συνάρτησης */
int square( int y)
{
    return y * y;
}
```

Κλήση Συνάρτησης

- Καλείται η συνάρτηση για εκτέλεση με συγκεκριμένα ορίσματα

<όνομα – συν> (<ορισ1>, <ορισ2>, …, <ορισN>);



πραγματικά ορίσματα (σταθερές, μεταβλητές, εκφράσεις|)

- Τα πραγματικά ορίσματα πρέπει να είναι του ίδιου αριθμού και τύπου με τα τυπικά ορίσματα

Παραδείγματα

```
swap ( x, y);

draw_circle (a/2.0, 2.0 * b, c);

max_num = max(num1, num2);

min_num = (num, 5);

x = y + max (num1/2.0, 3.0*num2);

printf ( “ο μέγιστος είναι: %d\n”, max(x1,x2));
```

Ορισμός Συνάρτησης

- Για κάθε μη ενσωματωμένη (δική μας) συνάρτηση πρέπει να ορίσουμε το σώμα της στο πρόγραμμα (μετά την main)

```
<τύπος> (<όνομα - συν> (<παράμετροι>)
        {
                <δηλώσεις τοπικών μεταβλητών>
                <προτάσεις>
        }
```

Παράδειγμα (1)

```
double embadon (double platos, double mikos)
{
double apotelesma;
apotelesma = platos * mikos;
platos=15;
 return (apotelesma);
}
```

τυπικές παράμετροι

τοπική μεταβλητή

Επιστροφή ελέγχου και τιμής στην καλούσα συνάρτηση

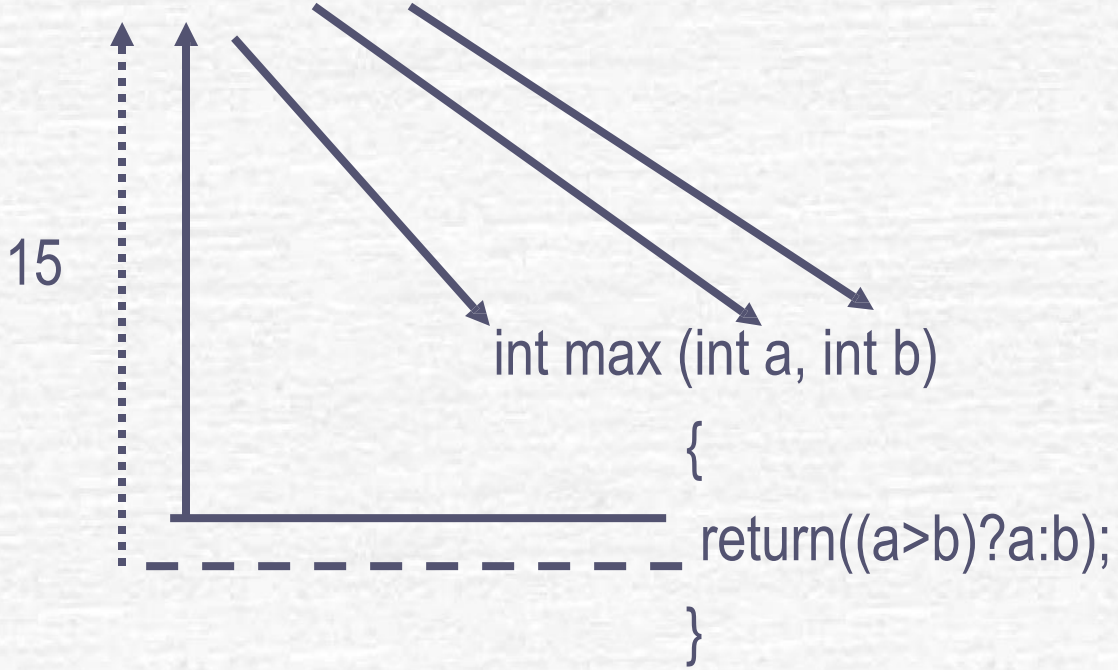
Παράδειγμα (2)

```
int max (int a, int b )
{
    int max;
    max = ( a > b) ? a : b;
    return (max);
}

int max( int a, int b)
{
    return ((a > b) ? a : b);
}
```

Μηχανισμός Κλήσης (1)

max_num = max (max (15, 13), 17);

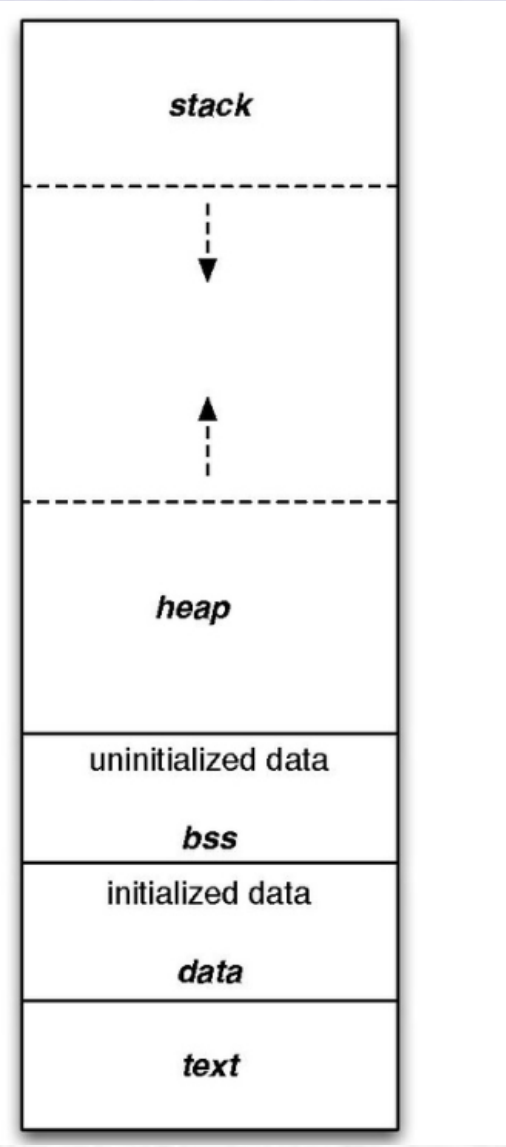


Αναδρομή (Recursion) (1)

- Ορισμός συνάρτησης μέσω κλήσης του εαυτού της
- Μία τεχνική επίλυσης προβλημάτων
- Π.χ. εύρεση αθροίσματος 1+2+…+n

- Κλασσική λύση

```
int sum (int n) {
s=0;
for (i=1; i<=n; i++)
  s=s+i;
return s;}
```



Memory Layout of a C program

https://commons.wikimedia.org/wiki/File:Program_memory_layout.pdf

Εμβέλεια Μεταβλητών

- Εμβέλεια = το τμήμα του προγράμματος στο οποίο έχει ισχύ ή είναι ορατή η μεταβλητή.
- Διάρκεια ζωής= πόσο υπάρχει στη μνήμη.

- Πότε μας αφορά η εμβέλεια;

  Όταν έχουμε μεταβλητές με το ίδιο όνομα.

Τύποι Εμβέλειας (1)

- Εξωτερικές μεταβλητές

  - Γενικές ή καθολικές μεταβλητές (global variables)
    - Δηλώνονται έξω και πάνω από κάθε συνάρτηση (και την main), στο τμήμα δηλώσεων μεταβλητών.
    - Η εμβέλειά τους είναι όλο το αρχείο του πηγαίου κώδικα που

    ανήκει η δήλωση, από το σημείο της δήλωσης και κάτω (εμβέλεια αρχείου

Τύποι Εμβέλειας (2)

- Εσωτερικές ή αυτόματες μεταβλητές
  - Τοπικές μεταβλητές (local variables)
    - Δηλώνονται μέσα σε μία συνάρτηση.
    - Η εμβέλειά τους είναι το σώμα της συνάρτησης (εμβέλεια μπλοκ)
  - Τυπικές παράμετροι συναρτήσεων
    - Η εμβέλειά τους είναι το σώμα της συνάρτησης (εμβέλεια μπλοκ)
  - Μεταβλητές σύνθετης πρότασης
    - Δηλώνονται μέσα σε μία σύνθετη πρόταση
    - Η εμβέλειά τους είναι το σημείο της δήλωσης μέχρι το τέλος της σύνθετης πρότασης (εμβέλεια μπλοκ).
  - Στατικές Μεταβλητές

- Δηλώνονται σε κάποια (ες) συνάρτηση (εις) (μετά την main) με τη λέξη κλειδί static πριν από τον τύπο μεταβλητής.

-Η εμβέλειά τους είναι το αρχείο του πηγαίου κώδικα που ανήκει η δήλωση, από το σημείο της δήλωσης και κάτω (εμβέλεια αρχείου)

Κανόνες Εμβέλειας Μεταβλητών

- Μεταβλητές με το ίδιο όνομα επιτρέπονται μόνο όταν έχουν διαφορετική εμβέλεια

- Μεταβλητή με μικρότερη εμβέλεια αποκρύπτει πιθανώς ομώνυμες μεταβλητές μεγαλύτερης εμβέλειας.

Εμβέλεια Συνάρτησης

- Οι συναρτήσεις, όπως και οι μεταβλητές έχουν εμβέλεια
- Η εμβέλεια μίας συνάρτησης εκτείνεται από το σημείο της δήλωσής της μέχρι το τέλος του προγράμματος.
- Αν μία συνάρτηση δηλωθεί static, τότε η εμβέλειά της περιορίζεται το αρχείο που δηλώθηκε.

- Μεταβλητή με μικρότερη εμβέλεια αποκρύπτει πιθανώς ομώνυμες μεταβλητές μεγαλύτερης εμβέλειας.

Διάρκεια Μεταβλητής

- Ο χρόνος δέσμευσης της μνήμης που περιέχει την τιμή της μεταβλητής
- Καθολική μεταβλητή: διάρκεια εκτέλεσης προγράμματος (πλήρης διάρκεια).
- Τοπική μεταβλητή: διάρκεια εκτέλεσης συνάρτησης (περιορισμένη διάρκεια)
- Τυπική παράμετρος: διάρκεια εκτέλεσης συνάρτησης (περιορισμένη διάρκεια)
- Στατική τοπική μεταβλητή: διάρκεια εκτέλεσης προγράμματος (πλήρης διάρκεια)

Αρχικοποίηση Μεταβλητών

- Μία τοπική μεταβλητή περιορισμένης διάρκειας αρχικοποιείται με κάθε είσοδο στο μπλοκ (συνάρτηση) που ορίζεται
- Μία τοπική μεταβλητή πλήρους διάρκειας αρχικοποιείται με την έναρξη εκτέλεσης του προγράμματος
- Π.χ. static int num;

```
func (int) {
static int count = 0;
int num =100; ... }
```

Παράδειγμα

```
#include <stdio.h>
void increment(void)

main() {
int j=0;
increment ( );
increment ( );
increment ( );
printf("%d", j}

void increment(void) {
int j=2;
static int k=2;
printf("j: %d\t k: %d\n", j++, k++); }
```

Πέρασμα Παραμέτρων

- Κατ΄αξία ή τιμή (by value)

  Η συνάρτηση δουλεύει σε αντίγραφα των πραγματικών παραμέτρων

- Κατ' αναφορά (by reference)

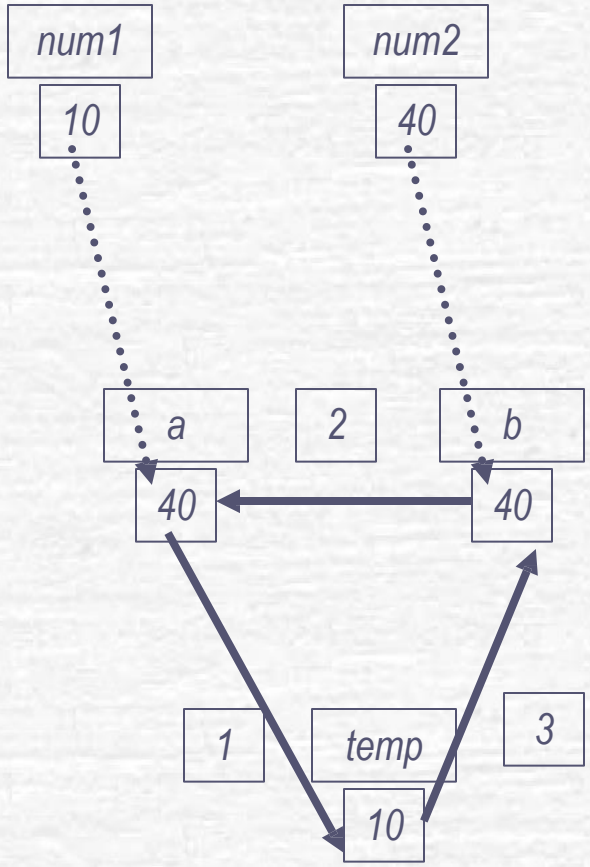
  Η συνάρτηση δουλεύει στις πραγματικές παραμέτρους (μόνο για πίνακες)

Παράδειγμα (1)

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

```
swap (num1, num2);
```

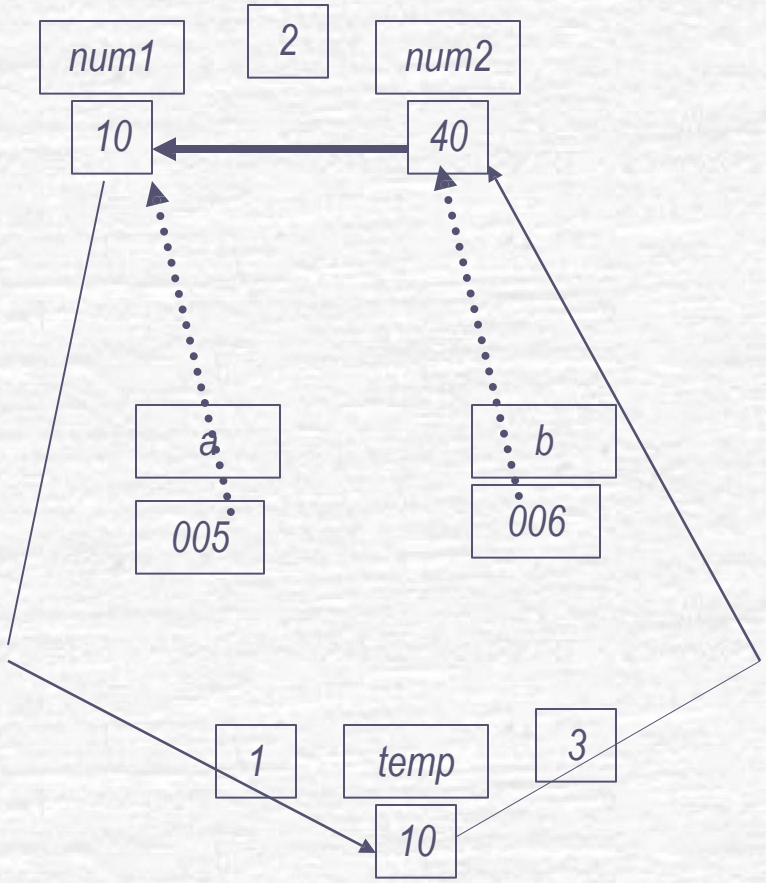
Ενώ οι τιμές των a,b αλλάζουν, οι τιμές των num1, num2 παραμένουν αμετάβλητες



Παράδειγμα (2)

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

 swap (&num1, &num2);
```



Διεύθυνση μεταβλητής

- Κάθε μεταβλητή καταλαμβάνει ένα ή περισσότερα byte στη μνήμη
  - **char** (1B), **short** (2B), **int** (4B), **long** (8B), **float** (4B), **double** (8B), κ.τ.λ.
  - Η διεύθυνση του πρώτου byte είναι η **διεύθυνση της μεταβλητής**

Computer		Programmers		
Address	Content	Name	Type	Value
90000000	00			
90000001	00	sum	int (4 bytes)	000000FF(255_{10})
90000002	00			
90000003	FF			
90000004	FF	age	short (2 bytes)	FFFF(-1_{10})
90000005	FF			
90000006	1F	averge	double (8 bytes)	1FFFFFFFFFFFFFFF ($4.45015E-308_{10}$)
90000007	FF			
90000008	FF			
90000009	FF			
9000000A	FF			
9000000B	FF			
9000000C	FF			
9000000D	FF			
9000000E	90	ptrSum	int* (4 bytes)	90000000
9000000F	00			
90000010	00			
90000011	00			

Note: All numbers in hexadecimal

Δείκτες

- Η C δίνει τη δυνατότητα να αποθηκεύουμε **διευθύνσεις μνήμης** σε ειδικές μεταβλητές που λέγονται **pointers**

```
int *i;      // pointer to int
char *c;     // pointer to char
double *d;   // pointer to double
```

- Ο κάθε pointer είναι associated με κάποιον τύπο δεδομένων.
  - Π.χ., το **i** στο παραπάνω παράδειγμα δείχνει σε int

- Ορίζονται με ένα αστεράκι πριν το όνομα της μεταβλητής
  - Π.χ., στο παρακάτω, μόνο το **p** είναι pointer

```
int i, a[100], *p;
```

Τελεστές Δεικτών

- Τελεστής διεύθυνσης (address operator): **&**
  - Επιστρέφει τη διεύθυνση μιας μεταβλητής

```
int i, *p;
p = &i;   // p = address of i
```

- Τελεστής αποαναφοράς (dereference operator): *
  - Μας επιτρέπει να προσπελάσουμε το περιεχόμενο της μνήμης που δείχνει ένας pointer

```
// συνέχεια από το προηγούμενο
*p = 42;
printf("%d\n", i);  // 42
i /= 2;
printf("%d\n", *p);  // 21
```

Τελεστές Δεικτών

- Τελεστής ανάθεσης: **=**

```
int i=5, *p, *q;
p = &i;   // p = address of i
q = p;
printf("%d\n", *q);  // 5
```

- Προσοχή! Άλλο η ανάθεση δείκτη, κι άλλο η ανάθεση της τιμής που δείχνει ο δείκτης

```
q = p;   // Ανάθεση δείκτη
*q = *p; // Ανάθεση τιμής
```

Παράδειγμα

```
int x=1;
int y=2;
int *p;
p = &x;  /* η p δείχνει τη διεύθυνση της x   */
y = *p;  /* η y γίνεται 1, δηλαδή η τιμή της x   */
*p = 0;  /* η x γίνεται 0 */
*p = *p + 10;  /* x=x+10 */

*p += 1; /* x=x+1 */
++*p;    /* x=x+1 */
(*p)++;  /* x=x+1 */
```

Παράδειγμα

```
#include <stdio.h>
main()
{
      int x=5; int y=10; int *ptr; int **ptr2;
      ptr=&x;
      ptr2=&ptr;
      *ptr2=&y;
      printf("%d", *ptr);
}
```

Τελεστές Δεικτών

- Οι τελεστές **&** και ***** είναι «αντίστροφοι», συνεπώς εν γένει αλληλοαναιρούμενοι

```
int i, j, *p=NULL, *q=&i;
i = *&j;   // i=j
i = &*j;   // compile error! (j not ptr)
p = &*q;   // p=q
```

- Γενικά είναι καλό να αρχικοποιείτε πάντα τους pointers (π.χ., με NULL), αλλιώς μπορεί να «δείχνουν» σε αυθαίρετη διεύθυνση με ολέθρια αποτελέσματα!
  - Τι είναι το NULL???
  - Είναι ειδική τιμή (συγκεκριμένα η τιμή 0), που σημαίνει τον **κενό δείκτη**, δηλαδή έναν pointer που δεν δείχνει πουθενά
  - Ο NULL pointer δεν αποδεικτοδοτείται. Π.χ., το p=NULL; *p = 5; θα δημιουργήσει λάθος, δεν θα βάλει το 5 στη διεύθυνση 0.

NULL

- Τι είναι το **NULL**???
  - Είναι ειδική τιμή (συγκεκριμένα η τιμή 0), που σημαίνει τον **κενό δείκτη**, δηλαδή έναν pointer που δεν δείχνει πουθενά
  - Ο NULL pointer δεν αποδεικτοδοτείται
  - Π.χ., ο παρακάτω κώδικας θα δημιουργήσει λάθος, δεν θα βάλει το 5 στη διεύθυνση 0

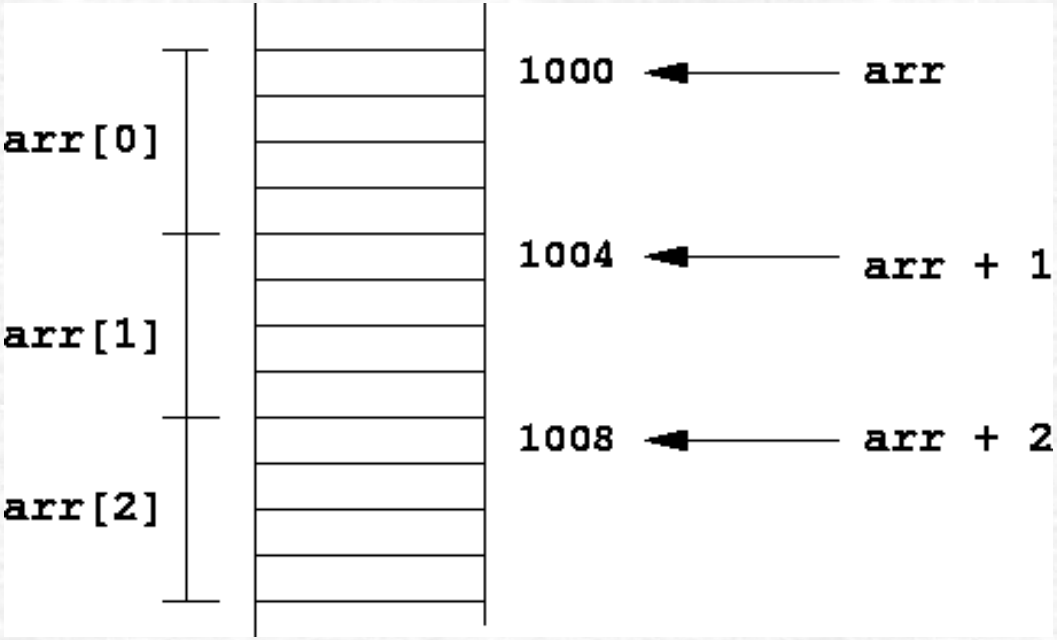
```
int *p = NULL;
*p = 5;
```

- Και τι είναι το **void *****???
  - Είναι ο τύπος δεδομένων γενικού δείκτη (δείκτη σε κενό)
  - Δηλαδή δείκτης που δεν είναι associated με συγκεκριμένο τύπο (int, char, κ.τ.λ.)
  - Κανονικός δείκτης, απλά δεν μπορεί να κάνει **pointer arithmetic**
  - Μα, τι είναι pointer arithmetic??? ☺

Pointer Arithmetic (Αριθμητική Δεικτών)

- Ας θεωρήσουμε ότι έχουμε ένα array ακεραίων **int arr[10]**
- Γνωρίζουμε πως **arr[i]** είναι το i-οστό στοιχείο του array
- Αυτό που δεν γνωρίζουμε είναι πως το **arr[i]** ορίζεται ως ***(arr + i)**

```
arr και &arr[0] συνώνυμα
  arr[0] == *arr
arr[i] == *(arr + i)
```

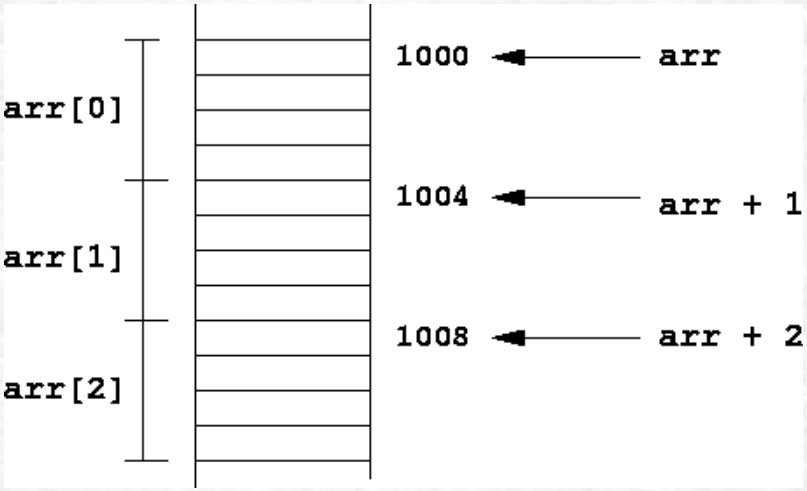


Pointer Arithmetic (Αριθμητική Δεικτών)

Το κατά πόσο αυξάνεται/μειώνεται ένας pointer σε πράξεις με ακεραίους, εξαρτάται από τον τύπο του.

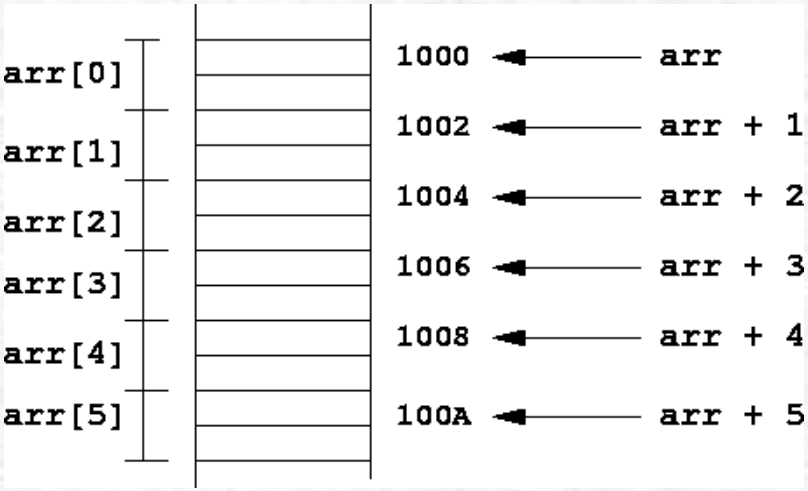
int arr[10];

sizeof(int) == 4



short arr[10];

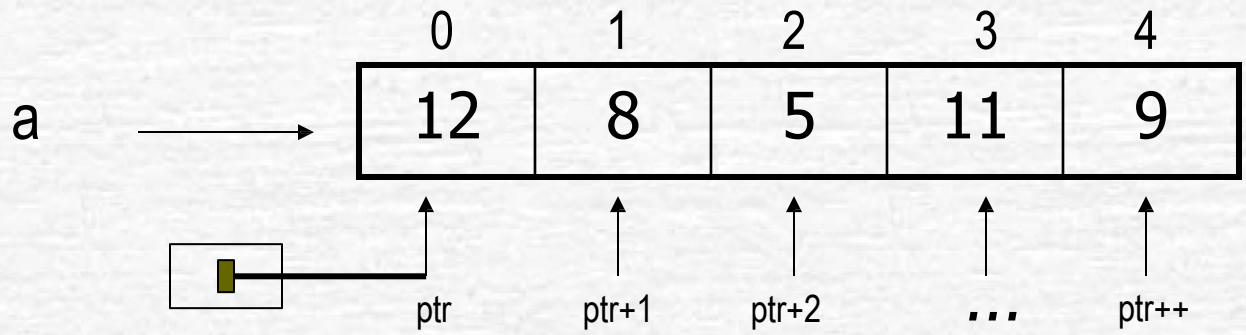
sizeof(short) == 2



Pointers και Arrays

- Μπορούμε να δώσουμε σε έναν pointer την διεύθυνση ενός array, και στη συνέχεια να τον χρησιμοποιήσουμε ως array

```
int a[] = {12, 8, 5, 11, 9}, *ptr;
ptr=a;
printf("%d\n", ptr[1]);    // 8
ptr += 2;
printf("%d\n", ptr-a);     // 2
printf("%d\n", ptr[1]);    // 11
ptr--;
printf("%d\n", ptr[1]);    // 5
a++;  // Compile error, arr is const!
```



Pointer Arithmetic

- Έγκυρες πράξεις με δείκτες είναι:
  - απόδοση τιμής με δείκτη ίδιου τύπου
  - η πρόσθεση ή αφαίρεση δείκτη και ακεραίου
  - η αφαίρεση ή σύγκριση δύο δεικτών για μέλη ίδιου πίνακα
  - η αντικατάσταση ή σύγκριση με NULL

- Δεν είναι έγκυρες πράξεις οι:
  - πρόσθεση δύο δεικτών
  - πολλαπλασιασμός, διαίρεση, ολίσθηση, η πρόσθεση float, double σε δείκτες.

Πέρασμα Μεταβλητών

```
#include <stdio.h>
void swap(int x, int y);

int main()
{
        int x=2, y=3;
        printf("x:%d, y:%d \n", x,y);
        swap(x,y);
        printf("x:%d, y:%d \n", x,y);
}

void swap(int x, int y)  /* λάθος */
{
        int temp;
        temp=x;
        x=y;
        y=temp;
}
```

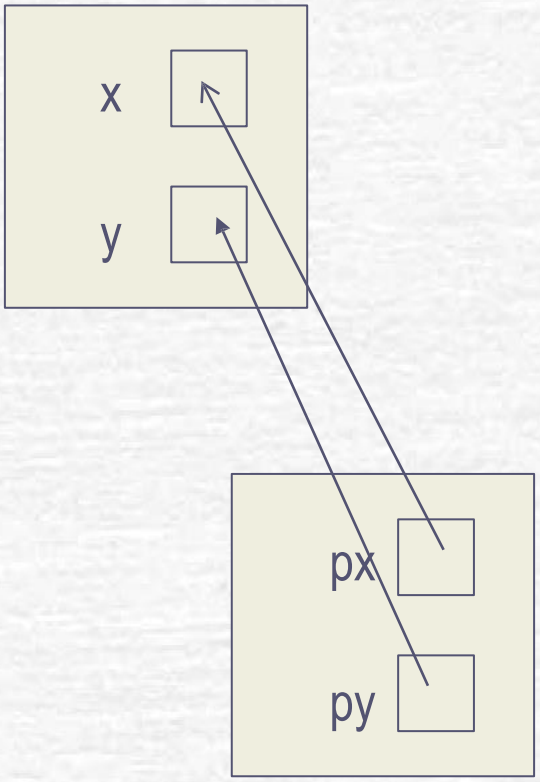
Πέρασμα Μεταβλητών (2)

```
#include <stdio.h>
void swap(int x, int y);

int main()
{
        int x=2, y=3;
        printf("x:%d, y:%d \n", x,y);
        swap(&x,&y);
        printf("x:%d, y:%d \n", x,y);
}

void swap(int *px, int *py)   /*σωστή υλοποίηση
*/
{
        int temp;
        temp=*px;
        *px=*py;
        *py=temp;
}
```

Πέρασμα Μεταβλητών (3)



Δυναμική Διαχείριση Μνήμης

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

- ***Calloc, Malloc***,
  για δέσμευση μνήμης
- ***Realloc***
  για αλλαγή μεγέθους δεσμευμένης μνήμης
- ***Free***
  για απελευθέρωση μνήμης