ΚΛΑΣΕΙΣ ΚΑΙ ΕΜΒΕΛΕΙΑ ΜΕΤΑΒΛΗΤΩΝ

(Storage Classes)

Όλες οι μεταβλητές που ορίζονται εντός μιας συνάρτησης (της main() συμπεριλαμβανομένης) είναι τοπικές και αντιστοιχούν στην κλάση **auto** της C.
Η κλάση **auto** είναι η **default κλάση** και δεν χρειάζεται να τη δηλώνουμε κατά τη διακήρυξη των μεταβλητών. Αν για κάποιο λόγο θα θέλαμε να υπενθυμίσουμε ότι μια μεταβλητή είναι σκοπίμως **αυτόματη** τότε μπορούμε να είμαστε αναλυτικοί κατά τη δήλωση των μεταβλητών και αντί, για παράδειγμα, της διακήρυξης

```
int sum;
float a=10.3, b=-13.9;
```

θα μπορούσαμε να έχουμε την ισοδύναμη διακήρυξη

```
auto int sum;
auto float a=10.3, b=-13.9;
```

Οι μεταβλητές κλάσης auto δημιουργούνται κάθε φορά που καλείται η συνάρτηση και καταργούνται με το τέλος της συνάρτηση. Συνεπώς, η εμβέλειά τους είναι στο αυστηρό πλαίσιο της συνάρτησης. Αντίθετα, μια μεταβλητή **στατικής κλάσης** (**static class**) δεν "χάνει" την τιμή της κάθε φορά που εκτελείται η συνάρτηση. Οι στατικές μεταβλητές έχουν την ίδια εμβέλεια με τις αυτόματες μεταβλητές.

Παράδειγμα

```
#include <stdio.h>

void auto_static();

int main()
{
  int i;

  for(i=0; i<3; i++)
    auto_static();
  return 0;
}

void auto_static() /* Αυτόματες/στατικές μεταβλητές */
{
  int auto_var = 1;
  static int static_var = 1;

  printf("auto=%d, static=%d\n",auto_var++,static_var++);
}
```

Έξοδος προγράμματος:
auto=1, static=1
auto=1, static=2
auto=1, static=3

ΕΞΩΤΕΡΙΚΕΣ – ΚΑΘΟΛΙΚΕΣ ΜΕΤΑΒΛΗΤΕΣ [1/2]

Οι μεταβλητές που ορίζονται έξω από συναρτήσεις είναι **καθολικής εμβέλειας** και ονομάζονται **εξωτερικές** (**external**) ή καθολικές μεταβλητές. Μια καθολική μεταβλητή πρέπει να δηλώνεται μέσα στη συνάρτηση που τη χρησιμοποιεί με τη λέξη-κλειδί **extern**.

Παράδειγμα:

```
int  a,b;      /* Δήλωση 4 καθολικών μεταβλητών */
char gl_ch;
float fl;

int main()
{
  extern int a; /* Δήλωση ότι η a είναι καθολική */
  auto int b;   /* Σκόπιμη δήλωση νέας τοπικής μεταβλητής που
                   έχει το ίδιο όνομα με μια καθολική μεταβλητή */
  extern char gl_ch;
  ...
}
void func1()
{
  extern int a;   /* Δήλωση ότι η a είναι καθολική */
  int  i,j;       /* Δήλωση νέων αυτόματων τοπικών μεταβλητών */
  extern float fl;
  ...
}
```

ΕΞΩΤΕΡΙΚΕΣ – ΚΑΘΟΛΙΚΕΣ ΜΕΤΑΒΛΗΤΕΣ [2/2]

Οι μεταβλητές **a** και **gl_ch** έχουν δηλωθεί ως καθολικές μέσα στην m**ain()**.

Όταν οι δηλώσεις των καθολικών μεταβλητών είναι στο ίδιο αρχείο και προηγούνται κάποιας συνάρτησης, τότε δεν είναι απαραίτητο να δηλωθούν ως καθολικές και μέσα στη συνάρτηση. Έτσι, η **fl** μπορεί επίσης να χρησιμοποιηθεί από τη **main()** ως καθολική μεταβλητή.

Όμως, επειδή η **b** έχει δηλωθεί μέσα στη **main()** ως auto μεταβλητή, η μεταβλητή αυτή θα είναι διαφορετική από την εξωτερική.

Η **main()** δεν θα έχει δυνατότητα προσπέλασης της καθολικής μεταβλητής b.

Αντίστοιχα, η **func1()** μπορεί να προσπελάσει όλες τις καθολικές μεταβλητές (δηλωμένες και αδήλωτες).

```
int  a,b;      /* Δήλωση 4 καθολικών μεταβλητών */
char gl_ch;
float fl;

int main()
{
  extern int a; /* Δήλωση ότι η a είναι καθολική */
  auto int b;   /* Σκόπιμη δήλωση νέας τοπικής μεταβλητής που
                   έχει το ίδιο όνομα με μια καθολική μεταβλητή */
  extern char gl_ch;
  ...
}
void func1()
{
  extern int a;   /* Δήλωση ότι η a είναι καθολική */
  int  i,j;       /* Δήλωση νέων αυτόματων τοπικών μεταβλητών */
  extern float fl;
  ...
}
```

ΔΕΙΚΤΕΣ (Pointers) [1/4]

- Θα ονομάζουμε **θέση μνήμης**, την περιοχή στη μνήμη του υπολογιστή που χρησιμοποιείται για την αποθήκευση κάποιας τιμής. Η τιμή αυτή θα αποτελεί το **περιεχόμενο** της θέσης μνήμης.
- Μια διακήρυξη μεταβλητής υποχρεώνει τον υπολογιστή να **δεσμεύσει** (ή να *παραχωρήσει*) ένα μικρό τμήμα της μνήμης του, δηλαδή μια θέση μνήμης, για την αποθήκευση της τιμής αυτής της μεταβλητής (το μέγεθος της θέσης μνήμης καθορίζεται από τον τύπο της μεταβλητής). Για παράδειγμα, αν η μεταβλητή είναι τύπου int, θα δεσμευθούν 4 bytes για την αποθήκευση κάποιου ακεραίου, με τη θέση μνήμης να προσδιορίζεται από: α) τη **διεύθυνση** του πρώτου byte και β) το **μέγεθος** (σε bytes) των δεδομένων τύπου int.
- Η πρόσβαση στο περιεχόμενο μιας μεταβλητής (για διάβασμα της τιμής ή για αποθήκευση κάποιας τιμής) γίνεται με άμεσο τρόπο και ονομάζεται **άμεση προσπέλαση μνήμης**.

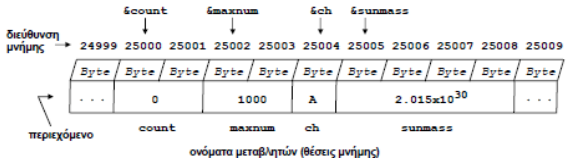
ΔΕΙΚΤΕΣ (Pointers) [2/4]

- Στην C, η διεύθυνση [*του πρώτου byte*] της θέσης μνήμης μιας μεταβλητής προσδιορίζεται με χρήση του μοναδιαίου τελεστή διεύθυνσης '**&**' πριν από το όνομα της μεταβλητής.
- Παράδειγμα: έστω οι διακηρύξεις

  short count = 0, maxnum = 1000;

  char ch = 'A';

  float sunmass = 2.015e30;
- και έστω ότι το υπολογιστικό σύστημα είναι byte-addressable (μπορεί να δεικτοδοτήσει οποιοδήποτε byte της μνήμης).
- Οι διευθύνσεις των μεταβλητών (&count, &maxnum, &ch, &sunmass) καθώς και το περιεχόμενό τους στη μνήμη του υπολογιστή δείχνονται στο παρακάτω σχήμα:



ΔΕΙΚΤΕΣ (Pointers) [3/4]

- Οι διευθύνσεις μνήμης είναι μη προσημασμένοι ακέραιοι. Συνεπώς, μπορούμε να εμφανίσουμε την τιμή και τη διεύθυνση της sunmass ως εξής:

```
printf("sunmass = %g, &sunmass = %u\n",sunmass,&sunmass);
```

Έξοδος: sunmass = 2.015e+030, &sunmass = 25005

- Ο **δείκτης** (**pointer**) είναι μια συμβολική αναπαράσταση μιας διεύθυνσης. Για παράδειγμα, η &sunmass δεν είναι παρά ένας δείκτης στη μεταβλητή sunmass.
- Στην C μπορούμε να δηλώσουμε και **μεταβλητές δείκτη** (pointer variables) ή απλά **δείκτες**, **στους οποίους θα αποθηκεύονται διευθύνσεις και όχι δεδομένα ή τιμές**. Όπως ένας ακέραιος είναι η τιμή μιας μεταβλητής τύπου int, έτσι και μια διεύθυνση είναι η τιμή ενός δείκτη.

ΔΕΙΚΤΕΣ (Pointers) [4/4]

- Έστω ότι η μεταβλητή ptr είναι ένας δείκτης. Τότε, μπορούμε να έχουμε δηλώσεις της μορφής:

```
ptr = &count; /* εκχωρεί τη διεύθυνση της count στον ptr */
```

- Θα λέμε ότι ο **ptr** "δείχνει" στην **count**. Η διαφορά μεταξύ του **ptr** και της **&count** είναι ότι ο **ptr** είναι μια μεταβλητή ενώ η **&count** είναι μια σταθερά (η διεύθυνση της count δεν πρόκειται να αλλάξει ενώ εκτελείται το πρόγραμμα).
- Αφού ο **ptr** είναι μεταβλητή, μπορούμε να τον κάνουμε να δείχνει σε κάποια άλλη διεύθυνση, π.χ.:

```
ptr = &maxnum; /* ο ptr τώρα δείχνει στην maxnum */
```

Ο τελεστής έμμεσης αναφοράς: *

- Έστω ότι γνωρίζουμε ότι ο ptr δείχνει στην μεταβλητή maxnum. Μπορούμε να χρησιμοποιήσουμε τον μοναδιαίο τελεστή * **π**ροκειμένου να προσπελάσουμε το περιεχόμενο της maxnum.
- Για παράδειγμα, ο κλασικός τρόπος εκχώρησης της τιμής της maxnum σε μια άλλη μεταβλητή val είναι με την δήλωση:

```
val = maxnum; /* άμεση προσπέλαση μνήμης */
```

- Με χρήση του **ptr** και του τελεστή * ο τρόπος προσπέλασης θα είναι έμμεσος:

```
val = *ptr; /* έμμεση προσπέλαση μνήμης */
```

- Θα λέμε ότι στη μεταβλητή val εκχωρείται το περιεχόμενο της θέσης μνήμης στην οποία δείχνει ο ptr.

Δήλωση ενός δείκτη

- Η δήλωση ενός δείκτη γίνεται όπως και για τις μεταβλητές με τη διαφορά ότι πρέπει να προτίθεται ο τελεστής * του ονόματος της μεταβλητής. Για παράδειγμα:

```
int i, j, *ptr1, *ptr2;
char ch, *ptrc;
float a, b, *ptrf;
```

- Ο καθορισμός του τύπου των δεδομένων μας λέει σε τι τύπου μεταβλητή δείχνει ο δείκτης ενώ ο αστερίσκος προσδιορίζει τη μεταβλητή (π.χ. **ptr1**) ως δείκτη. Η δήλωση **char *ptrc;** μας λέει ότι η μεταβλητή **ptrc** είναι ένας δείκτης και ότι το περιεχόμενο ***ptrc** είναι τύπου **char**. Θα λέμε ότι η μεταβλητή **ptrc** είναι ένας "**δείκτης σε χαρακτήρα**" ή "pointer σε char" (δηλαδή, δείκτης σε θέση μνήμης δεσμευμένης για αποθήκευση χαρακτήρα).

Παράδειγμα

```
int a, b, c, *ptr, *m; /* Δήλωση 5 μεταβλητών (2 δείκτες) */
a = 25;
ptr = &a;
m = ptr;
b = *m;
c = a + b;
```

Παράδειγμα

```
int a, b, c, *ptr, *m; /* Δήλωση 5 μεταβλητών (2 δείκτες) */
a = 25;
ptr = &a;
m = ptr;
b = *m;
c = a + b;
```

Διευθύνση Μνήμης,	Όνομα Μεταβλητής	Περιεχόμενο μεταβλητών
1000	a	
2000	ptr	
3000	m	
4000	b	
5000	c	

Παράδειγμα

```
int a, b, c, *ptr, *m; /* Δήλωση 5 μεταβλητών (2 δείκτες) */
a = 25;
ptr = &a;
m = ptr;
b = *m;
c = a + b;
```

Διευθύνση Μνήμης,	Όνομα Μεταβλητής	Περιεχόμενο μεταβλητών
1000	a	25
2000	ptr	
3000	m	
4000	b	
5000	c	

Παράδειγμα

```
int a, b, c, *ptr, *m; /* Δήλωση 5 μεταβλητών (2 δείκτες) */
a = 25;
ptr = &a;
m = ptr;
b = *m;
c = a + b;
```

Διευθύνση Μνήμης,	Όνομα Μεταβλητής	Περιεχόμενο μεταβλητών
1000	a	25
2000	ptr	1000
3000	m	
4000	b	
5000	c	

Παράδειγμα

```
int a, b, c, *ptr, *m; /* Δήλωση 5 μεταβλητών (2 δείκτες) */
a = 25;
ptr = &a;
m = ptr;
b = *m;
c = a + b;
```

Διευθύνση Μνήμης,	Όνομα Μεταβλητής	Περιεχόμενο μεταβλητών
1000	a	25
2000	ptr	1000
3000	m	1000
4000	b	
5000	c	

Παράδειγμα

```
int a, b, c, *ptr, *m; /* Δήλωση 5 μεταβλητών (2 δείκτες) */
a = 25;
ptr = &a;
m = ptr;
b = *m;
c = a + b;
```

Διευθύνση Μνήμης,	Όνομα Μεταβλητής	Περιεχόμενο μεταβλητών
1000	a	25
2000	ptr	1000
3000	m	1000
4000	b	25
5000	c	

Παράδειγμα

int a, b, c, *ptr, *m; /* Δήλωση 5 μεταβλητών (2 δείκτες) */

a = 25;

ptr = &a;

m = ptr;

b = *m;

c = a + b;

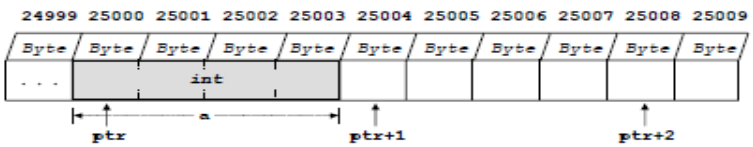
Διευθύνση Μνήμης,	Όνομα Μεταβλητής	Περιεχόμενο μεταβλητών
1000	a	25
2000	ptr	1000
3000	m	1000
4000	b	25
5000	c	50

Αριθμητική Δεικτών [1/3]

pointer arithmetic

Έστω το παρακάτω τμήμα προγράμματος:

```
int a, *ptr;
ptr = &a; /* ptr == 25000 */
ptr = ptr + 1; /* ptr == 25004 */
++ptr; /* ptr == 25008 */
ptr = ptr + 3; /* ptr == 25020 */
```



Πρώτα, η τιμή του ptr γίνεται 25000 (δηλαδή, όσο η διεύθυνση της μεταβλητής a). Μετά, η τιμή του ptr αυξάνεται κατά τόσα bytes όσα αντιστοιχούν στον τύπο δεδομένων της θέσης μνήμης στην οποία "δείχνει" (επειδή ο ptr δείχνει σε ακέραιο τύπο θα αυξηθεί κατά 4 bytes). Τέλος, η τιμή του ptr θα αυξηθεί κατά 3 x 4 = 12 bytes.

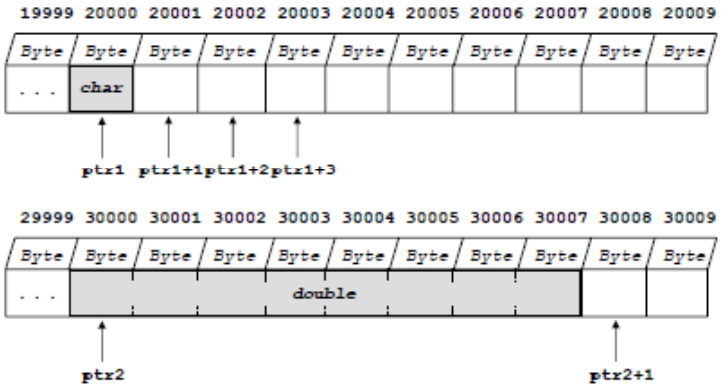
Αριθμητική Δεικτών [2/3]

pointer arithmetic

Αριθμητική με δείκτες σε char και double:

```
char ch, *ptr1;
ptr1 = &ch;
++ptr1;
```

```
double d, *ptr2;
ptr2 = &d;
++ptr2;
```



Αριθμητική Δεικτών [3/3]

pointer arithmetic

Τι κάνει το παρακάτω πρόγραμμα?

```
int a, *p;
p = &a;
a = 15;
*p = 10;
(*p)++;
p++;
*p++ = 50;
```

Αριθμητική Δεικτών [3/3]

pointer arithmetic

Τι κάνει το παρακάτω πρόγραμμα?

```
int a, *p;
p = &a;
a = 15;
*p = 10;
(*p)++;
p++;
*p++ = 50;
```

Διευθύνση Μνήμης,	Όνομα Μεταβλητής	Περιεχόμενο μεταβλητών
2000	a	
3000	p	

Έστω ότι η διεύθυνση της μεταβλητής a είναι 2000.

Αριθμητική Δεικτών [3/3]

pointer arithmetic

Τι κάνει το παρακάτω πρόγραμμα?

```
int a, *p;
p = &a;
a = 15;
*p = 10;
(*p)++;
p++;
*p++ = 50;
```

Διευθύνση Μνήμης,	Όνομα Μεταβλητής	Περιεχόμενο μεταβλητών
2000	a	
3000	p	2000

Ο δείκτης p αρχικοποιείται στη διεύθυνση της a (p == 2000).

Αριθμητική Δεικτών [3/3]

pointer arithmetic

Τι κάνει το παρακάτω πρόγραμμα?

```
int a, *p;
p = &a;
a = 15;
*p = 10;
(*p)++;
p++;
*p++ = 50;
```

Διευθύνση Μνήμης,	Όνομα Μεταβλητής	Περιεχόμενο μεταβλητών
2000	a	15
3000	p	2000

Εκχώρηση τιμής στην a.

Αριθμητική Δεικτών [3/3]

pointer arithmetic

Τι κάνει το παρακάτω πρόγραμμα?

```
int a, *p;
p = &a;
a = 15;
*p = 10;
(*p)++;
p++;
*p++ = 50;
```

Διευθύνση Μνήμης,	Όνομα Μεταβλητής	Περιεχόμενο μεταβλητών
2000	a	10
3000	p	2000

Εκ νέου εκχώρηση τιμής στην a (επειδή ο p δείχνει στην a).

Αριθμητική Δεικτών [3/3]

pointer arithmetic

Τι κάνει το παρακάτω πρόγραμμα?

```
int a, *p;
p = &a;
a = 15;
*p = 10;
(*p)++;
p++;
*p++ = 50;
```

Διευθύνση Μνήμης,	Όνομα Μεταβλητής	Περιεχόμενο μεταβλητών
2000	a	11
3000	p	2000

Πρώτα υπολογίζεται η παράσταση μέσα στις παρενθέσεις, δηλαδή το περιεχόμενο της θέσης μνήμης στην οποία δείχνει ο p (*p == a == 10) και στη συνέχεια το αυξάνει κατά ένα (η εντολή αυτή αναλυτικά μπορεί να γραφεί ως *p = *p + 1; και είναι ισοδύναμη με την a++;)

Αριθμητική Δεικτών [3/3]

pointer arithmetic

Τι κάνει το παρακάτω πρόγραμμα?

```
int a, *p;
p = &a;
a = 15;
*p = 10;
(*p)++;
p++;
*p++ = 50;
```

Διευθύνση Μνήμης,	Όνομα Μεταβλητής	Περιεχόμενο μεταβλητών
2000	a	11
3000	p	2004

Αυξάνεται η τιμή της p κατά 4 (επειδή δείχνει σε θέσεις μνήμης τύπου int).

Αριθμητική Δεικτών [3/3]

pointer arithmetic

Τι κάνει το παρακάτω πρόγραμμα?

```
int a, *p;
p = &a;
a = 15;
*p = 10;
(*p)++;
p++;
*p++ = 50;
```

Διευθύνση Μνήμης,	Όνομα Μεταβλητής	Περιεχόμενο μεταβλητών
2000	a	11
2008		50
3000	p	2008

Τέλος, πρώτα αυξάνεται η τιμή της p κατά 4 (δηλαδή, p == 2008) και μετά αποθηκεύεται στη θέση μνήμης με διεύθυνση 2008 ο ακέραιος 50 (ότι δεδομένα είχαμε εκεί θα χαθούν!). Είναι ισοδύναμη με *(p++) = 50; καθώς ο ++ έχει μεγαλύτερη προτεραιότητα από τον *.

Δείκτες τύπου void

Δήλωση δείκτη τύπου void:

```
void *ptr;
```

Ο δείκτης αυτός δεν δείχνει σε δεδομένα συγκεκριμένου τύπου και κατά συνέπεια δεν μπορούμε να εφαρμόσουμε αριθμητικούς τελεστές σε αυτόν. Εκείνο βέβαια που μπορούμε να κάνουμε σε δείκτες τύπου void είναι να τους αναθέτουμε τιμές (δηλαδή διευθύνσεις) όπως ακριβώς και με τους άλλους δείκτες. Για παράδειγμα, η παρακάτω εκχώρηση τιμής στον p είναι καθόλα νόμιμη

```
int a;
void *p;
p = &a;
```

όχι όμως και παραστάσεις όπως οι: **p++; *p++; (*p)++;**

Δείκτης τύπου void είναι η τιμή που επιστρέφεται από αρκετές συναρτήσεις βιβλιοθήκης. Ο δείκτης αυτός μπορεί στη συνέχεια να μετατραπεί σε κανονικό δείκτη μέσω της τεχνικής μετατροπής τύπου (type casting).

Εμφάνιση διευθύνσεων μνήμης

- Καθώς οι διευθύνσεις είναι μη προσημασμένοι ακέραιοι, είδαμε ότι για την εμφάνιση κάποιας διεύθυνσης στο δεκαδικό σύστημα μπορεί να χρησιμοποιηθεί το προσδιοριστικό μετατροπής %u. Είναι όμως καλύτερη πρακτική να χρησιμοποιείται το προσδιοριστικό μετατροπής %p με το αποτέλεσμα να εμφανίζεται σε δεκαεξαδική μορφή. Παράδειγμα:

```
#include <stdio.h>
int main()
{
  int a, *pa;
  char b, *pb;
  pa = &a;  pb = &b;

  printf("&a=%u, &a=%p\n",&a,&a);
  printf("pa=%p, pa++=%p\n",pa,pa++);
  printf("pb=%p, pb++=%p\n",pb,pb++);

  pa = NULL; /* … από stdio.h */
  printf("pa=%p\n",pa);
  return 0;
}
```

```
Εξοδος
&a=2293620, &a=0022FF74
pa=0022FF74, pa++=0022FF78
pb=0022FF67, pb++=0022FF68
pa=00000000
```

Δείκτες και συναρτήσεις [1/4]

Έστω ότι θέλουμε μια συνάρτηση που να ανταλλάσσει τις τιμές δύο μεταβλητών. Ο κλασικός τρόπος, με μόνο τοπικές μεταβλητές, δεν δουλεύει. Π.χ.:

```
int main()
{
  int x = 5, y = 10;
  printf("Prin: x = %d, y = %d\n",x,y);
  switch1(x,y);
  printf("Meta: x = %d, y = %d\n",x,y);
  return 0;
}

void switch1(int u, int v)
{
  int temp;
  temp = u; u = v; v = temp;
}
```

```
Εξοδος προγράμματος:

Prin: x = 5, y = 10
Meta: x = 5, y = 10
```

Δείκτες και συναρτήσεις [2/4]

Μια προσπάθεια θα ήταν να χρησιμοποιήσουμε καθολικές μεταβλητές αλλά σε αυτήν την περίπτωση δημιουργούμε μια συνάρτηση χωρίς ορίσματα ενώ θα ήταν καλύτερο να μπορούμε να ανταλλάσσουμε τις τιμές δύο τοπικών μεταβλητών της καλούσας συνάρτησης.

```
int x = 5, y = 10;
int main()
{
  printf("Prin: x = %d, y = %d\n",x,y);
  switch2();
  printf("Meta: x = %d, y = %d\n",x,y);
  return 0;
}
void switch2()
{
  int temp;
  temp = x; x = y; y = temp;
}
```

```
Εξοδος προγράμματος:

Prin: x = 5, y = 10
Meta: x = 10, y = 5
```

Δείκτες και συναρτήσεις [3/4]

Η λύση απαιτεί την χρήση δεικτών. Αντί να περνάμε τις τιμές των μεταβλητών x και y θα περνάμε τις διευθύνσεις αυτών των μεταβλητών:

```
int main()
{
  int x = 5, y = 10;
  printf("Prin: x = %d, y = %d\n",x,y);
  switch3(&x,&y);
  printf("Meta: x = %d, y = %d\n",x,y);
  return 0;
}
void switch3(int *u, int *v)
{
  int temp;
  temp = *u; *u = *v; *v = temp;
}
```

```
Εξοδος προγράμματος:

Prin: x = 5, y = 10
Meta: x = 10, y = 5
```

Δείκτες και συναρτήσεις [4/4]

Με την κλήση της switch3(), δημιουργούνται δύο νέοι δείκτες, οι u και v, στους οποίους αντιγράφονται οι διευθύνσεις των x και y (δηλαδή, u = &x και v = &y).

Στη συνέχεια, με χρήση του τελεστή έμμεσης αναφοράς έχουμε πρόσβαση στο περιεχόμενο των x και y.