Δείκτες

Δείκτες (pointers)

- Πίνακες ως παράμετροι συναρτήσεων:

Περνά τη διεύθυνση του πίνακα

```
main () {
    int nums[5]={1,2,3,4,5};
    clear(nums, 5);

}
```

```
void clear(int ar[ ], int size) {
    int i;
    for (i=0; i < size; i++)
        ar[i]=0;
}
```

Αλλάζει τα στοιχεία του nums

Δείκτες

- Μπορούμε να αλλάξουμε τις τιμές που είναι αποθηκευμένες στον πίνακα **nums** μέσα από τη συνάρτηση *clear()*.
- Αυτό συμβαίνει γιατί το όνομα **nums** αναφέρεται στη **διεύθυνση μνήμης**, από την οποία ξεκινά ο πίνακας.

```
main () {
    int x=3; y=5;
    swap(x,y);
    printf("x=%d, y=%d", x, y);

}
```

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

Τυπώνει x=3, y=5. Δε γίνεται η ανταλλαγή (swapping)!

Δείκτες

- Το πρόβλημα συμβαίνει επειδή απλώς αντιγράφονται οι τιμές των x και y στα ορίσματα a και b.

- Άρα, τι θα γινόταν εάν μπορούσαμε με κάποιον τρόπο να ανταλλάξουμε τις διευθύνσεις των x και y;
  - Τότε, θα μπορούσαμε να αλλάξουμε τα x και y μέσα στη συνάρτηση *swap()*, όπως συνέβη με τον πίνακα.

Δήλωση δείκτη

- **Δείκτης:** μία μεταβλητή που κρατά **μία διεύθυνση**.

Διεύθυν.	Περιεχόμ.
900 :	x , 32
904 :	
908 :	px, 900

Κανονική μεταβλητή.

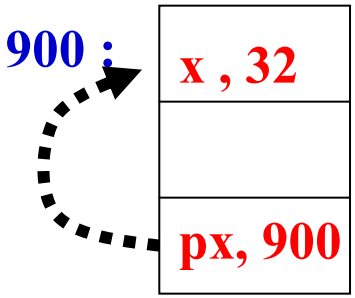
Όνομα: x, Τιμή: 32, Τύπος: int

Μεταβλητή δείκτη.

Όνομα: px, Τιμή: 900, Τύπος: δείκτης σε int

Λέμε ότι ο px δείχνει στην x

Φανταστείτε ένα τόξο από τη μεταβλητή δείκτη στην κανονική μεταβλητή, το οποίο δείχνει πού «δείχνει» ο δείκτης.



Δήλωση δείκτη

- Ο δείκτης πρέπει να δηλωθεί:

base_type * pointer_name ;

ο τύπος της μεταβλητής αποθηκεύεται στη θέση που δείχνει ο δείκτης

το όνομα της μεταβλητής δείκτη. Καλή προγραμματιστική πρακτικά: το πρώτο γράμμα του ονόματος να είναι πάντοτε p

προσδιορίζει ότι δηλώνεται μία μεταβλητή δείκτη

Δήλωση δείκτη

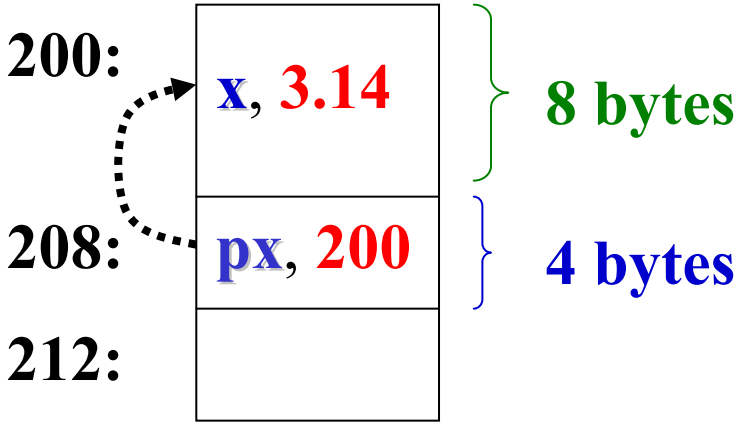
Γιατί πρέπει να δηλωθεί ο τύπος της κανονικής μεταβλητής;

- **Γιατί όταν δηλώνεται μία (κανονική) μεταβλητή, δεσμεύεται συγκεκριμένη μνήμη, π.χ. 8 bytes για double, 4 bytes για int.**
- **Ο δείκτης αναφέρεται σε μία διεύθυνση, στην οποία αποθηκεύεται η τιμή μίας κανονικής μεταβλητής.**
- **Ο δείκτης χρησιμοποιείται για να γίνεται έμμεση αναφορά σ' αυτήν την τιμή. Έτσι, πρέπει να γνωρίζουμε πόση ακριβώς μνήμη καταλαμβάνει αυτή η τιμή.**

Δήλωση δείκτη

Πόση μνήμη καταλαμβάνει η ίδια η μεταβλητή δείκτη;

- **Η διεύθυνση είναι ένας ακέραιος, έτσι ο δείκτης καταλαμβάνει 4 bytes, <u>ανεξάρτητα από τον τύπο της κανονικής μεταβλητής που δείχνει.</u>**



Η x είναι μεταβλητή τύπου double

Ο px είναι δείκτης στην x

<u>Αναλογία με την πραγματικότητα</u> : μία οκταμελής οικογένεια δε χρειάζεται μεγαλύτερη ταχυδρομική διεύθυνση από μία τετραμελή (ίσως μεγαλύτερο γραμματοκιβώτιο!)

Δήλωση δείκτη

Γιατί είναι απαραίτητος ο αστερίσκος;

Διότι προσδιορίζει ότι δηλώνεται μία μεταβλητή με δείκτη κι όχι μία κανονική μεταβλητή.

ΠΡΟΣΟΧΗ : Αν και η μεταβλητή με δείκτη περιέχει μία διεύθυνση (ακέραιος αριθμός), ΔΕΝ ΕΙΝΑΙ ΙΔΙΑ με μία κανονική ακέραια μεταβλητή. Ο μεταγλωττιστής γνωρίζει ότι η τιμή της μεταβλητής με δείκτη είναι μία συγκεκριμένη διεύθυνση μνήμης, σε αντιδιαστολή με την «κανονική» ακέραια τιμή.

Δήλωση δείκτη

Ο αστερίσκος συνδέεται με το όνομα κι όχι με τον τύπο:

int *pcount; // δείκτης σε ακεραίους, με ονομασία ***pcount***

int *pcount, *pnum; /* δείκτες σε ακεραίους, με ονομασίες ***pcount*** και ***pnum*** */

int *pcount, number; /* ένας δείκτης σε ακέραιο, με ονομασία ***pcount*** και ένας ακέραιος με ονομασία ***number*** */

Δήλωση δείκτη

Πώς επιλέγεται το όνομα ενός δείκτη;

- **Οι ίδιες συμβάσεις που ισχύουν στις κανονικές μεταβλητές.**
- **Ωστόσο, συνήθως ο αρχικός χαρακτήρας του ονόματος δείκτη είναι το p, έτσι ώστε το πρόγραμμα να καθίσταται περισσότερο ευανάγνωστο, καθώς με τον πρώτο χαρακτήρα φαίνεται εάν μία μεταβλητή είναι δείκτης ή όχι. Εναλλακτικά, μπορούμε να προσθέτουμε την κατάληξη _ptr.**

Παράδειγμα:

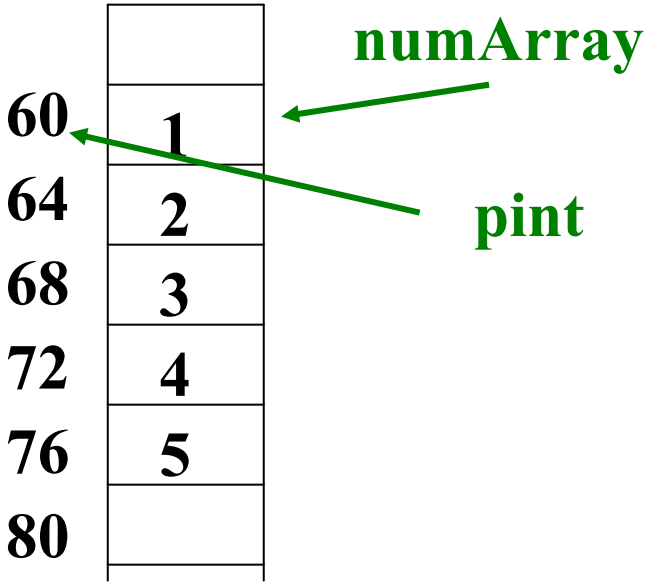
- **int *pcount, *count_ptr;** // δείκτες σε ακεραίους
- **char *pword, *word_ptr;** // δείκτες σε χαρακτήρες

Αρχικοποίηση δεικτών

1) Χρησιμοποιώντας πίνακα

(<u>Υπενθύμιση</u>: το όνομα ενός πίνακα είναι μία διεύθυνση.

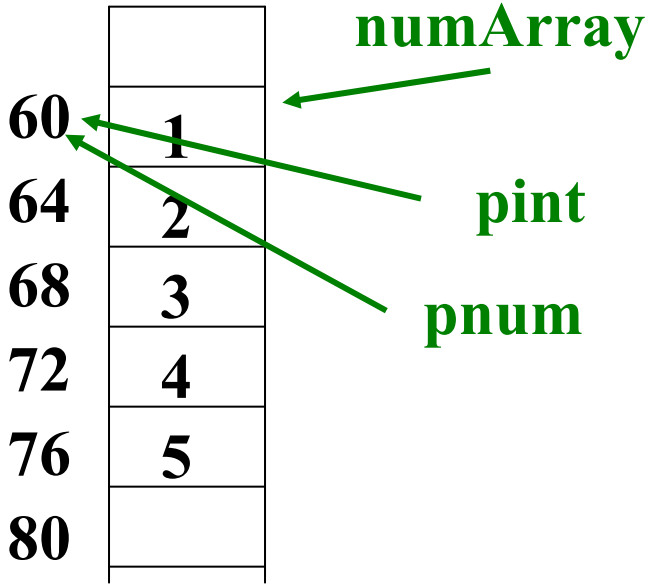
```
int  numArray[5] = {1,2,3,4,5};

int *pint;

pint = numArray;
```



Αρχικοποίηση δεικτών

2) Χρησιμοποιώντας άλλους δείκτες ίδιου τύπου

```
int  numArray[5] = {1,2,3,4,5};
int *pint, *pnum;
pint = numArray;
pnum = pint;
```

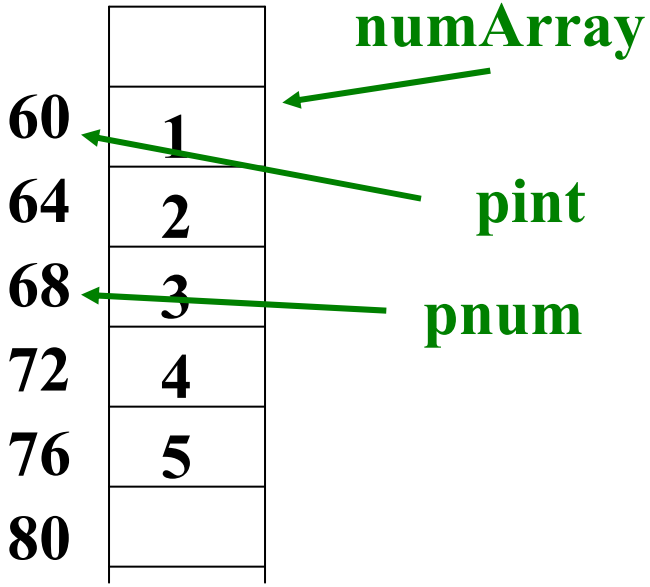


Αρχικοποίηση δεικτών

3) Χρησιμοποιώντας αριθμητική δεικτών

```
int  numArray[5] = {1,2,3,4,5};
int *pint, *pnum;
pint = numArray;
pnum = pint+2;
```

Πήγαινε δύο θέσεις <u>ακεραίων</u> πιο κάτω



Αρχικοποίηση δεικτών

4) Χρησιμοποιώντας τον τελεστή διεύθυνσης & (address-of operator)

```
int *pnum;

int count;

pnum = &count;
```

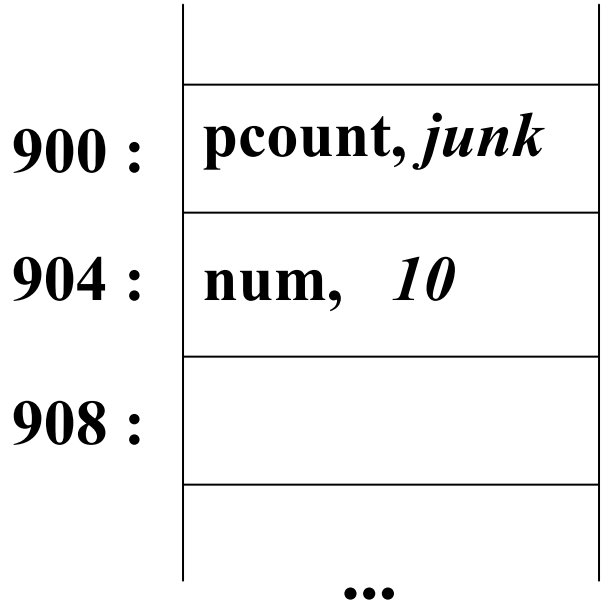
Η γραμμή αυτή αναφέρει: ο pnum να λάβει ως τιμή τη διεύθυνση της μεταβλητής count, δηλαδή ο δείκτης pnum να «δείχνει» στη μεταβλητή count.

Ακολούθως φαίνεται πώς μπορούμε να προσπελάσουμε την τιμή μίας μεταβλητής με τη χρήση δείκτη.

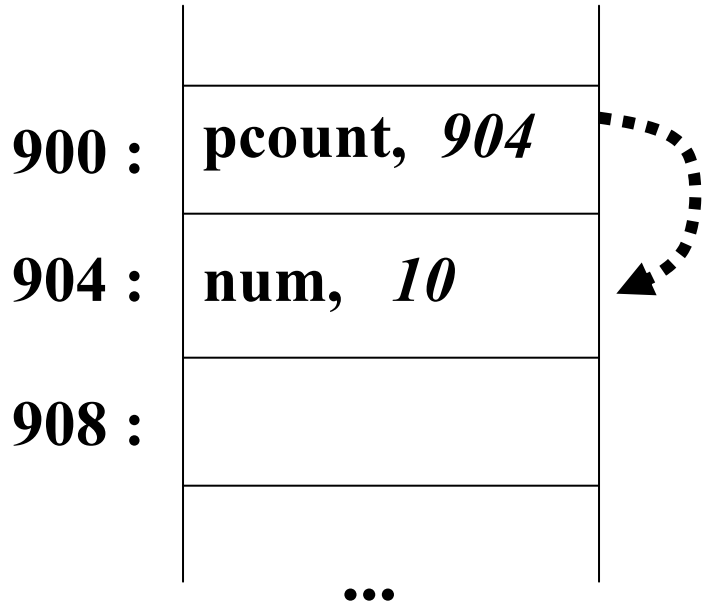
```
int *pcount, num;
num = 10;
pcount = &num;
*pcount = 20;
```

900 :	pcount, *junk*
904 :	num, *junk*
908 :	
	...

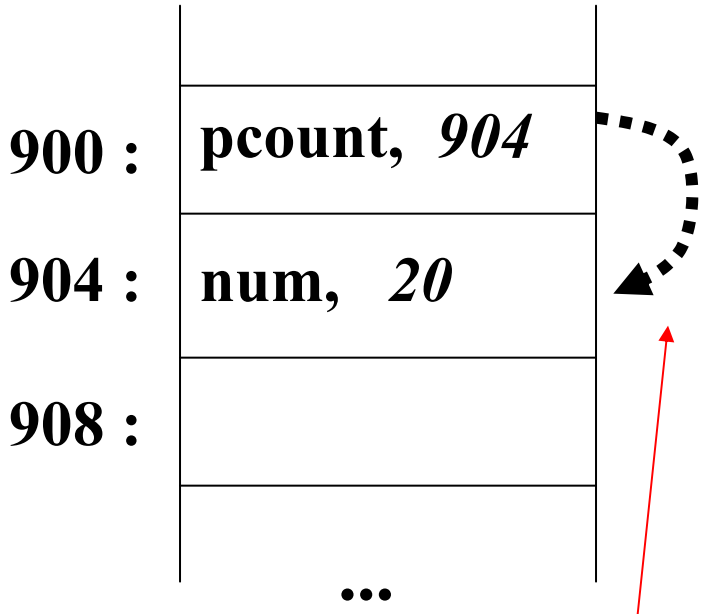
```
int *pcount, num;
num = 10;
pcount = &num;
*pcount = 20;
```




```
int *pcount, num;
num = 10;
pcount = &num;
*pcount = 20;
```



```
int *pcount, num;
num = 10;
pcount = &num;
*pcount = 20;
```



Ο αστερίσκος υποδηλώνει ότι πρέπει να ακολουθηθεί το <u>βέλος</u> για να προσπελασθούν τα δεδομένα της θέσης, στην οποία δείχνει.

***pcount = 20;**

- **Ο αστερίσκος ονομάζεται *τελεστής περιεχομένου* (dereferencing operator).**
- **Διαβάζεται «*στη διεύθυνση*».**
- **Χρησιμοποιείται για να προσπελαύνει τα περιεχόμενα της θέσης μνήμης στην οποία δείχνει ο δείκτης.**
- **Δε θα πρέπει να συγχέεται με τον αστερίσκο της δήλωσης δείκτη.**

Εφαρμογή δεικτών

Στο παράδειγμα της συνάρτησης *swap()* κατέστη φανερό ότι έπρεπε να γίνουν διορθώσεις:

- **Μπορούμε να χρησιμοποιήσουμε δείκτες για να μεταβάλλουμε έμμεσα τις τιμές των μεταβλητών της *main()* μέσα από τη συνάρτηση *swap()*.**
- **Γνωρίζουμε πλέον:**
  - **Πώς ορίζονται οι δείκτες.**
  - **Πώς τους αρχικοποιούμε με τη διεύθυνση μίας μεταβλητής.**
  - **Πώς προσπελαύνουμε την τιμή της μεταβλητής.**

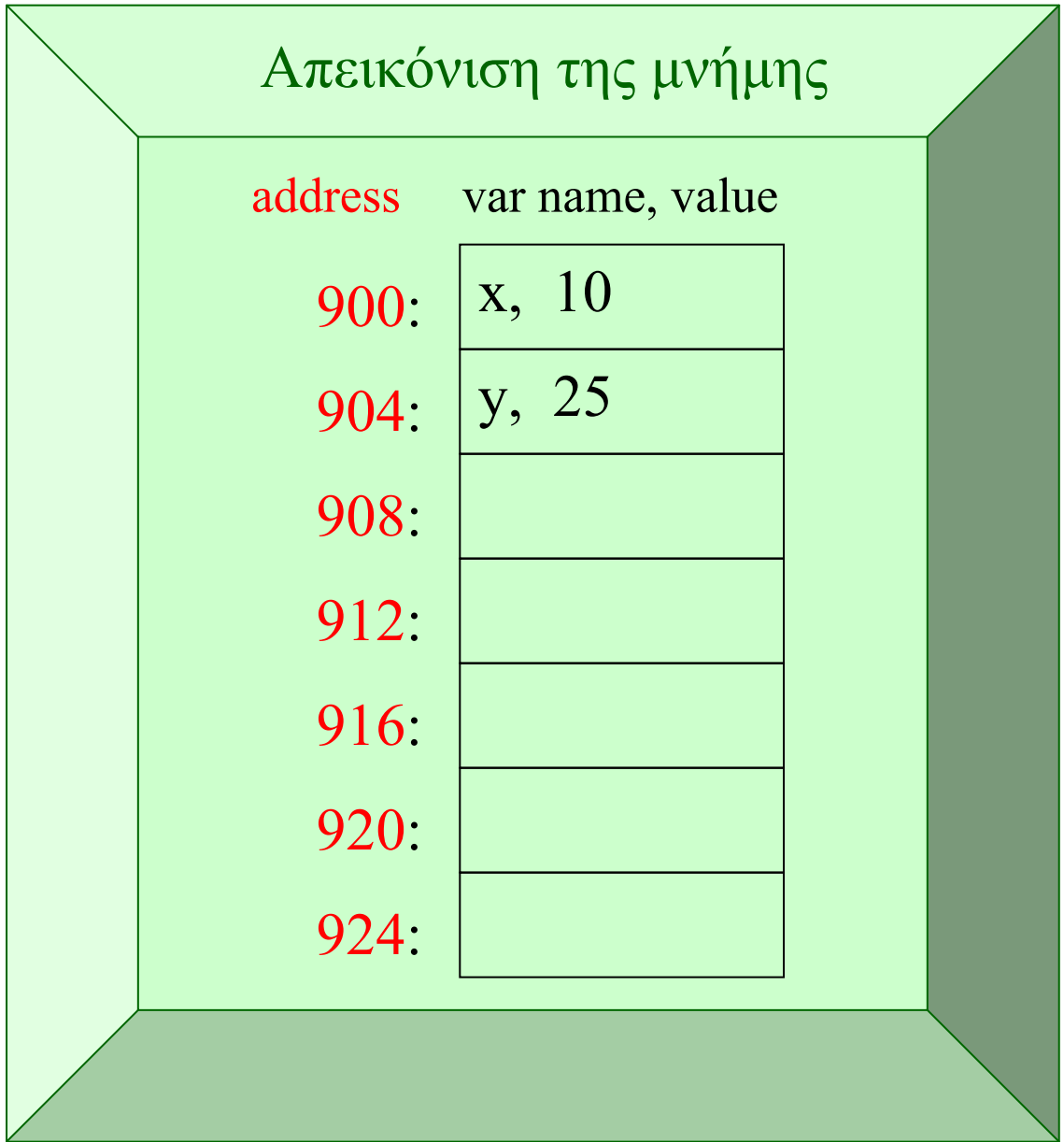
Εφαρμογή δεικτών

```
main()
{
   int x=10, y=25;
   int *px, *py;
   px = &x;
   py = &y;
   swap(px, py);
}

void swap (int *pa, int *pb) {
   int temp;
   temp = *pa;
   *pa = *pb;
   *pb = temp;
}
```

Τα ορίσματα της συνάρτησης είναι τώρα δείκτες σε ακέραιες μεταβλητές.

Θα χρησιμοποιήσουμε μαύρο χρώμα για τις τοπικές μεταβλητές της main() και μπλε για τις τοπικές μεταβλητές της swap().

```
main()
{
    int x=10, y=25;
    int *px, *py;
    px = &x;
    py = &y;
    swap(px, py);
}

void swap (int *pa, int *pb) {
    int temp;
    temp = *pa;
    *pa = *pb;
    *pb = temp;
}
```

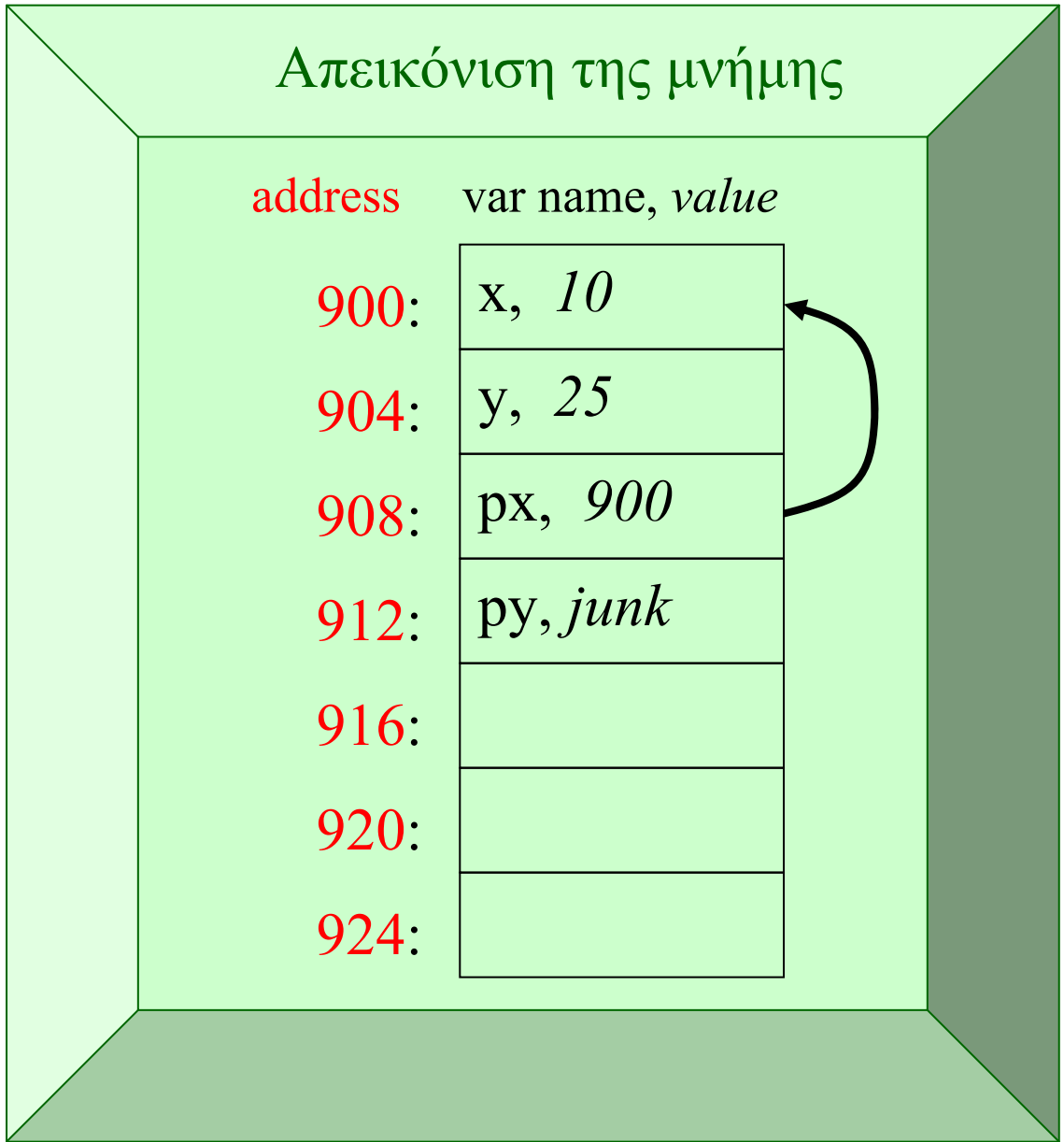


```
main ()
{
    int x=10, y=25;
    int *px, *py;
    px = &x;
    py = &y;
    swap(px, py);
}

void swap (int *pa, int *pb) {
    int temp;
    temp = *pa;
    *pa = *pb;
    *pb = temp;
}
```

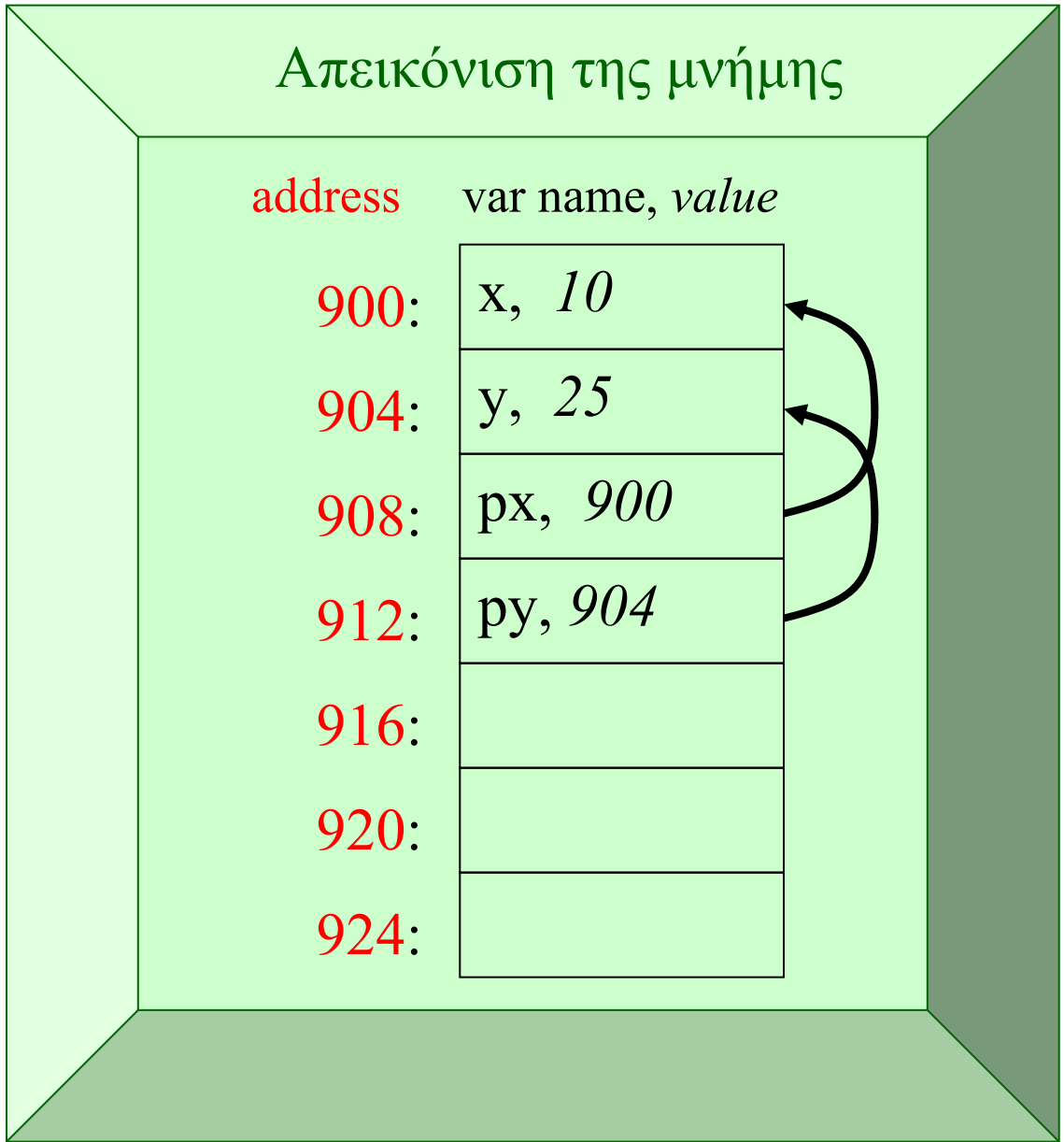
Απεικόνιση της μνήμης

address	var name, value
900:	x, 10
904:	y, 25
908:	px, *junk*
912:	py, *junk*
916:	
920:	
924:	

```
main ()
{
    int x=10, y=25;
    int *px, *py;
    px = &x;
    py = &y;
    swap(px, py);
}

void swap (int *pa, int *pb) {
    int temp;
    temp = *pa;
    *pa = *pb;
    *pb = temp;
}
```



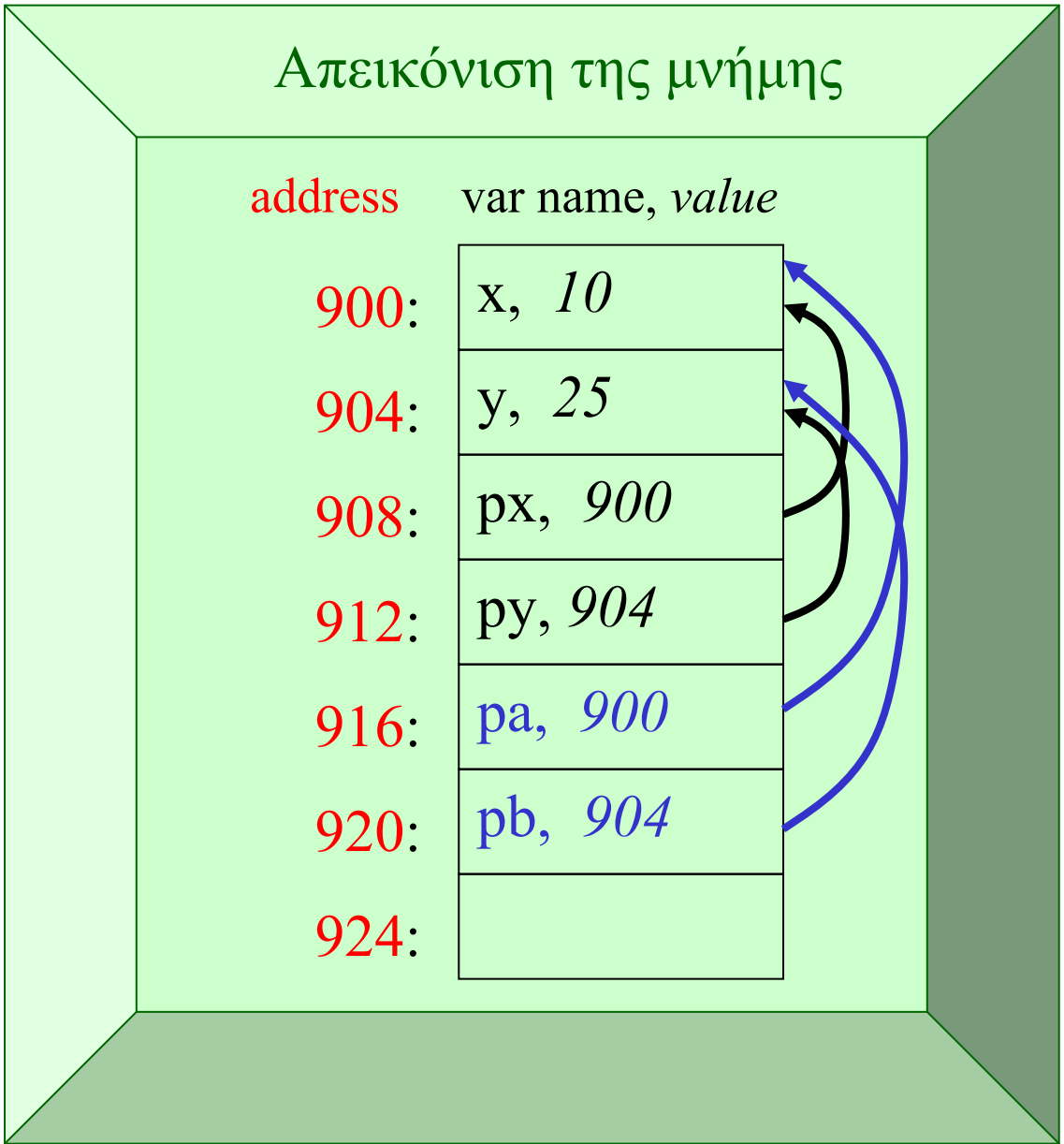
```
main ()
{
    int x=10, y=25;
    int *px, *py;
    px = &x;
    py = &y;
    swap(px, py);
}

void swap (int *pa, int *pb) {
    int temp;
    temp = *pa;
    *pa = *pb;
    *pb = temp;
}
```



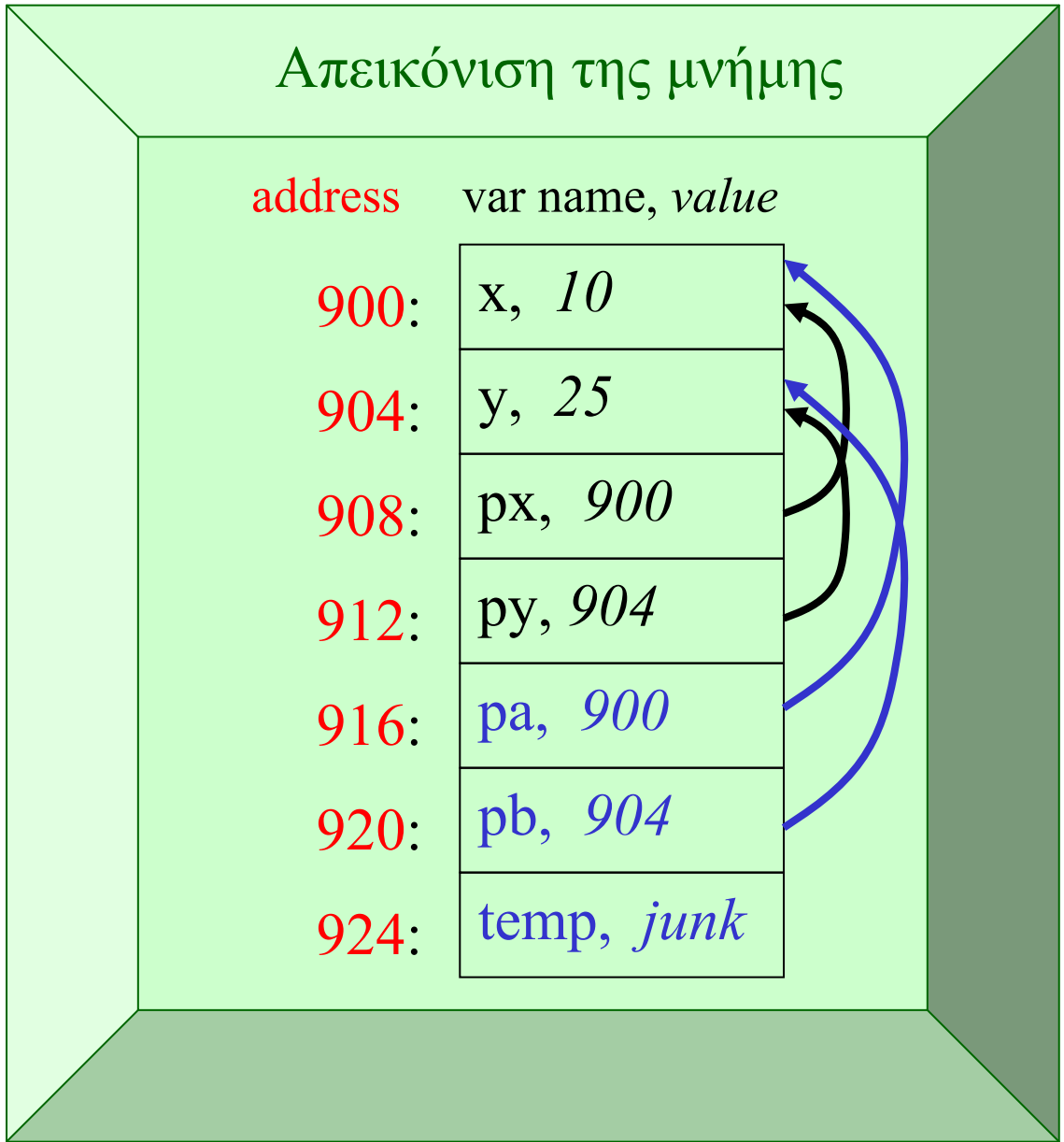
```
main ()
{
    int x=10, y=25;
    int *px, *py;
    px = &x;
    py = &y;
    swap(px, py);
}
```

Αντίγραφο της τιμής της *py*

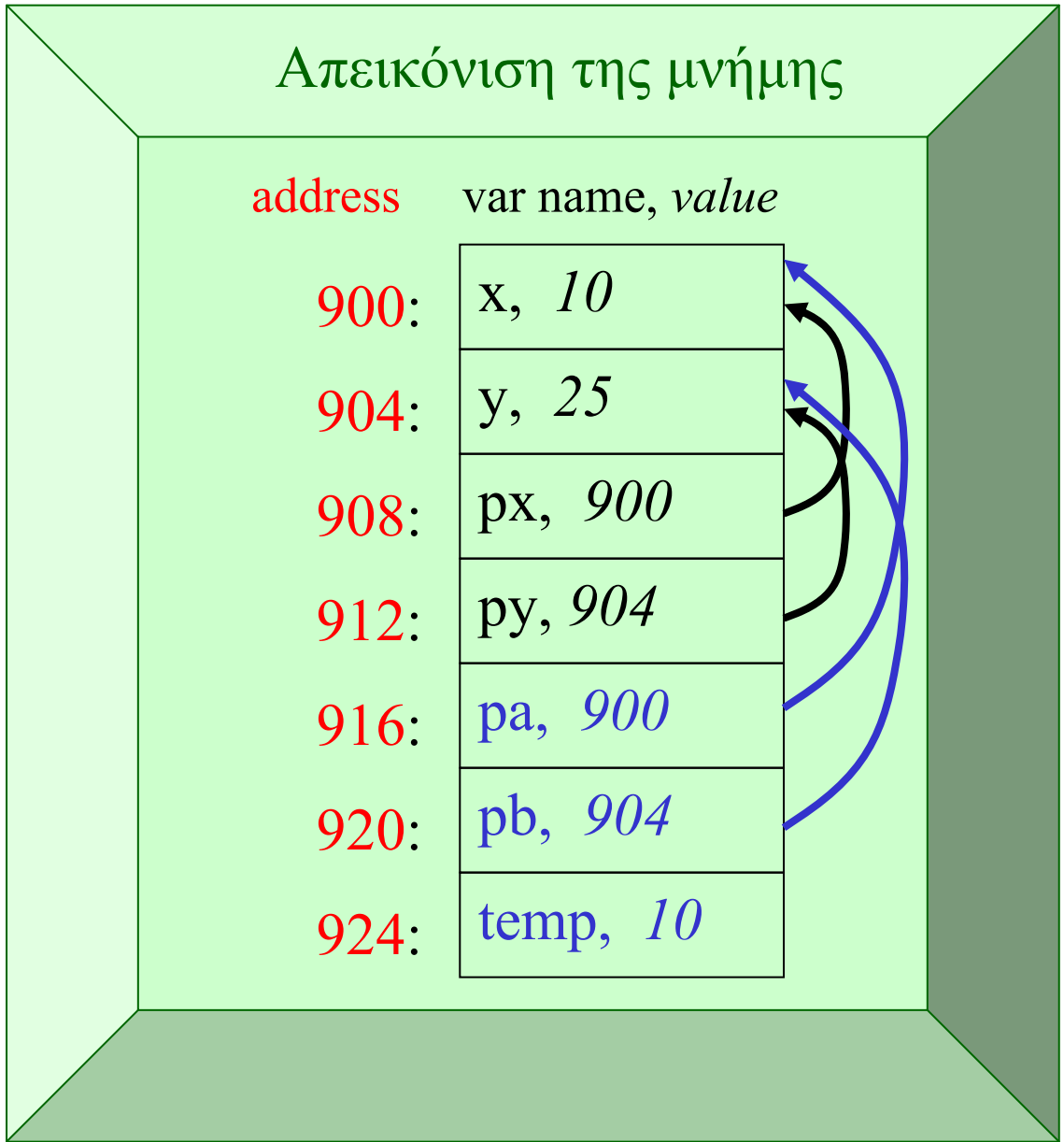
```
void swap (int *pa, int *pb) {
    int temp;
    temp = *pa;
    *pa = *pb;
    *pb = temp;
}
```



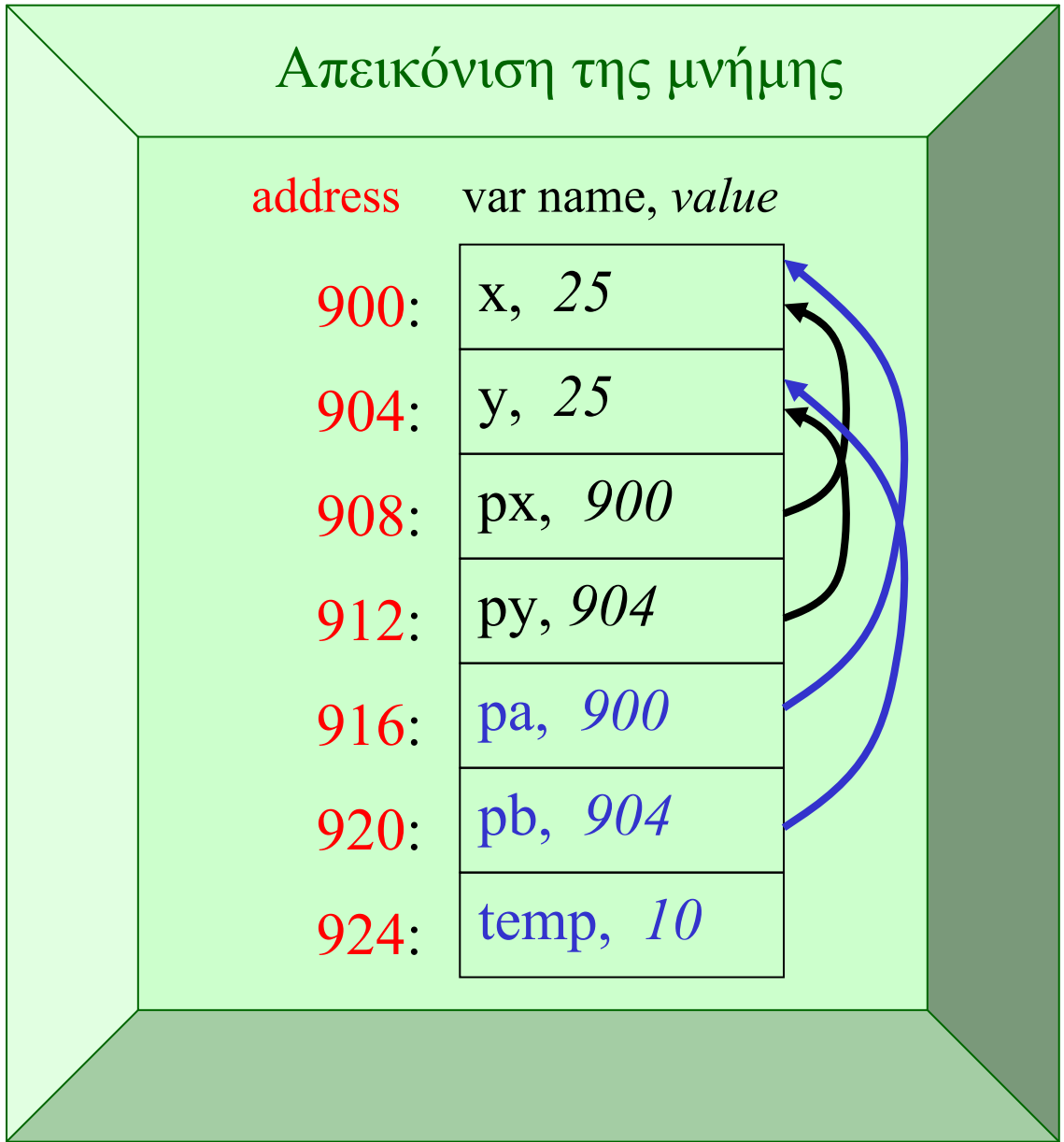
```
main ()
{
    int x=10, y=25;
    int *px, *py;
    px = &x;
    py = &y;
    swap(px, py);
}

void swap (int *pa, int *pb) {
    int temp;
    temp = *pa;
    *pa = *pb;
    *pb = temp;
}
```



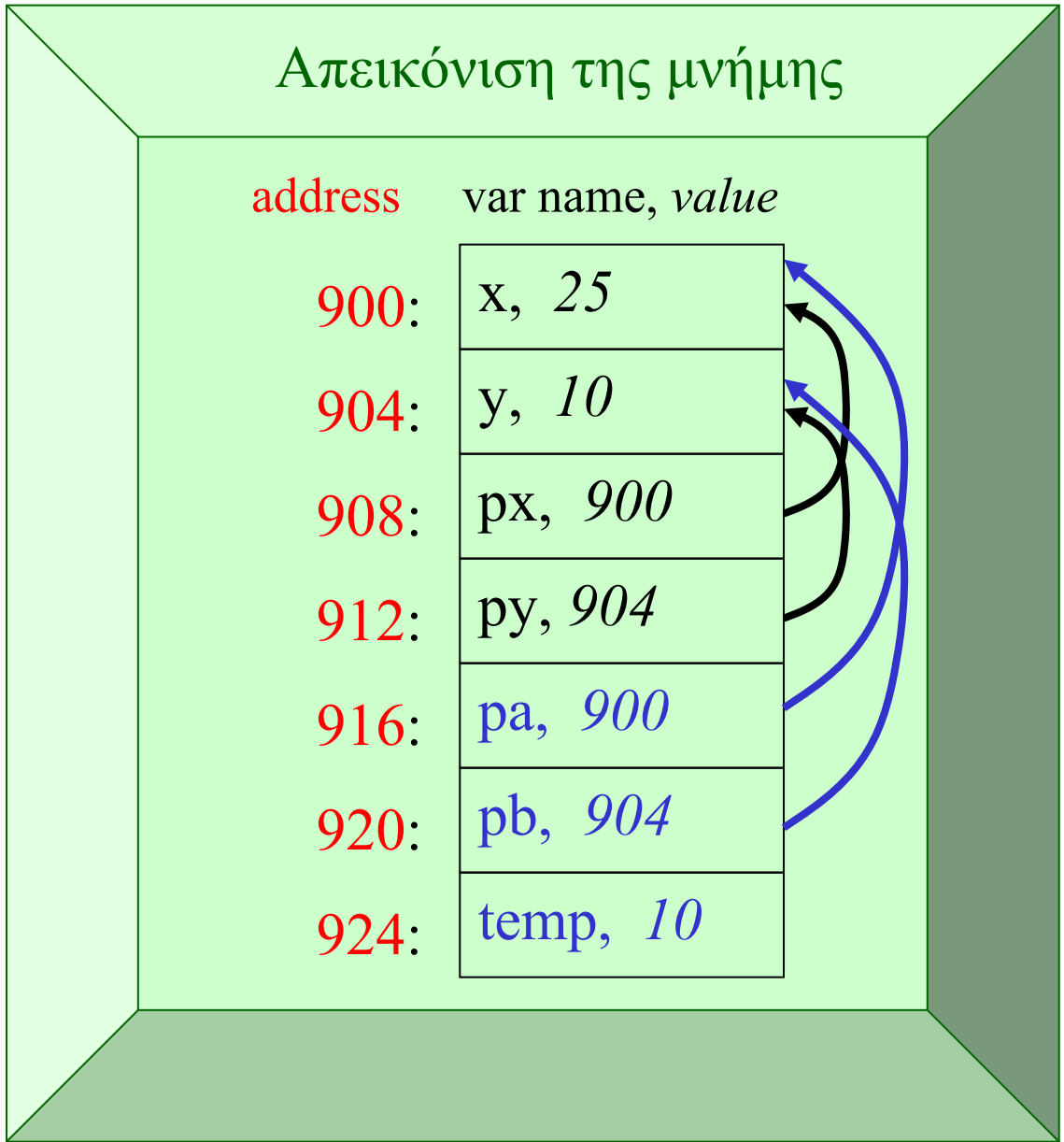
```
main ()
{
    int x=10, y=25;
    int *px, *py;
    px = &x;
    py = &y;
    swap(px, py);
}

void swap (int *pa, int *pb) {
    int temp;
    temp = *pa;
    *pa = *pb;
    *pb = temp;
}
```



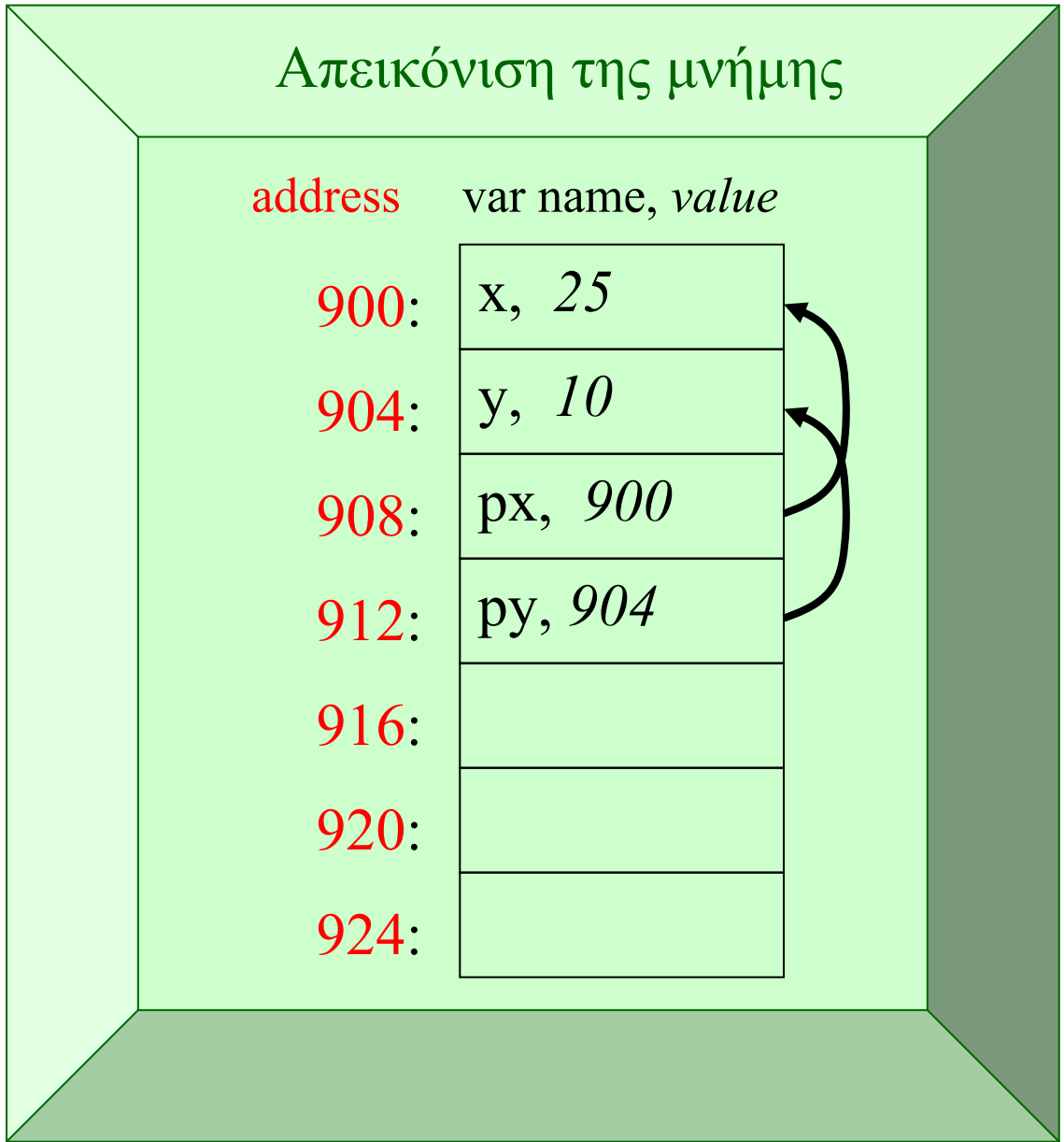
```
main ()
{
    int x=10, y=25;
    int *px, *py;
    px = &x;
    py = &y;
    swap(px, py);
}

void swap (int *pa, int *pb) {
    int temp;
    temp = *pa;
    *pa = *pb;
    *pb = temp;
}
```



```
main ()
{
    int x=10, y=25;
    int *px, *py;
    px = &x;
    py = &y;
    swap(px, py);
}

void swap (int *pa, int *pb) {
    int temp;
    temp = *pa;
    *pa = *pb;
    *pb = temp;
}
```



```
main ()
{
    int x=10, y=25;
    int *px, *py;
    px = &x;
    py = &y;
    swap(px, py);
}

void swap (int *pa, int *pb) {
    int temp;
    temp = *pa;
    *pa = *pb;
    *pb = temp;
}
```



Επεξηγήσεις:

- Αν και η συνάρτηση *swap()* δεν επιστρέφει τίποτε άμεσα στη *main()*, έχει μία παρενέργεια (side effect).
- Όταν καλείται η *swap()*, τα ορίσματά της είναι οι δείκτες px και py, οι οποίες σχετίζονται με τις διευθύνσεις των x και y, αντίστοιχα.
- Παρατήρηση: δε χρειάζεται να δηλώσουμε τους δείκτες px και py. Το μόνο που απαιτείται είναι να περασθούν οι διευθύνσεις των x και y στη *swap()*, όπως φαίνεται ακολούθως:

```
main ()
{
   int x=10, y=25;
   swap(&x, &y);
}
```

```
void swap (int *pa, int *pb)
 {
   int temp;
   temp = *pa;
   *pa = *pb;
   *pb = temp;
 }
```

Λειτουργεί με τον ίδιο τρόπο που λειτουργούσε το πρόγραμμα με τους **px** *και* **py**.

pointer_ex.cpp

Δείκτες και συναρτήσεις

- Έως τώρα περνούσαμε τους δείκτες ως ορίσματα σε συναρτήσεις. Μπορούμε όμως να επιστρέφουμε δείκτες;
- Όταν επιστρέφεται ένας δείκτης, θα πρέπει να δείχνει σε δεδομένα της καλούσας συνάρτησης.
- Δεν πρέπει ποτέ να επιστρέφεται δείκτης που δείχνει σε τοπική μεταβλητή της καλούμενης συνάρτησης, γιατί όταν τερματισθεί η συνάρτηση οι τοπικές μεταβλητές εξαφανίζονται.
- Μπορούμε να χρησιμοποιήσουμε δείκτη για να αλλάξουμε το περιεχόμενο της θέσης στην οποία δείχνει, αλλά δεν πρέπει να αλλάξουμε τον ίδιο το δείκτη μέσα στην καλούμενη συνάρτηση.

```
main () {
    int *pscore, num;
    num = 32;
    pscore = &num;
    print(pscore);
}
```

Αντιγράφει την τιμή του ***pscore*** (δηλαδή τη διεύθυνση στην οποία δείχνει) στον ***ptr***

```
void print(int *ptr) {
    printf("%d", *ptr);
    ptr=ptr+1;
}
```

ΚΑΚΟ! Όταν τυπώνει η συνάρτηση τελειώνει και ο ***ptr*** εξαφανίζεται.

Ο ***pscore*** είναι ό,τι ήταν και πριν την κλήση της συνάρτησης ***print***.

```
main () {
    int *pscore, num;
    num = 32;
    pscore = incr(num);

}
```

```
int *incr(int x) {
    x = x+10;
    return &x;
}
```

ΛΑΘΟΣ! Όταν τελειώνει η *incr*, η *x* εξαφανίζεται και η τιμή της χάνεται.

Ωστόσο, πίσω στη συνάρτηση ο *pscore* θα δείχνει στη διεύθυνση που επέστρεψε από τη συνάρτηση αλλά θα υπάρχει «σκουπίδι» (junk) σ' αυτή τη διεύθυνση, εφόσον το *x* έχει εξαφανισθεί.

Σωστή χρήση της επιστρεφόμενης τιμής διεύθυνσης

```
// pointer_func.cpp
#include <stdio.h>


int *incr(int *pkitsos);

main()    {
  int *pscore, *pm, num;

  num=32;
  pscore=&num;

  printf( "addr(num)=%d  addr(pscore)=%d  addr(pm)=%d
pscore=%d\n\n*pscore=%d\n\n",&num,&pscore,&pm,pscore,*pscore
);

  pm=incr(pscore);

  printf( "pm=%d  num=%d\n\n\n",pm,num );
}
```

```
int *incr(int *pk)  {
  int x;
  x=*pk;
  printf( "Prior: addr(pk)=%d  pk=%d  addr(x)=%d
x=%d\n\n",&pk,pk,&x,x);
  x=x+10;
  *pk=x;
  printf( "*pk=%d\n\n",*pk);
  return(pk);
}
```

C:\temp\Project1.exe

```
addr(num)=1245056   addr(pscore)=1245064   addr(pm)=1245060   pscore=1245056

*pscore=32

Prior: addr(pk)=1245052   pk=1245056   addr(x)=1245040   x=32

*pk=42

pm=1245056   num=42
```

pointer_return_value.cpp

(Υπενθύμιση) Συνάρτηση και Πίνακες

Εάν η πραγματική παράμετρος είναι όνομα πίνακα (πχ. **key**), στέλνει τη διεύθυνση του πρώτου byte του πίνακα.

Η παράμετρος στη δήλωση της συνάρτησης είναι ένα όνομα τοπικού πίνακα (π.χ. **x**). Κρατά ένα αντίγραφο της ίδιας διεύθυνσης αλλά χρησιμοποιεί διαφορετικό όνομα.

Πραγματική παράμετρος

```
main ()
{
  float key[10];
    setKeyA(key,4);
}
/*key has 10 varied #s */
```

Παράμετρος συνάρτησης

```
setKeyA(float x[], int s)
{ /* 10 seeded random #s */
int i;
    srand(s); /* seed it */
    for(i=0; i<10; i++)
              x[i] = rand();
}
```

Κλήση κατ' αξία

- Το όρισμα **4** αντιγράφει μία ΤΙΜΗ στη συνάρτηση. Η διαδικασία αυτή ονομάζεται **κλήση κατ' αξία (call by value)**.

Πραγματική παράμετρος

Παράμετρος συνάρτησης

```
main ()
{
float key[10];
     setKeyA(key,4);
}
/*key has 10 varied #s */
```

```
setKeyA(float x[], int s)
{ /* 10 seeded random #s */
int i;
     srand(s); /* seed it */
     for(i=0; i<10; i++)
                x[i] = rand();
}
```

Κλήση κατ' αξία

Το όρισμα **key** αντιγράφει τη ΔΙΕΥΘΥΝΣΗ σε μία συνάρτηση. Η συνάρτηση χρησιμοποιεί αυτή τη διεύθυνση για να *βρει* *το προς χρήση δεδομένο.*

Πραγματική παράμετρος

```
main ()
{
float key[10];
     setKeyA(key,4);
}
/*key has 10 varied #s */
```

Παράμετρος συνάρτησης

```
setKeyA(float x[], int s)
{ /* 10 seeded random #s */
int i;
     srand(s); /* seed it */
     for(i=0; i<10; i++)
                x[i] = rand();
}
```

Κλήση κατ' αναφορά

Όταν τα ορίσματα είναι πίνακες, περνιούνται στις συναρτήσεις **'κατ' αναφορά' (call by reference)** — και μπορεί να είναι ΤΟΣΟ είσοδοι ΟΣΟ και έξοδοι.

(τόσο η `main()` όσο και η `setKeyA()` μπορούν να θέσουν τιμές σε στοιχεία πινάκων).

Πραγματική παράμετρος

```
main ()
{
float key[10];
     setKeyA(key, 4);
}
/*key has 10 varied #s */
```

Παράμετρος συνάρτησης

```
setKeyA(float x[], int s)
{ /* 10 seeded random #s */
int i;
     srand(s); /* seed it */
     for(i=0; i<10; i++)
                x[i] = rand();
}
```

Κλήση κατ' αναφορά

- **Οι δείκτες επιτρέπουν να περνάμε ΟΠΟΙΟΔΗΠΟΤΕ δεδομένο κατ'αναφορά**
- Παράδειγμα: αλλάζουμε την **setKeyA** έτσι ώστε να δέχεται ως ορίσματα μόνο δείκτες και διευρύνονται οι δυνατότητες. **(αλλάζουμε το όνομα σε setKeyP)**

Πραγματική παράμετρος

```
main ()
{ /*double-width key */
int seed0 = 4; seed1 = 89;
float key2[20];
   setKeyP(&key2[0], &seed0);
   setKeyP(&key2[10],&seed1);
}
```

Παράμετρος συνάρτησης

```
setKeyP(float *pK, int *pS)
{ /* 10 seeded random #s */
int i;
    srand(*pS); /* seed...*/
    for(i=0; i<10; i++)
            pK[i] = rand();
    *pS=0;      /* clear it*/
}
```

Κλήση κατ' αναφορά

```
main ()
{ /*double-width key */
int seed0 = 4; seed1 = 89;
float key2[20];
   setKeyP(&key2[0], &seed0);
   setKeyP(&key2[10],&seed1);
} /* now seed0,seed1 are 0 */
```

```
setKeyP(float *pK, int *pS)
{ /* 10 seeded random #s */
int i;
    srand(*pS); /* seed...*/
    for(i=0; i<10; i++)
            pK[i] = rand();
    *pS=0;      /* clear it*/
}
```

//rand_scenarios

Συναρτήσεις και Δείκτες

Οι πραγματικές παράμετροι που είναι δείκτες αντιγράφουν μία **διεύθυνση στις** παραμέτρους της συνάρτησης, αλλά εάν αλλαχθεί η παράμετρος στη συνάρτηση (δηλ. η διεύθυνση) ΔΕ θα αλλαχθεί η πραγματική παράμετρος !!

Πραγματικό όρισμα

```
main ()
{
double cbn[2]={1.2,5.0};
double *pC;
    pC = &cbn[1];
    dfixit(pC);
} /* now cbn[1]=42.0, */
  /* but pC unchanged */
```

Παράμετρος συνάρτησης

```
dfixit(double *x)
{
    x[0] = 42.0;
    x--;        /*move x?*/
}
```

ΟΧΙ!!

Ο δείκτης ως όρισμα

- **Ταχύ, αποδοτικό**

(αντιγράφει μόνο τη διεύθυνση, όχι όλα τα δεδομένα)

- **Ισχυρό: μπορεί να είναι είσοδος, έξοδος ή και τα δύο**

→!!ΚΙΝΔΥΝΟΣ!!←

Τι γίνεται όταν έχουμε άκυρα ορίσματα;

<u>Απάντηση</u>: Να αναμένετε το χείριστο χάος – το λανθασμένο στοιχείο να αποθηκεύεται τη λάθος στιγμή σε λάθος θέση!!

Συναρτήσεις με τύπο επιστροφής δείκτη

Οι συναρτήσεις των οποίων ο τύπος επιστροφής είναι δείκτης;

ΠΩΣ: η δήλωση της συνάρτησης μοιάζει:

```
char *findNextVowel(char *str);
```

→!!ΚΙΝΔΥΝΟΣ!!←

- **Όλες οι μεταβλητές της συνάρτησης είναι τοπικές και προσωρινές. Μπορεί να εξαφανισθούν όταν φύγουμε από τη συνάρτηση.**
- **Μην επιστρέφετε δείκτες σε μη ορισμένες μεταβλητές!**

- **Να μην επιστρέφετε ποτέ δείκτες ΕΚΤΟΣ εάν χρησιμοποιείτε τη λέξη κλειδί "static".**

Παράδειγμα static

ΣΦΑΛΜΑ

```
main()
 {
 float* pKey;

pKey = setKey(0);
 printf("keys 0,7 are %d,%d.\n",pKey[0],pKey[7]);
  ... (κώδικας που δεν αλλάζει τον pKey) ...
 printf("keys 0,7 are %d,%d.\n",pKey[0],pKey[7]);
 }
```

```
 float* setKey(int s)  /* make a cryptographic key */
 {
 float keep[10];
 int i;
    srand(s);       /* set rand's seed */
     for(i=0; i<10; i++) keep[i] = rand();
     return(keep);
 }
```

Παράδειγμα static

```
main()
 {
 float* pKey;

pKey = setKey(0);
 printf("keys 0,7 are %d,%d.\n",pKey[0],pKey[7]);
  ... (code that doesn't change pKey) ...
 printf("keys 0,7 are %d,%d.\n",pKey[0],pKey[7]);
 }
```

ΣΦΑΛΜΑ

```
 float* setKey(int s)  /* make a crypt
 {
 float keep[10];
 int i;
    srand(s);       /* set rand's seed *
     for(i=0; i<10; i++) keep[i] = rand
     return(keep);
 }
```

ΔΙΟΤΙ:
Στην έξοδο η προσωρινή μεταβλητή ***i*** και ο πίνακας ***keep*** είναι ακαθόριστοι

Παράδειγμα static

ΣΩΣΤΟ

```
main()
 {
 float* pKey;

pKey = setKey(0);
 printf("keys 0,7 are %d,%d.\n",pKey[0],pKey[7]);
  ... (code that doesn't change pKey) ...
 printf("keys 0,7 are %d,%d.\n",pKey[0],pKey[7]);
 }
```

```
float* setKey(int s)  /* make a crypto
   {
   static float keep[10];
   int i;
      srand(s);        /* set rand's seed
       for(i=0; i<10; i++) keep[i] = rand();
       return(keep);
   }
```

Η λέξη κλειδί '`static`' διατηρεί τον πίνακα ***keep***, ακόμη και μετά την έξοδο από τη συνάρτηση.

επιστροφή(δείκτης)→→Πάντοτε έλεγξε για ***static***!

Παράδειγμα: *Το ακόλουθο πρόγραμμα αφορά σε δείκτες που δείχνουν σε δομές και παρουσιάζει συγκριτικά τον τρόπο λειτουργίας της κλήσης κατ' αξία και της κλήσης κατ' αναφορά.*

```
#include<conio.h>                                  //vectors
#include<stdio.h>

struct vector  //define the structure vector
{
  float x,y;
};
//-----------------------------------
// function declaration
vector readvect(); // Reads a vector, call by value (cbv)
void prvect(char d, vector v); // Prints a vector, (cbv)
void scanvect( vector *p); // Reads a vector,call by reference (cbr)
float inprod(vector v, vector u); // Inner product, (cbv)
float inprodr(vector *p, vector *q); // Inner product, (cbr)
vector addvect(vector v, vector u);  // Add vectors, (cbv)
vector addvectr(vector *p, vector *q); // Add vectors, (cbr)
```

//vectors

```
main()
{
 vector a,b,c;
 a=readvect();
 prvect('a',a);
 scanvect(&b);
 prvect('b',b);
 printf("v a*b=%.2f\n",inprod(a,b));
 printf("r a*b=%.2f\n",inprodr(&a,&b));
 c = addvect(a,b);
 printf("cbv addition: ");
 prvect('c',c);
 addvectr(&a,&b,&c);
 printf("cbr addition: ");
 printf("Vector %c is  (%.2f,%.2f)\n",'c',c.x,c.y);
 printf("\t\t\tPRESS ANY KEY TO FINISH\n" );
 getch();
} //end of main
```

```
void prvect(char d, vector v)
{
  printf( "Vector %c is  ",d );
  printf( "(%.2f,%.2f)\n",v.x,v.y );
} //end of prvect
vector readvect()
{
  vector v;
  printf( "Give the x co-ordinate:" ); scanf("%f",&v.x);
  printf( "Give the y co-ordinate:" ); scanf("%f",&v.y);
  return(v);
} //end of readvect
void scanvect( vector *p)
{
  printf( "Give the x co-ordinate:" ); scanf("%f",&p->x);
  printf( "Give the y co-ordinate:" ); scanf("%f",&p->y);
} //end of scanvect
```

Προγραμματισμός ΙΙ

```
float inprod(vector v, vector u)
{
  return(v.x*u.x+v.y*u.y);
} //end of inprod
float inprodr(vector *p, vector *q)
{// parentheses are not necessary, due to precedence
  return((*p).x*(*q).x+(p->y)*(q->y));
} //end of inprodr
vector addvect(vector v, vector u)
{
  vector sum;
  sum.x=v.x+u.x;    sum.y=v.y+u.y;
  return(sum);
} //end of addvect
void addvectr(vector *p, vector *q, vector *t)
{
  t->x=(p->x)+(q->x);    t->y=(p->y)+(q->y);
} //end of addvectr
```

Αποτέλεσμα:

```
C:\temp\Project1.exe
Give the x co-ordinate:-3.2
Give the y co-ordinate:2
Vector a is  (-3.20,2.00)
Give the x co-ordinate:-9
Give the y co-ordinate:1.0
Vector b is  (-9.00,1.00)
v a*b=30.80
r a*b=30.80
cbv addition: Vector c is  (-12.20,3.00)
cbr addition: Vector c is  (-12.20,3.00)
                          PRESS ANY KEY TO FINISH
```

pointer_nested_struct.cpp

Παράδειγμα:

Να αναπτυχθεί πρόγραμμα που να λαμβάνει από το πληκτρολόγιο τα στοιχεία ενός εργαζόμενου και να δημιουργεί πίνακα εργαζόμενων, με τύπο δεδομένου κατάλληλη δομή. Η διαδικασία θα επαναλαμβάνεται για 10 διαφορετικούς εργαζόμενους, όσο είναι και το μέγεθος του πίνακα. Οι πληροφορίες που διαβάζονται για κάθε εργαζόμενο είναι:

Ονοματεπώνυμο: Όνομα και επώνυμο ξεχωριστά (*σε μεταβλητή τύπου δομής*)

Διεύθυνση: Όνομα οδού, αριθμός οδού, πόλη, ταχ. Κώδικας (*σε μεταβλητή τύπου δομής*)

Τηλέφωνα: Τηλέφωνο εργασίας, κινητό (*σε μεταβλητή τύπου δομής*)

Θέση: Τίτλος, κωδικός αριθμός εργαζόμενου, τομέας της επιχείρησης στον οποίο εργάζεται, αριθμός γραφείου, ονοματεπώνυμο προϊσταμένου, ημερομηνία πρόσληψης (ημέρα, μήνας, έτος), μισθός (*σε μεταβλητή τύπου δομής – θα απαιτηθεί ένθετη δομή για την ημερομηνία πρόσληψης*)

Στη συνέχεια να δίνεται κάποιος κωδικός αριθμός εργαζόμενου από το πληκτρολόγιο και να αναζητείται στον πίνακα. Αν υπάρχει, τότε να εμφανίζονται στην οθόνη οι πληροφορίες του αντίστοιχου εργαζόμενου, αλλιώς να εμφανίζεται ένα ανάλογο μήνυμα.

```
// Combined use of calling functions  by value and by reference (including structures)
#include<conio.h>
#include<stdio.h>
#define N 2//10  //Number of employees
struct nmT  {
  char name[40],surname[40];
};
struct addressT  {
  char street_name[40], city[40];
  int street_number,zip_code;
};
struct teleT  {
  char office_number[14],home_number[14];
};
struct hiredateT {
  int year,month,date;
};
struct job_descriptionT
{
  char title[40], sector[100], boss_name[40];
  int code_number,office_number,salary;
  hiredateT hire;
};
```

Περισσότερες λεπτομέρειες στο cbr_cbv.cpp

```
struct personnelT
{
  nmT nm;
  addressT addr;
  teleT tele;
  job_descriptionT job;
};

void get_employee(personnelT *pers_ptr);
void search_employee(int i, personnelT person[N]);

main()
{
  personnelT pers[N];
  int i;
  for (i=0;i<N;i++) get_employee(&pers[i]);
  printf("\nGive an employee's code_number:  ");
  scanf("%d",&i);
  search_employee(i,pers);
}
```

Call by reference

Call by value

```
void get_employee(personnelT *pers_ptr)
{
  printf("\t\tAdd new employee:\n");

  printf("\nEmployee's name:  ");
  scanf("%s",pers_ptr->nm.name);
  printf("\nSurname:  ");
  scanf("%s",pers_ptr->nm.surname);

  printf("\t\nEmployee's job details:");
  printf("\nCode number:  ");
  scanf("%d",&pers_ptr->job.code_number);
  printf("\nSalary:  ");
  scanf("%d",&pers_ptr->job.salary);
  printf("\nYear of recruitment:  ");
  scanf("%d",&pers_ptr->job.hire.year);
}
```

```
void search_employee(int code, personnelT person[N])
{
  int j=0;
  int index=-1;
  while ((j<N) && (index==-1))
  {
    if (code==person[j].job.code_number) index=j;
    j++;
  }
  if (index==-1)
        printf("\nThe given code_number does not match to an existing one\n");
  else
  {
    printf("\n\t\tSearch results:\n");
    printf("\nEmployee's name: %s",person[index].nm.name);
    printf("\nSurname: %s",person[index].nm.surname);
    printf("\t\nJob details:");
    printf("\nCode number: %d",person[index].job.code_number);
    printf("\nSalary: %d",person[index].job.salary);
    printf("\nYear of recruitment: %d",person[index].job.hire.year);
  }
}
```

Αποτελέσματα:

```
C:\temp\Project1.exe
                    Add new employee:

Employee's name:  John

Surname:  Doe

Employee's job details:
Code number:  123

Salary:  31

Year of recruitment:  1987
                    Add new employee:

Employee's name:  George

Surname:  Doe

Employee's job details:
Code number:  345

Salary:  71

Year of recruitment:  1985

Give an employee's code_number:  345

                    Search results:

Employee's name: George
Surname: Doe
Job details:
Code number: 345
Salary: 71
Year of recruitment: 1985
```

C:\temp\Project1.exe

```
                    Add new employee:

Employee's name:  John

Surname:  Doe

Employee's job details:
Code number:  123

Salary:  31

Year of recruitment:  1987
                    Add new employee:

Employee's name:  George

Surname:  Doe

Employee's job details:
Code number:  345

Salary:  71

Year of recruitment:  1985

Give an employee's code_number:  43

The given code_number does not match to an existing one
```

Ορίσματα γραμμής διαταγής

Όπως κάθε συνάρτηση, έτσι κι η *main* μπορεί να δεχθεί παραμέτρους, οι οποίες επιτρέπουν να δίνουμε στο καλούμενο πρόγραμμα ένα σύνολο από εισόδους, που καλούνται **ορίσματα γραμμής διαταγής**. Ο μηχανισμός περάσματος ορισμάτων βασίζεται στην ακόλουθη δήλωση της *main*:

```
main(int argc, char *argv[])
{
    ...
}
```

• ***argc*** (*argument count*): ο αριθμός των ορισμάτων της γραμμής διαταγής, συμπεριλαμβανομένου και του ονόματος του προγράμματος.

• ***argv*** (*argument vector*): δείκτης σε πίνακα από δείκτες, που δείχνουν στα ορίσματα της γραμμής διαταγής, τα οποία αποθηκεύονται με τη μορφή αλφαριθμητικών. Ο πίνακας στον οποίο δείχνει ο *argv* έχει ένα επιπλέον στοιχείο, το *argv[argc]*, το οποίο έχει τιμή *NULL* (είναι δείκτης που δε δείχνει πουθενά).

Ορίσματα γραμμής διαταγής

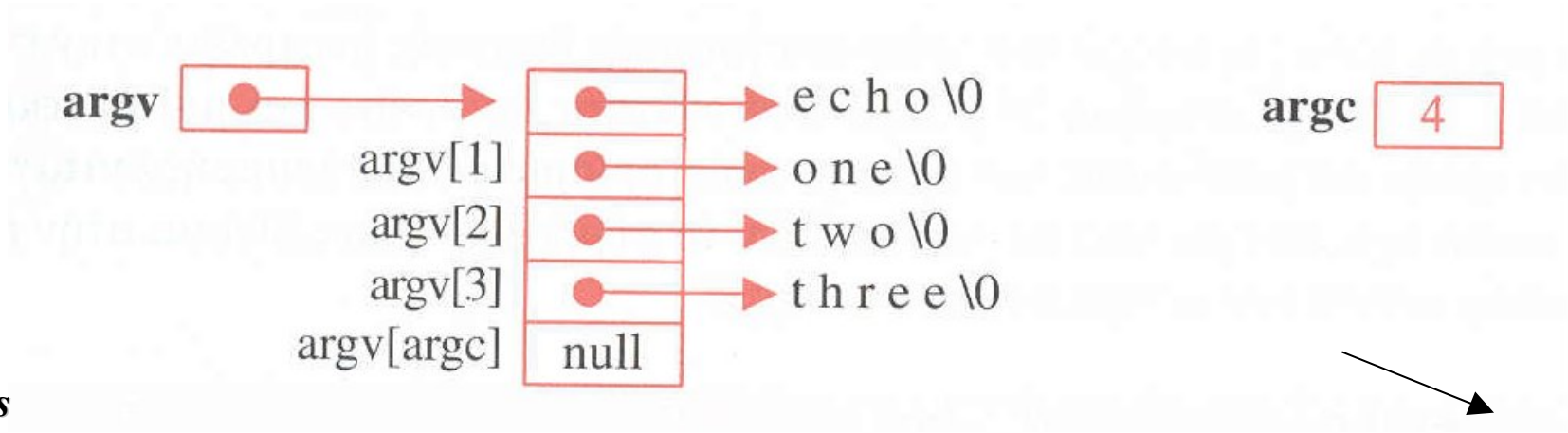
Παράδειγμα: **Να καταστρωθεί πρόγραμμα που θα τυπώνει τα ορίσματα της γραμμής διαταγής**

Λύση: **Έστω *echo* το όνομα του προγράμματος. Όταν το καλούμε με την εντολή**

```
echo one two three
```

θα πρέπει να εκτελείται και να τυπώνει στην οθόνη *one two three*.

Το σώμα της main έχει πρόσβαση στις μεταβλητές argc και argv όπως παριστάνονται από το ακόλουθο σχήμα:



//command_line_args

Ορίσματα γραμμής διαταγής

με βάση το προηγούμενο σχήμα, η διαμόρφωση του σώματος της *main* είναι απλή:

```
main(int argc, char *argv[])
{
  int i;
  for (i=1;i<argc;i++) printf( "%s%s",argv[i]," " );
  printf( "\n" );
}
```

Η `printf( "%s%s",argv[i]," " );` **τυπώνει μετά από κάθε όρισμα ένα κενό. Εάν θέλουμε να μην τυπώνεται το κενό μετά το τελευταίο όρισμα, η πρόταση πρέπει να διαμορφωθεί όπως παρακάτω:**

```
for (i=1;i<argc;i++) printf("%s%s",argv[i],(i<argc-1)?"
":"");
```

Ορίσματα γραμμής διαταγής

Εάν χρησιμοποιήσουμε τη σημειολογία των δεικτών αντί αυτής του πίνακα, η πρόταση μπορεί να γραφεί ως εξής:

```
while (--argc) printf( "%s%s",*++argv,(i<argc-1)?" ":"" );
```

Εναλλακτικά, θα μπορούσαμε να γράψουμε:

```
main(int argc, char *argv[])
{
 while (--argc) printf( (argc>1)?"%s":"%s",*++argv );
}
```

Δυσνόητο; δοκιμάστε το!!