ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ στη C: ΕΙΣΑΓΩΓΗ

http://www.codeblocks.org/

Code::Blocks - The IDE with all the features you need, having a consistent look, feel and operation across platforms.

Σ. Σιούτας

ΣΥΜΒΟΛΟΣΕΙΡΕΣ

Κώδικας ASCII

- Όπως όλα τα δεδομένα στον Η/Υ, οι χαρακτήρες αποθηκεύονται ως δυαδικοί αριθμοί (αντιστοιχούν σε ακέραιους με μέγεθος 1 byte).
- Η αντιστοιχία αυτή για τα σύμβολα και τα λατινικά γράμματα ακολουθεί τον κώδικα ASCII

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
48	49	50	51	52	53	54	55	56	57
A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**
65	66	67	68	69	70	71	72	73	74
a	**b**	**c**	**d**	**e**	**f**	**g**	**h**	**i**	**j**
97	98	99	100	101	102	103	104	105	106
NULL (\0)		**BEL (\g)**		**TAB (\t)**		**NEWLINE (\n)**		**SPACE**	
0		7		9		10		32	

Συμβολοσειρές (Strings)

- Ένα String (συμβολοσειρά ή αλφαριθμητικό) είναι μια σειρά χαρακτήρων που τερματίζεται με έναν ειδικό χαρακτήρα με αριθμητική τιμή 0 ο οποίος συμβολίζεται με το `'\0'` και ονομάζεται null character.
- Στη γλώσσα C τα strings ορίζονται ως πίνακες χαρακτήρων, π.χ.

  ```
  char st[10];  /* πίνακας 10 χαρακτήρων */
  ```

  Οι 9 θέσεις αποθηκεύουν συνήθεις χαρακτήρες
  Ο 10ος χαρακτήρας είναι το \0 (null character)

  ```
  char st[10]={'A','l','a','b','a','m','a','\0'};
  char string[10] = "Alabama";
  ```

  Το **"Alabama"** λέγεται κυριολεκτική ή σταθερή συμβολοσειρά (literal string). Σε τέτοιες συμβολοσειρές η γλώσσα C προσαρτά αυτόματα τον χαρακτήρα \0

Συμβολοσειρές (2)

- Αν δεν δηλώνεται το πλήθος των στοιχείων του πίνακα χαρακτήρων, η γλώσσα το υπολογίζει αυτόματα αν τον αρχικοποιήσουμε, π.χ.

  `char string[] ="Alabama";` /*πίνακας με 8 στοιχεία*/

- Οι συναρτήσεις της γλώσσας που διαχειρίζονται τα strings αναζητούν πάντοτε τον χαρακτήρα τέλους (\0). Αν αυτός ΔΕΝ υπάρχει συνεχίζουν την επεξεργασία της συμβολοσειράς με απροσδιόριστα αποτελέσματα!! (μέχρι να βρουν τον πρώτο χαρακτήρα \0)
- Η τυποποιημένη (standard) βιβλιοθήκη της C περιλαμβάνει μια σειρά συναρτήσεων για συμβολοσειρές (stdio.h, stdlib.h, string.h). Αναφέρονται μερικές στη συνέχεια

Είσοδος / Έξοδος συμβολοσειρών (stdio.h)

- scanf() και printf() για είσοδο και έξοδο, π.χ.

```
#include <stdio.h>
#define MAXLENGTH 15

int main()
{
   char string1[MAXLENGTH];
   char string2[MAXLENGTH];

   scanf("%s %s", string1, string2);
   printf("%s %s\n", string1, string2);

   return 0;
}
```

Διάβασμα χωρίς & !!!

Είσοδος/Έξοδος συμβολοσειρών (stdio.h) (2)

- gets() και puts() για είσοδο και έξοδο, π.χ.

```
#include <stdio.h>
char input[81]; /* A character array to hold input */

main()
{
      puts("Enter some text, then press Enter: ");
      gets(input);
      printf("You entered:");
      puts(input);
      return 0;
}
```

- *Οι συναρτήσεις getchar() και putchar() είναι αντίστοιχες, αλλά αφορούν ένα μεμονωμένο χαρακτήρα*

Συναρτήσεις χειρισμού συμβολοσειρών (string.h)

- strlen(s) → επιστρέφει το μήκος του string s
- strcat(s1, s2) → συνενώνει τα strings s1 και s2 στο s1
- strcpy(s1, s2) → αντιγράφει το string s2 στο s1

```
char string1[MAXLENGTH];
char string2[MAXLENGTH];
strcpy(string1, "Goodbye");
strcpy(string2, ", Cruel ");
strcat(string1, string2);
strcat(string1, "World!");
```

string1: "Goodbye, Cruel World!"
string2: ", Cruel "

Συνάρτηση σύγκρισης συμβολοσειρών (string.h)

- strcmp(s1, s2) → συγκρίνει τα s1 και s2 και επιστρέφει αρνητική τιμή αν s1 < s2, μηδέν αν s1 == s2 και θετική τιμή αν s1 > s2

```
if (strcmp(string1, string2) < 0)
{
  printf("%s %s\n", string1, string2);
}
else
{
  printf("%s %s\n", string2, string1);
}
```

- Εντολή του τύπου
  `if (string1 < string2) ...`
  είναι λάθος

Μετατροπή συμβολοσειράς σε αριθμό (stdlib.h)

- atof() και atoi() για μετατροπή σε float και int, π.χ.

```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
void main()
{  char buf[80];  int k;   double d;
   while (1) {
     printf("\nEnter string - blank to exit):");
     gets(buf);
     if ( strlen(buf) == 0 ) break;
     d = atof( buf );
     k = atoi(buf);
     printf("converted values %f,%d",d,k);
   }
}
```

Description

The C library function **double atof***constchar * str* converts the string argument **str** to a floating-point number *typedouble*.

Declaration

Following is the declaration for atof function.

```
double atof(const char *str)
```

Parameters

- **str** -- This is the string having the representation of a floating-point number.

Return Value

This function returns the converted floating point number as a double value. If no valid conversion could be performed, it returns zero 0.0.

Example

The following example shows the usage of atof function.

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

int main()
{
   float val;
   char str[20];

   strcpy(str, "98993489");
   val = atof(str);
   printf("String value = %s, Float value = %f\n", str, val);

   strcpy(str, "tutorialspoint.com");
   val = atof(str);
   printf("String value = %s, Float value = %f\n", str, val);

   return(0);
}
```

ΣΥΝΑΡΤΗΣΕΙΣ

Συναρτήσεις (functions)

- Μια συνάρτηση είναι ένα ανεξάρτητο τμήμα προγράμματος, που έχει δικό του όνομα, καλείται με ή χωρίς παραμέτρους, εκπληρώνει μια συγκεκριμένη εργασία και μπορεί να επιστρέψει μια τιμή στο πρόγραμμα που την καλεί
- Μια συνάρτηση δηλώνεται ως εξής:

  Επιστρεφόμενος-τύπος όνομα-συνάρτησης (λίστα-παραμέτρων)

  { σώμα-συνάρτησης }
- Αν η συνάρτηση δεν επιστρέφει τιμή, τότε ο επιστρεφόμενος τύπος πρέπει να είναι ο void. Αν δε δηλωθεί τύπος, εννοείται ο τύπος int

  Αν η συνάρτηση δεν παίρνει παραμέτρους, τότε πρέπει η λίστα παραμέτρων να είναι κενή, ή να είναι η λέξη void

Συναρτήσεις (2)

- Αν η συνάρτηση χρησιμοποιείται πριν οριστεί, τότε πρέπει να δοθεί το πρωτότυπό της, ώστε ο μεταγλωττιστής να γνωρίζει για την ύπαρξή της (συνήθως στην αρχή του κώδικα) ως εξής:

  Επιστρεφ.-τύπος όνομα-συνάρτησης (λίστα-τύπων-παραμέτρων)

- *Στα πρωτότυπα, τα ονόματα των παραμέτρων είναι προαιρετικά, σε αντίθεση με τους τύπους τους*

Παράδειγμα συνάρτησης

```
#include <stdio.h>

float div_f(int, int) ; /* function prototype */

int main(void) {
   int val1, val2;
   float answer;
   printf("Enter two integer values:\n");
   scanf("%d %d", &val1, &val2);
   answer = div_f(val1, val2);/* function call */
   printf ("The result is %f", answer);
   return 0;
}

 float div_f(int v1, v2){ /*function declaration*/
   return v1 /(float) v2; /* return value */
 }
```

Μεταβλητές και Παράμετροι συναρτήσεων

- Το *πεδίο* (scope) μιας μεταβλητής (το μέρος του κώδικα όπου μπορεί να χρησιμοποιηθεί) καθορίζεται από το που έχει οριστεί.

- Αν έχει οριστεί εκτός οποιαδήποτε συνάρτησης, τότε έχει ως πεδίο το αρχείο όπου έχει οριστεί και ονομάζεται *καθολική* (global) μεταβλητή (και συνήθως ορίζεται στην αρχή του αρχείου)

- Όλες οι άλλες μεταβλητές λέγονται *τοπικές* (local) μεταβλητές

Μεταβλητές και Παράμετροι συναρτήσεων (2)

- Αν μια μεταβλητή οριστεί μέσα σε μια ενότητα κώδικα που περιβάλλεται από { και }, το πεδίο της ξεκινά από το σημείο της δήλωσής της και τελειώνει στο } που τερματίζει την ενότητα
- Μια μεταβλητή που δηλώνεται ως παράμετρος συνάρτησης έχει ως πεδίο τη συνάρτηση αυτή
- Μια τοπική μεταβλητή δημιουργείται (της εκχωρείται μνήμη) όταν καλείται η συνάρτηση μέσα στην οποία έχει οριστεί και παύει να υπάρχει μετά την επιστροφή της συνάρτησης
- Για να υπάρχει η τοπική μεταβλητή και να διατηρηθεί η τιμή της και μετά το πέρας της συνάρτησης, πρέπει να προηγηθεί της δήλωσής της η λέξη static, π.χ. static int i;

Μεταβλητές και Παράμετροι συναρτήσεων (3)

- Στη C οι παράμετροι συναρτήσεων περνούν **με τιμή**: οι παράμετροι παίρνουν την τιμή της αντίστοιχης μεταβλητής ή σταθεράς με την οποία καλείται η συνάρτηση.

- Μια μεταβλητή που τυχόν χρησιμοποιήθηκε στην κλήση μιας συνάρτησης στη θέση μιας παραμέτρου της συνάρτησης παραμένει ανέπαφη (δε μπορεί να αλλάξει το περιεχόμενό της από τη λειτουργία της συνάρτησης)

Παράδειγμα Μεταβλητών και Παραμέτρων

```
#include <stdio.h>
int m;

void function1(int n, double x);

void main(void) {
   double y ;

   m=15;
   y=308.24;
   printf ("The value of y is %lf\n",y);
   function1(m,y);
   printf ("The value of y is still %lf\n",y);
}

void function1(int n, double x) {
      int i =5;
      x = n * i;
}
```

ΚΑΘΟΛΙΚΗ ΜΕΤΑΒΛΗΤΗ

ΠΡΩΤΟΤΥΠΟ ΣΥΝΑΡΤΗΣΗΣ

ΑΝΤΙΣΤΟΙΧΙΑ ΠΑΡΑΜΕΤΡΩΝ

ΚΛΗΣΗ ΣΥΝΑΡΤΗΣΗΣ

ΠΑΡΑΜΕΤΡΟΙ ΣΥΝΑΡΤΗΣΗΣ

ΔΗΛΩΣΗ ΣΥΝΑΡΤΗΣΗΣ

ΤΟΠΙΚΗ ΜΕΤΑΒΛΗΤΗ

Η τιμή της y παραμένει ανέπαφη από τη λειτουργία της function1

Στοίβα Προγράμματος

- Όταν καλείται μια συνάρτηση, όλες οι πληροφορίες που σχετίζονται με τη συνάρτηση αυτή (τοπικές μεταβλητές, παράμετροι κλήσης, γραμμή κώδικα που πρόκειται να εκτελεστεί μετά την επιστροφή από τη συνάρτηση) τοποθετούνται σε μια εγγραφή ενεργοποίησης (*activation record)*
- Η εγγραφή αυτή ωθείται (pushed) στη στοίβα προγράμματος, μια ειδική περιοχή μνήμης που είναι μια δομή Last-In-First-Out (LIFO)
- Με τον τερματισμό της συνάρτησης, η εγγραφή αυτή εξάγεται (popped) από τη στοίβα

Αναδρομικές κλήσεις συναρτήσεων

- Ορισμός Παραγοντικού:
  0! = 1
  N! = N * (N - 1)!

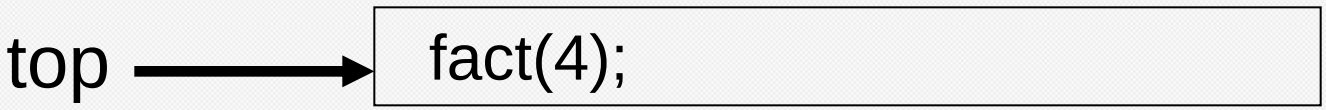
Παρατήρηση: ο ορισμός είναι αναδρομικός.

- Αναδρομική συνάρτηση για τον υπολογισμό του παραγοντικού (καλεί τον εαυτό της)

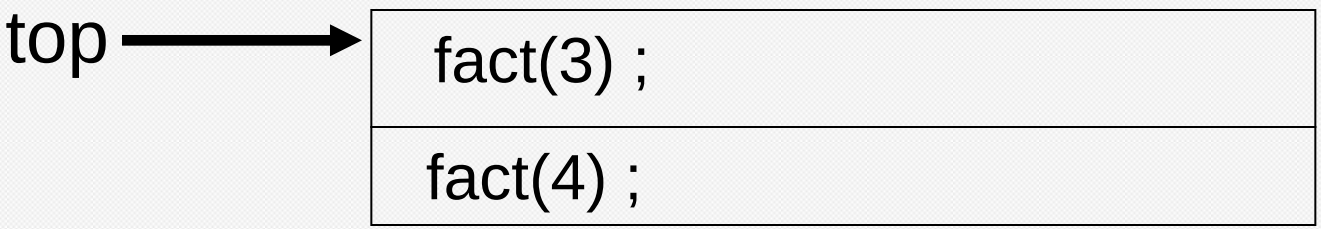
```
int fact(int n)
{ if(n == 0)
     return 1;
  else
     return n * fact(n-1);
}
```

Ιχνηλάτηση στοίβας προγράμματος

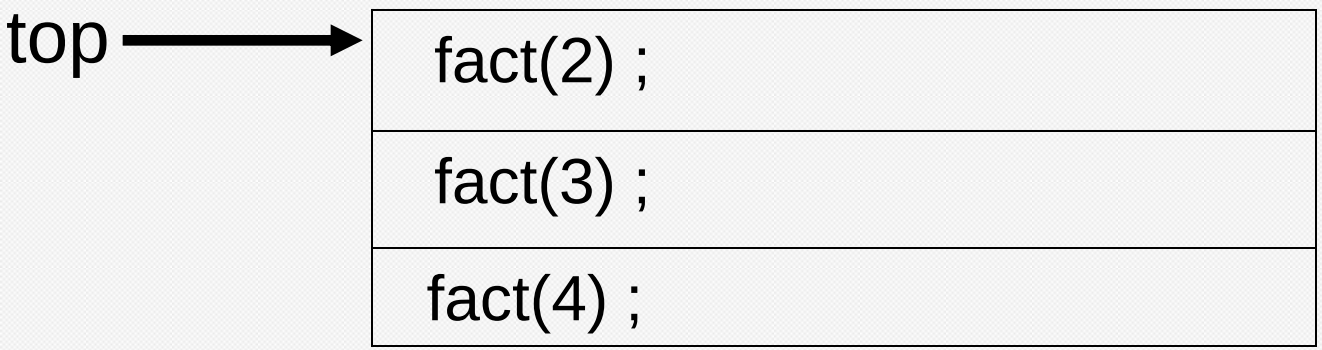
- Καλούμε την fact(4)



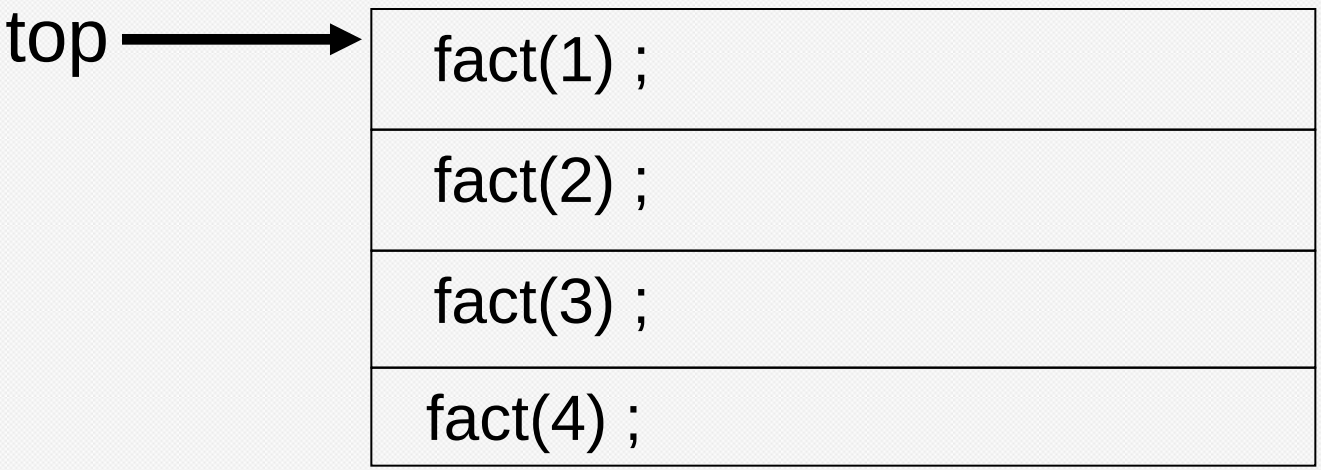
Ιχνηλάτηση στοίβας προγράμματος (2)



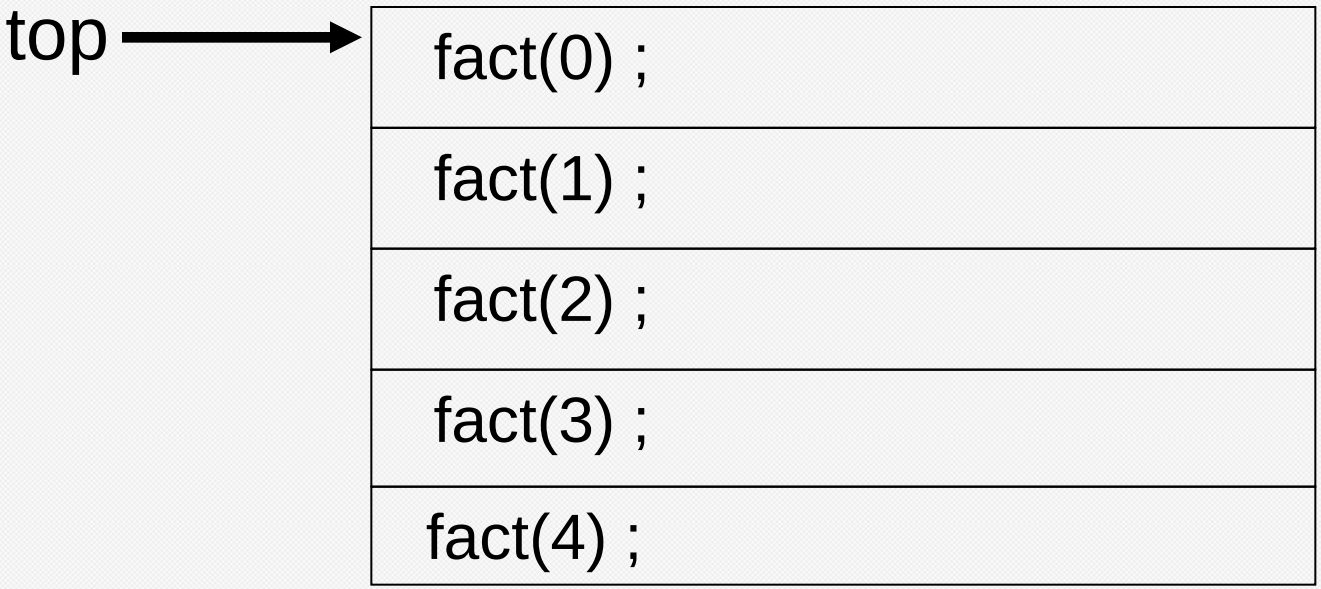
Ιχνηλάτηση στοίβας προγράμματος (3)



Ιχνηλάτηση στοίβας προγράμματος (4)

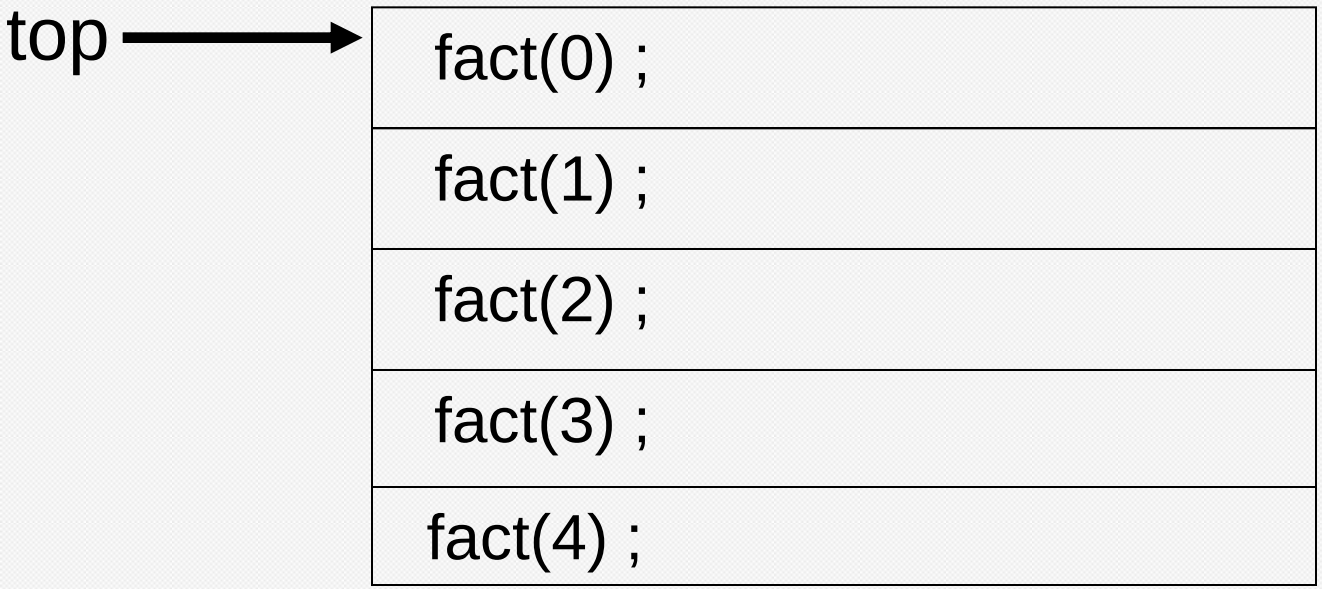


Ιχνηλάτηση στοίβας προγράμματος (5)

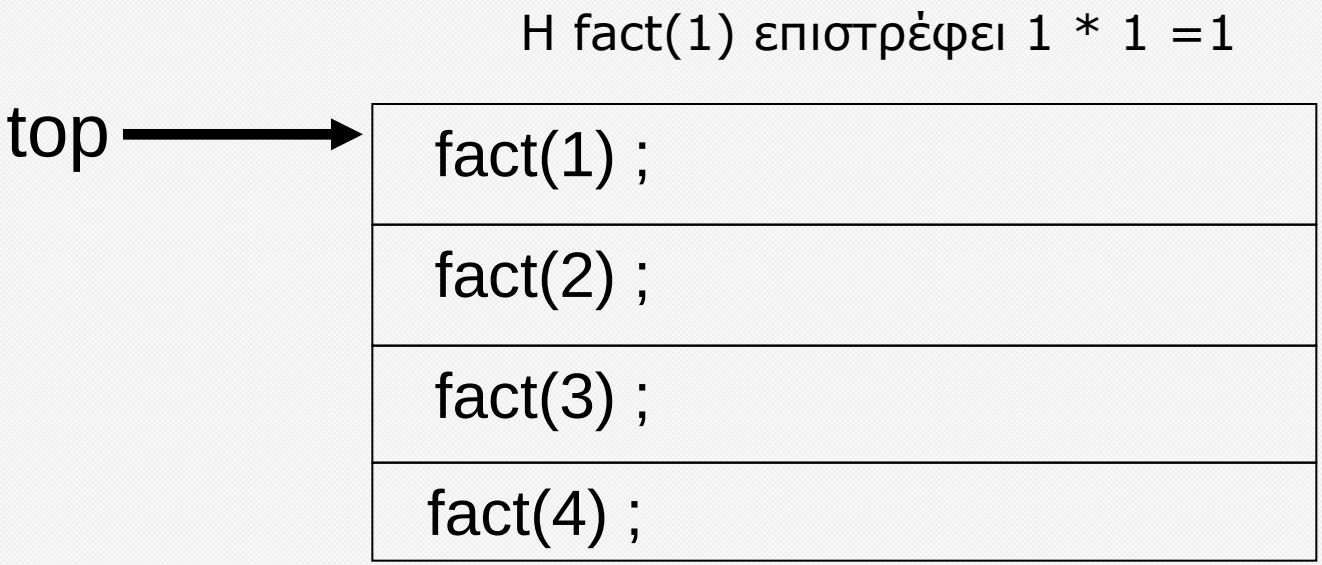


Ιχνηλάτηση στοίβας προγράμματος (6)

Η fact(0) επιστρέφει 1

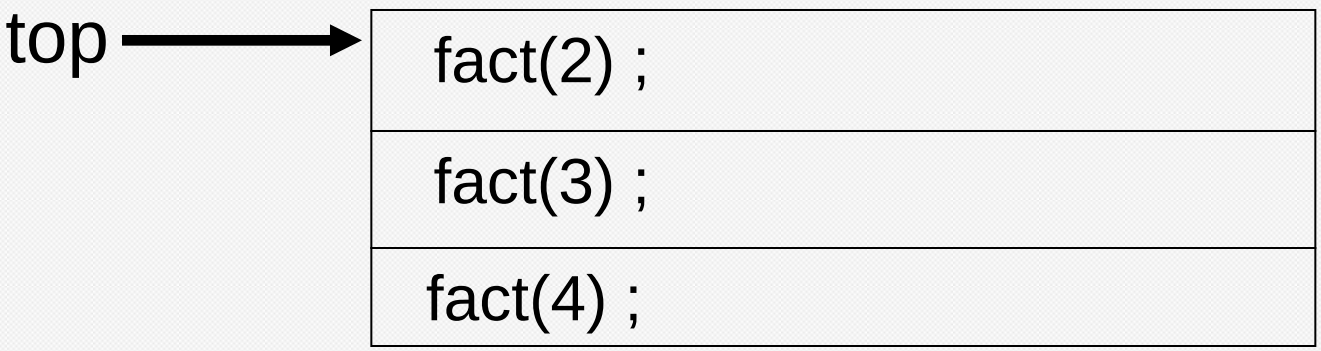


Ιχνηλάτηση στοίβας προγράμματος (7)

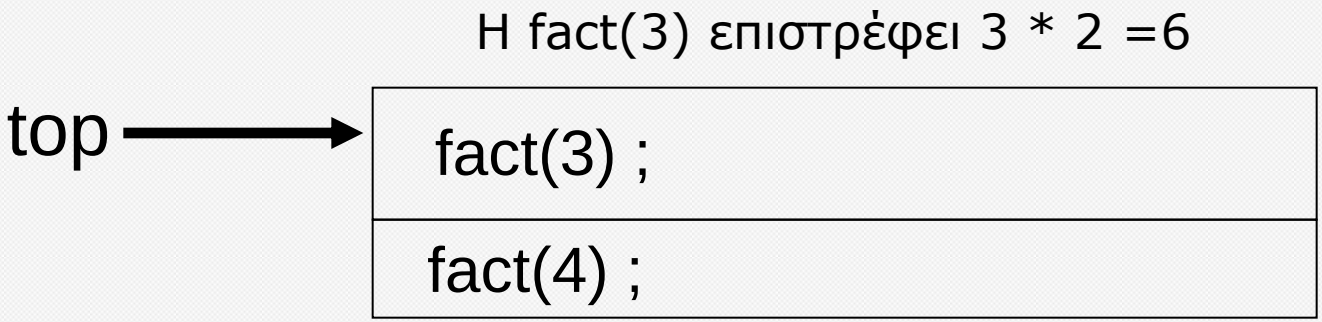


Ιχνηλάτηση στοίβας προγράμματος (8)

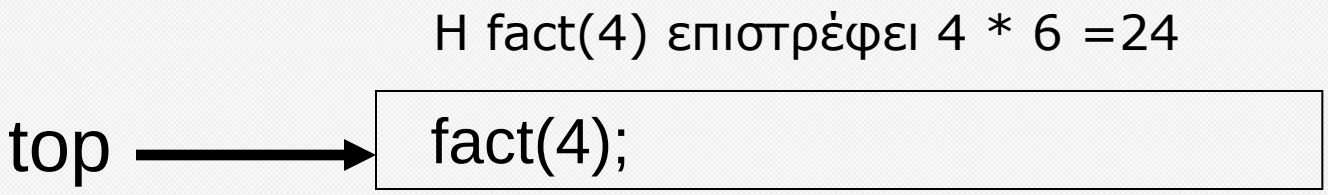
Η fact(2) επιστρέφει 2 * 1 =2



Ιχνηλάτηση στοίβας προγράμματος (9)



Ιχνηλάτηση στοίβας προγράμματος (10)



ΠΟΛΥΔΙΑΣΤΑΤΟΙ ΠΙΝΑΚΕΣ

Πολυδιάστατοι Πίνακες

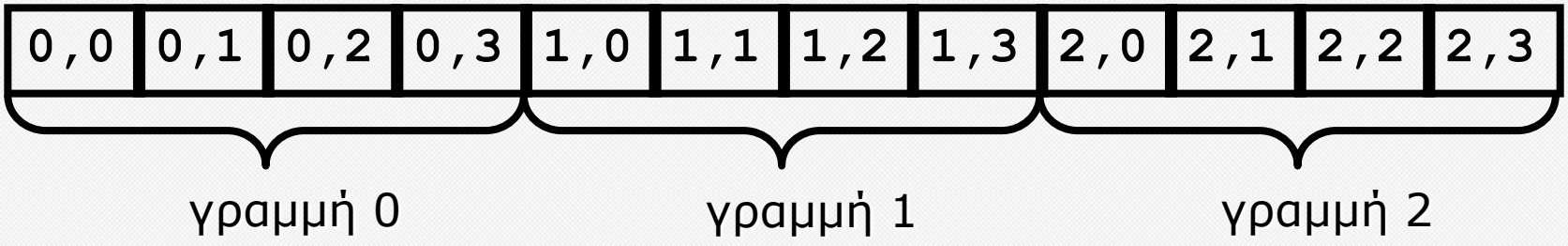
- Στη C μπορούμε να δηλώσουμε και πίνακες πολλών διαστάσεων
- Παράδειγμα δήλωσης πίνακα 2 διαστάσεων:

```
int a[3][4];
```

Γραμμές 0..2 **Στήλες 0..3**

- Παράδειγμα προσπέλασης στοιχείου πίνακα 2 διαστάσεων:
  a[1][0] = 5; /* στη γραμμή 1 και στήλη 0 αποθήκευσε το 5 */

- Στη C η αποθήκευση των στοιχείων πολυδιάστατου πίνακα γίνεται ανά γραμμή, π.χ.



ΔΕΙΚΤΕΣ

Δείκτες

• Οι Δείκτες είναι βασικό κομμάτι της C, καθώς παρέχουν μεγάλη ευελιξία (όμως με επιπόλαια χρήση, δημιουργούν λάθη εκτέλεσης)

• Χρησιμοποιούνται με πίνακες, δομές και συναρτήσεις

• Ένας δείκτης είναι μια μεταβλητή που περιέχει
τη διεύθυνση μνήμης μιας άλλης μεταβλητής

• Ο μοναδιαίος τελεστής & εφαρμοζόμενος σε μια μεταβλητή δίνει τη διεύθυνση μιας μεταβλητής

• Ο τελεστής αποδεικτοδότησης * εφαρμοζόμενος σε ένα δείκτη δίνει τη δεικνυόμενη μεταβλητή

Μεταβλητές και Δείκτες

Παραδείγματα

- Δήλωση (π.χ. ακέραιας) μεταβλητής, π.χ.: **`int k;`**
- Δήλωση δείκτη προς μεταβλητή, π.χ: **`int *ptr;`**
- Αρχικοποίηση δείκτη, π.χ.: **`ptr = NULL;`**
- Διεύθυνση μεταβλητής, συμβολίζεται με: **`&k`**
- Ανάθεση δείκτη, π.χ.: **`prt = &k;`**
- Η μεταβλητή συμβολίζεται με: **`k`** ή **`*ptr`**
  Π.χ.: **`k = 5;`**
  ή
  `*ptr = 5;`

Το * εφαμόζεται στον τύπο int

Το * εφαμόζεται στο δείκτη

Ίδιος συμβολισμός, άλλο νόημα

Επισημάνσεις για τους Δείκτες

Προσοχή: Ένας δείκτης

- σχετίζεται με τον τύπο της μεταβλητής που δείχνει, καθώς η κάθε μεταβλητή καταλαμβάνει, ανάλογα με τον τύπο της, διαφορετική ποσότητα μνήμης
- μέχρι να του αποδοθεί μια τιμή (διεύθυνση) δεν είναι χρήσιμος και η προσπέλαση του δεικνυόμενου στοιχείου συνήθως δημιουργεί Λάθος Μνήμης κατά την εκτέλεση

Προσοχή σε δείκτες που δεν έχουν αρχικοποιηθεί

Όταν ένας δείκτης έχει τιμή NULL δεν επιτρέπεται η προσπέλαση του δεικνυόμενου στοιχείου

Ας συζητήσουμε τι συμβαίνει στον κώδικα που ακολουθεί:

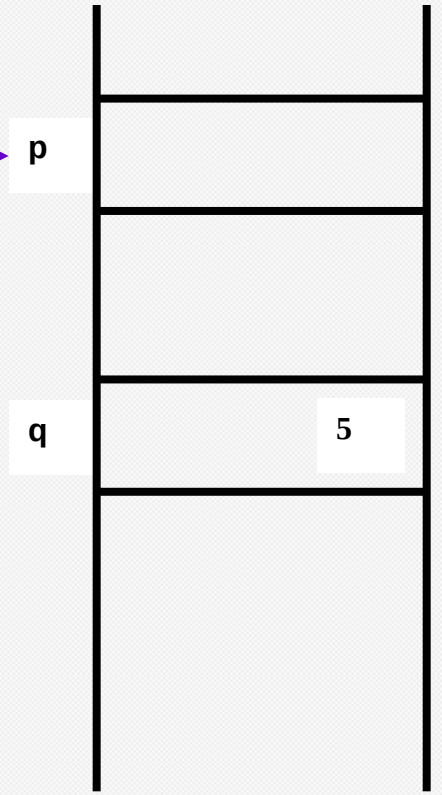
```
int *p=NULL;
int q=5;
p=&q;
*p=6;
```

Δήλωση και Αρχικοποίηση δείκτη με κενή (NULL) τιμή

Δήλωση και Αρχικοποίηση μεταβλητής int

Ο δείκτης «δείχνει» πλέον στη μεταβλητή

Η θέση μνήμης που «δείχνει» ο δείκτης (δηλαδή η μεταβλητή) παίρνει την τιμή 6



Δυναμική Παραχώρηση Μνήνης

Συναρτήσεις βιβλιοθήκης (<stdlib.h>)

- **`void * malloc (size_t n);`**

  Δυναμική παραχώρηση μνήμης **`n`** bytes.
  Ο επιστρεφόμενος δείκτης σε **`void`** είναι γενικός τύπος δείκτη και πρέπει να μετατραπεί στον κατάλληλο τύπο.
  Επιστρέφεται **`NULL`** αν εξαντληθεί η μνήμη του συστήματος

- **`void free (void * p);`**

  Απελευθέρωση της μνήμης στην οποία δείχνει ο **`p`** (που πρέπει να έχει παραχωρηθεί με προηγούμενη κλήση της **`malloc`**)

Υπολογισμός απαιτούμενων bytes:

`sizeof(`*`type`***`)`** Π.χ. **`sizeof(int)`**

`sizeof(`*`variable`***`)`** Π.χ. **`sizeof(x)`**

Ας συζητήσουμε τι συμβαίνει στον κώδικα που ακολουθεί:

Δήλωση δείκτη P χωρίς Αρχικοποίηση

```
int *P;
```

Ο P «δείχνει» σε περιοχή μνήμης μεγέθους 1 ακεραίου που παραχωρήθηκε δυναμικά


```
P =(int *)malloc(sizeof(int));
```

Στην περιοχή μνήμης που «δείχνει» ο P αποθηκεύεται η τιμή 10

```
*P = 10;
```

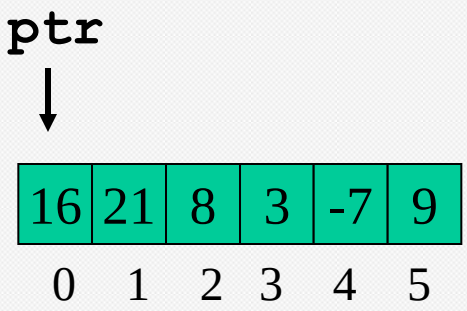
Απελευθερώνεται η περιοχή μνήμης όπου «δείχνει» ο P

```
free(P);
```

Δείκτες και Πίνακες

Μια μεταβλητή πίνακα είναι ένας **σταθερός** δείκτης προς το πρώτο στοιχείο του πίνακα

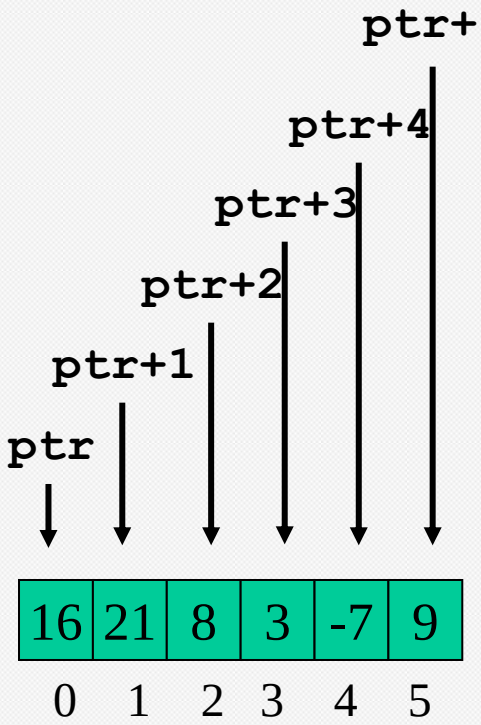
```
int table[6] = {16,21,8,3,-7,9};
int *ptr;
ptr = &table[0];
```



Το να γράψουμε **`table`** είναι το ίδιο με το να γράψουμε **`&table[0]`** ή **`ptr`**.

Αριθμητική Δεικτών

Μπορούμε να προσθέσουμε μια ακέραια τιμή σε ένα δείκτη, π.χ. το **`ptr=ptr+1`** ή **`ptr++`** κάνει το δείκτη να δείξει το επόμενο στοιχείο του πίνακα table (δηλ. στον επόμενο int)




```
for(ptr=table; ptr!=table+6; ptr++)
  printf("%4d", *ptr);
```

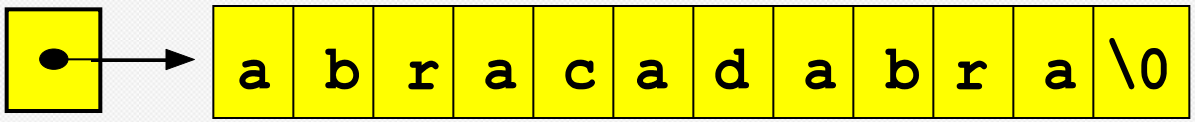
Μπορούμε να αφαιρέσουμε ή να συγκρίνουμε δείκτες, π.χ.

```
int *P1, *P2;
P1 = &table[5]; P2 = &table[0];
```

- Το **`P1-P2`** αναπαριστά τη διαφορά θέσεων μεταξύ στοιχείων του πίνακα (5 θέσεις)
- Το **`P1 == P2`** (ή **`P1 != P2`**) εκφράζει αν οι **`P1`** και **`P2`** (δεν) δείχνουν την ίδια θέση μνήμης

Προσοχή: η αριθμητική δεικτών δεν εφαρμόζεται σε μεταβλητές πίνακα

Συμβολοσειρές και Δείκτες




```
#include <stdio.h>

void main(void)
{ const char nul = 0;
  char *magic = "abracadabra";
  char *ptr;

  for (ptr = magic; *ptr != nul; ptr++)
    printf("%c", *ptr);
}
```

Δυναμική δημιουργία πίνακα μέσω δείκτη

```
int *p;
```

Ορίζουμε δείκτη προς τον επιθυμητό τύπο στοιχείων

Παραχωρούμε στο δείκτη την απαιτούμενη ποσότητα μνήμης (ανάλογα με τον αριθμό των στοιχείων που επιθυμούμε)

```
p = (int *) malloc(100 * sizeof(int));

if (p == NULL)
   ...
```

Ελέγχουμε αν δεν ήταν δυνατή η παραχώρηση της μνήμης

```
for (i = 0; i < 100; i++)
   p[i] = ...
```

Προσπελάζουμε τα στοιχεία του πίνακα

```
free(p);
```

Απελευθερώνουμε την παραχωρηθείσα μνήμη, όταν δε χρειαζόμαστε πλέον τον πίνακα

Συναρτήσεις και πέρασμα με αναφορά

- Στη C οι παράμετροι συναρτήσεων περνούν **με τιμή** (call-by-value): μια παράμετρος παίρνει την τιμή της αντίστοιχης μεταβλητής με την οποία καλείται η συνάρτηση (οπότε, η μεταβλητή αυτή παραμένει ανέπαφη από την κλήση της συνάρτησης)
- Αν όμως περάσουμε στη συνάρτηση ως παράμετρο τη διεύθυνση μνήμης μιας μεταβλητής (δηλαδή ένα δείκτη προς τη μεταβλητή), ο δείκτης προς τη μεταβλητή περνά με τιμή και παραμένει ανέπαφος, αλλά μπορούμε να τροποποιήσουμε τη δεικνυόμενη μεταβλητή
- Έτσι, επιτυγχάνουμε πέρασμα παραμέτρου **με αναφορά** (call-by-reference)

Συναρτήσεις και πέρασμα με αναφορά (2)

Πέρασμα με τιμή

```
#include <stdio.h>

void AddTwo (int v)
{
      v += 2;
}

void main( )
{
      int val = 4;
      AddTwo ( val );
      printf("%d\n",val);
}
```

Η τιμή της val αντιγράφεται στην v και η val παραμένει ανέπαφη

Πέρασμα με αναφορά

```
#include <stdio.h>

void AddTwo (int *ref_v)
{
      *ref_v += 2;
}

void main( )
{
      int val = 4;
      AddTwo ( &val );
      printf("%d\n",val);
}
```

Η διεύθυνση της val αντιγράφεται στην ref_v, η οποία τροποποιεί την val (την *ref_v)

Δείκτης/Πίνακας ως Παράμετρος

```
/* ορισμός συνάρτησης με παράμετρο πίνακα, ή δείκτη ακεραίων */
void bubblesort (int a[], int r) /* πίνακας a[0..r-1] */
{
  int i, x;                                   ή int *a
  int at_least_one_swap = 1;

  /* ενώ στο προηγ.πέρασμα έγινε μία τουλ. εναλλαγή */
  while (at_least_one_swap) {
    at_least_one_swap = 0;
    for (i = 0; i < r-1; i ++) {
      if (a [i] > a [i + 1]) {  /* αν τα γειτονικά στοιχεία */
                                /* δεν τηρούν τη διάταξη ...*/
        SwapInt(a[i],a[i+1],x); /* ενάλλαξε τη θέση τους    */
        at_least_one_swap = 1;  /* και ετοιμάσου για άλλο...*/
      }                         /* ένα πέρασμα              */
    }
    r --; /* το δεξί άκρο περιέχει το σωστό στοιχείο */
  }
}
```

Ο κώδικας της bubblesort όπως τον έχουμε ξαναδεί χωρίς παραμέτρους κλήσης

Δείκτης/Πίνακας ως Παράμετρος (2)

```
#include <stdio.h>

/* μακροεντολή για εναλλαγή δύο αριθμών */
#define SwapInt(x,y,tmp) {tmp=x;x=y;y=tmp;}

int b[5] = {3,2,1,7,4}; /* ορισμός global πίνακα ακεραίων */

void bubblesort (int a[], int r); /* πρωτότυπο bubblesort */

int main() {
  int i;

  printf("arxikos pinakas\n");
  for (i=0; i<5; i++)
     printf("%d\n",b[i]);

  bubblesort (b, 5); /* κλήση bubblesort για συγκεκριμένο πίνακα */
  printf("diatetagmenos pinakas\n");
  for (i=0; i<5; i++)
     printf("%d\n",b[i]);
}

/* ορισμός συνάρτησης με παράμετρο πίνακα, ή δείκτη ακεραίων */
void bubblesort ( ...
```

Περνά στην bubblesort δείκτης προς το πρώτο στοιχείο του πίνακα b

Τα στοιχεία του πίνακα b έχουν τροποποιηθεί από την bubblesort (ο b έχει ταξινομηθεί)

ΔΟΜΕΣ

Δομές (structures)

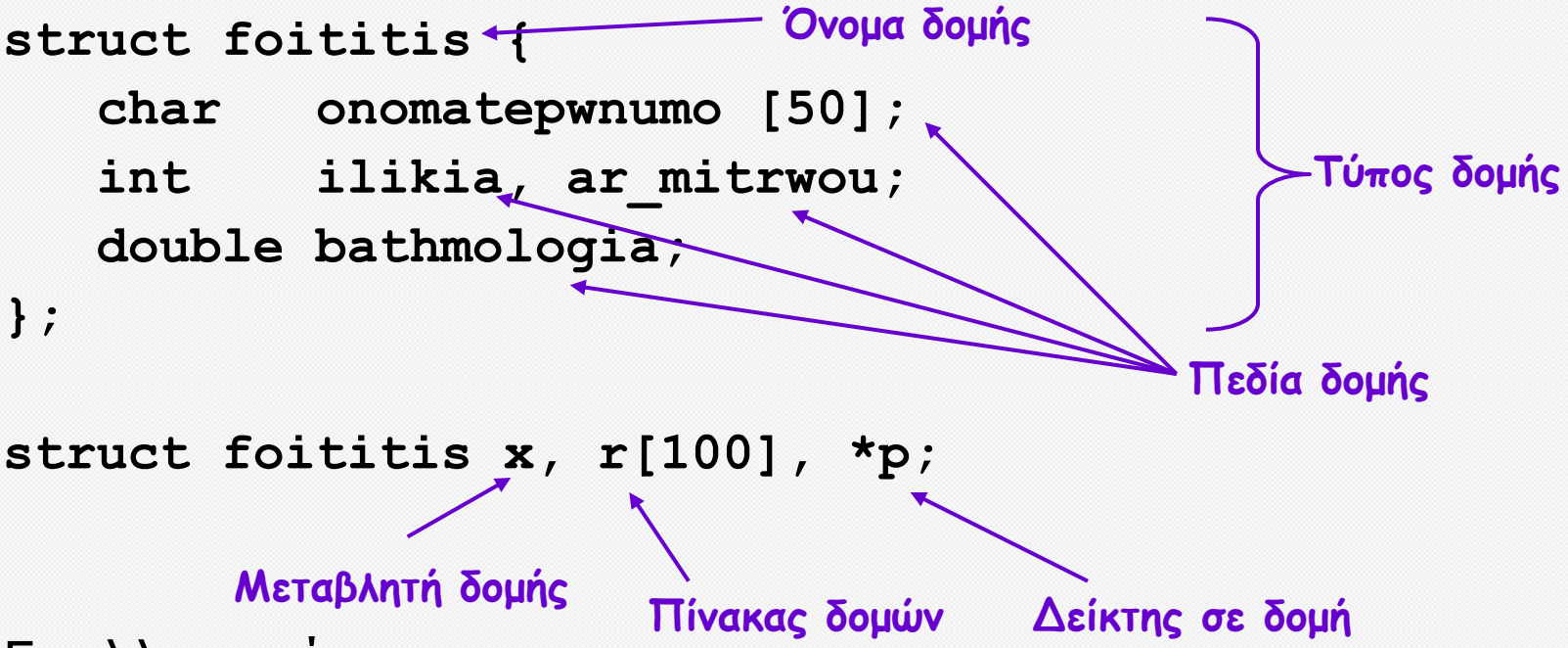
- Μέχρι στιγμής, έχουμε χρησιμοποιήσει δεδομένα με απλούς τύπους (χαρακτήρες, ακεραίους, πραγματικούς) και πίνακες, όπου όλα τα στοιχεία τους είναι δεδομένα του ίδιου τύπου (πίνακες ακεραίων, χαρακτήρων, κλπ)

- Οι δομές στη C μας δίνουν τη δυνατότητα να ορίσουμε ομάδες δεδομένων που έχουν λογική σχέση μεταξύ τους, αλλά μπορούν να ανήκουν σε διαφορετικούς τύπους

Δομές (2)

- Η δομή έχει ως αντίστοιχη έννοια στην καθημερινότητα την εγγραφή, π.χ. μια δομή μπορεί να χρησιμοποιηθεί για να αναπαραστήσει την εγγραφή ενός φοιτητή, που περιλαμβάνει το ονοματεπώνυμό του (μια συμβολοσειρά), τον αριθμό μητρώου του (έναν ακέραιο), το μέσο όρο βαθμολογίας του (έναν πραγματικό)
- Τα μέλη μιας δομής μπορεί να ανήκουν στους απλούς τύπους, να είναι πίνακες, ή άλλες δομές
- Μπορούμε να ορίσουμε δείκτες σε δομές και πίνακες με στοιχεία δομές
- Μια συνάρτηση μπορεί να επιστρέφει μια δομή

Δήλωση τύπου και μεταβλητών Δομής



Εναλλακτικά

```
struct {
   char   onomatepwnumo [50];
   int    ilikia, ar_mitrwou;
   double bathmologia;
} x, r[100], *p;
```

Αρχικοποίηση και Προσπέλαση Δομής

```
struct point {
   double x, y;
};

struct point p1 = {0.5, 0.4}, p2, *ppoint;

prinft("%f, %f\n",p1.x, p1.y);

ppoint = &p1;
prinft("%f, %f\n",ppoint->x, (*ppoint).y);

ppoint = &p2;
ppoint->x = 7.0;
(*ppoint).y = 6.0;
```

Αρχικοποίηση δομής

Μη αρχικοποιημένη δομή

Δείκτης σε δομή

Δομή.Πεδίο

Ο ppoint δείχνει στην p1

Δείκτης->Πεδίο

ή (*Δείκτης).Πεδίο

Εμφωλευμένες Δομές

```
struct point { /* δομή για ένα σημείο */
   double x, y;
};

 struct rect {   /* δομή για ένα παραλληλόγραμμο
                    που ορίζεται από 2 αντιδιαμετρικές
                    κορυφές του */
    struct point pt1;
    struct point pt2;
};

struct rect screen;
```

Το **screen.pt1.x** αναφέρεται στη x συντεταγμένη του σημείου pt1 της δομής screen

Δομές και Συναρτήσεις

```
/* συνάρτηση που επιστρέφει δομή point */
struct point makepoint(double x, double y)
{
    struct point temp;
    temp.x = x;
    temp.y = y;
    return temp;
}

/* ορισμός παραλληλογράμμου με κορυφές (0,0) και (10,10)*/
screen.pt1 = makepoint(0.0, 0.0);
screen.pt2 = makepoint(10.0, 10.0);

if (screen.pt1 != screen.pt2) ...
```

Προσοχή: οι δομές δε μπορούν να συγκριθούν (μόνο τα επιμέρους πεδία τους)

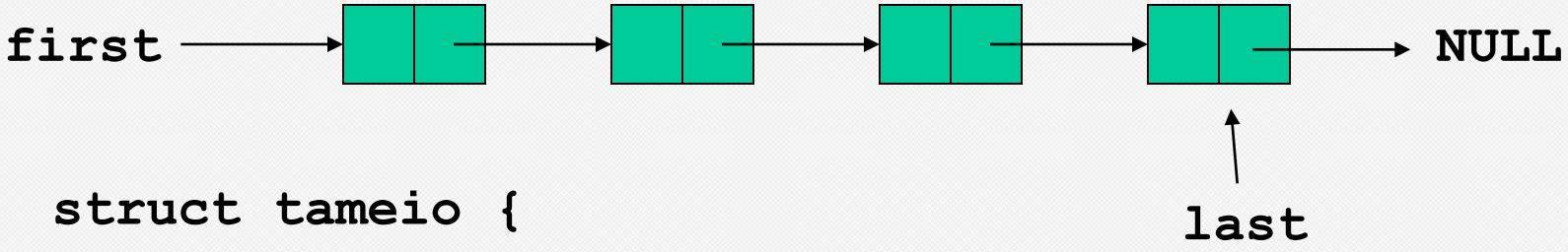
Πίνακες Δομών

```
struct key {
        char *word;
        int count;
} keytab[] = {
        {"break", 0},
        {"case", 0},
        {"char", 0},
        {"const", 0},
        {"continue", 0},
        {"default", 0},
        {"unsigned", 0},
        {"void", 0},
        {"while", 0}
  };
```

Πίνακας δομών με δεσμευμένες λέξεις της C και το πλήθος εμφάνισής τους. Το μέγεθος του πίνακα keytab προκύπτει από το πλήθος των στοιχείων αρχικοποίησης (keytab[9])

Δυναμικές (συνδεδεμένες) Δομές

- Πρόκειται για δομές που περιέχουν δείκτες προς άλλες δομές
- Μας επιτρέπουν να χειριστούμε δεδομένα όταν μας είναι άγνωστο το πλήθος τους (με δυναμική παραχώρηση μνήμης)
- Π.χ. μπορούμε να αναπαραστήσουμε τους πελάτες σε ένα ταμείο με μια συνδεδεμένη λίστα, στην οποία να προσθέτουμε (αφαιρούμε) κόμβους, καθώς προστίθενται (αποχωρούν) πελάτες

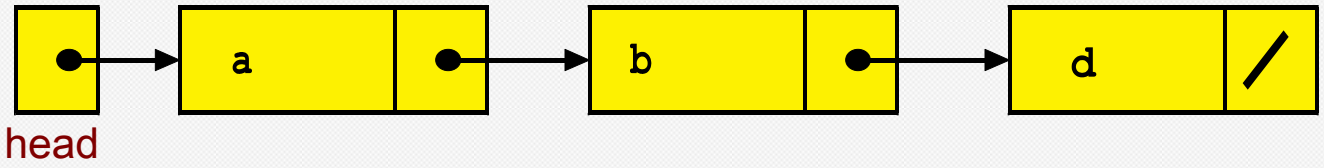



```
struct tameio {
   char pelatis[20];
   struct tameio *next;
};

struct tameio *first, *last;
```

Συνδεδεμένες Λίστες

Ακολουθίες κόμβων όπου κάθε κόμβος έχει δείκτη προς τον επόμενο




```
typedef struct node *NODEPOINTER;

typedef struct node {
        char data;
        NODEPOINTER next;
} NODE;
```

ή

```
typedef struct node {
        char data;
        struct node *next;
} NODE;

typedef NODE *NODEPOINTER;
```

Δείκτης σε κόμβο:
`NODEPOINTER head;`

Προσπέλαση στοιχείου κόμβου:
`head -> data` ή **`(*head).data`**

Συνδεδεμένες Λίστες (συνέχεια)

Αρχικοποίηση λίστας:

```
head = NULL;
```

Εισαγωγή στην αρχή:

```
NODEPOINTER insert(NODEPOINTER h, char l)  {
   NODEPOINTER temp;

   temp = (NODEPOINTER) malloc(sizeof(NODE));
   if (temp == NULL) {
      fprintf(stderr,"ERROR OF MEMORY ALLOCATION\n");
      exit(1);
   }
   temp ->data = l;
   temp ->next = h;
   return(temp);
}
```

Κλήση insert:

```
head = insert(head, 'a');
```

```
STACK example
/*Linked-List Implementation*/
head = NULL;
head = insert(head, 'd');
head = insert(head, 'b');
head = insert(head, 'a');

/*Το head είναι ο δείκτης top
που δείχνει το κορυφαίο
στοιχείο που θα γίνει pop. Εδώ
το insert είναι το push */
```

Συνδεδεμένες Λίστες (συνέχεια)

Διάσχιση και εμφάνιση:

```
void display(NODEPOINTER h)
{
    while (h != NULL) {
        printf("LETTER: %c\n", h->data);
        h = h->next;
    }
}
```

Μερικές άλλες Δυναμικές Δομές

- Στοίβες: LIFO (last-in-first-out)
- Ουρές: FIFO (first-in-first-out)
- Διπλά συνδεδεμένες λίστες
- Δυαδικά Δέντρα
- Γράφοι

Υλοποίηση ΣΤΟΙΒΑΣ με χρήση ΔΟΜΗΣ STRUCTκαι Πίνακα (ARRAY)

```
#include <stdio.h>
#define MAXSIZE 5
struct stack
{
   int stk[MAXSIZE];
   int top;
};
typedef struct stack STACK;
STACK s;

void push(void);
int  pop(void);
void display(void);

void main ()
{
   int choice;
   int option = 1;
   s.top = -1;
```

```
printf ("STACK OPERATION\n");
   while (option)
   {
      printf ("----------------------------------------\n");
      printf ("     1   -->   PUSH            \n");
      printf ("     2   -->   POP             \n");
      printf ("     3   -->   DISPLAY             \n");
      printf ("     4   -->   EXIT          \n");
      printf ("----------------------------------------\n");
      printf ("Enter your choice\n");
      scanf   ("%d", &choice);
      switch (choice)
      {
      case 1:
         push();
         break;
      case 2:
         pop();
         break;
      case 3:
         display();
         break;
      case 4:
         return;
      }
      fflush (stdin);
      printf ("Do you want to continue(Type 0 or 1)?\n");
      scanf   ("%d", &option);
   }
}
```

```
/*  Function to add an element to the stack */
void push ()
{
    int num;
    if (s.top == (MAXSIZE - 1))
    {
        printf ("Stack is Full\n");
        return;
    }
    else
    {
        printf ("Enter the element to be
pushed\n");
        scanf ("%d", &num);
        s.top = s.top + 1;
        s.stk[s.top] = num;
    }
    return;
}
```

```
/*  Function to delete an element from the stack */
int pop ()
{
    int num;
    if (s.top == - 1)
    {
        printf ("Stack is Empty\n");
        return (s.top);
    }
    else
    {
        num = s.stk[s.top];
        printf ("poped element is = %dn",
s.stk[s.top]);
        s.top = s.top - 1;
    }
    return(num);
}
```

```
/*  Function to display the status of the stack */
void display ()
{
    int i;
    if (s.top == -1)
    {
        printf ("Stack is empty\n");
        return;
    }
    else
    {
        printf ("\n The status of the stack is \n");
        for (i = s.top; i >= 0; i--)
        {
            printf ("%d\n", s.stk[i]);
        }
    }
    printf ("\n");
}
```

ΤΕΛΟΣ