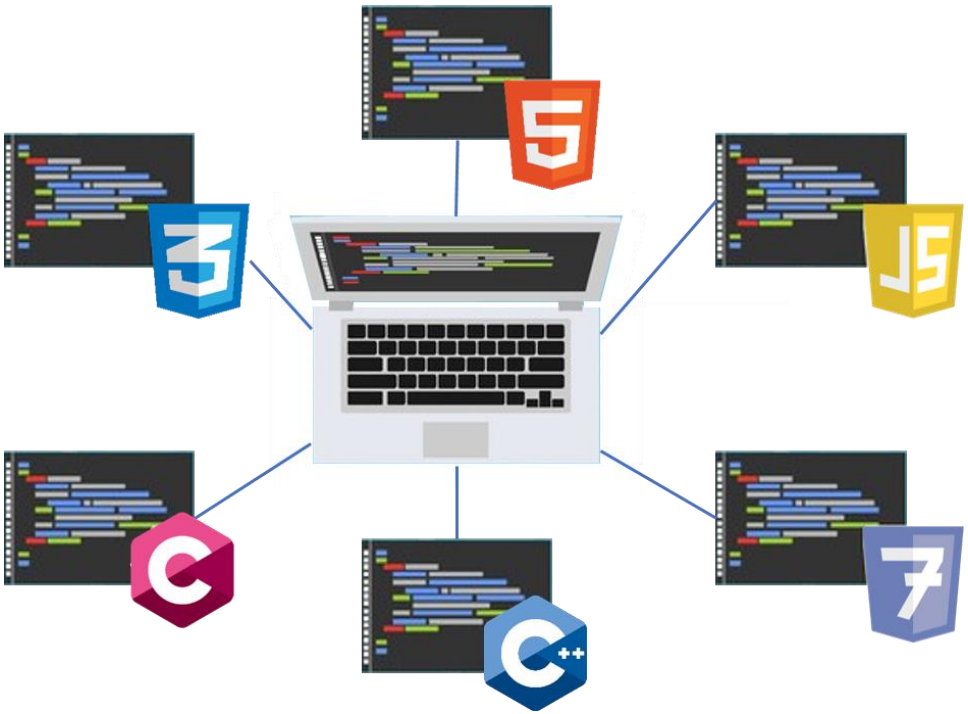
Εργαστήριο Γλώσσας Προγραμματισμού



Η γλώσσα προγραμματισμού C

Εισαγωγή στην C

- Η **C** είναι μια γλώσσα προγραμματισμού γενικής χρήσης
  - Είναι μια προστακτική γλώσσα που βασίζεται σε συναρτήσεις
  - Υποστηρίζει δομημένο προγραμματισμό
  - Έχει στατικούς τύπους
  - Μεταφράζεται σε γλώσσα μηχανής
  - Υποστηρίζει πρόσβαση χαμηλού επιπέδου σε μνήμη και υλικό
- Η C είναι μια από τις πιο συχνά χρησιμοποιούμενες γλώσσες προγραμματισμού
  - Υποστηρίζεται σε όλες τις πλατφόρμες και τα λειτουργικά συστήματα
  - Παράγει πολύ αποδοτικό κώδικα
    - Η ταχύτητα εκτέλεσης είναι πολύ πιο γρήγορη από το να τρέχαμε αντίστοιχους υπολογισμούς σε Python, Matlab, Mapple, Mathematica,κλπ.
  - Παραδείγματα χρήσης της C
    - Λειτουργικά συστήματα
    - Μεταγλωττιστές
    - Κειμενογράφοι
    - Βάσεις δεδομένων

Ένα απλό πρόγραμμα C (1/3)

- Ένα πρόγραμμα C είναι ένα αρχείο κειμένου που το αποθηκεύουμε με κατάληξη **.c**
- Ο κώδικας αποτελείται κυρίως από ορισμούς τύπων, μεταβλητών και συναρτήσεων
  - Σε αντίθεση με τις JavaScript και PHP όπου ο κώδικας μπορεί να έχει απευθείας και εντολές – statements
- Το πρόγραμμα ξεκινάει από την ειδική συνάρτηση ***main***, εκτελώντας τις εντολές που περιέχει
  - Μέσα από τη main μπορεί να κληθεί οποιαδήποτε (ορατή) συνάρτηση του προγράμματος
    - Και κάθε συνάρτηση μπορεί και αυτή με τη σειρά της να καλεί οποιαδήποτε άλλη (ορατή) συνάρτηση
  - Πρέπει να υπάρχει πάντα main

```c
#include <stdio.h>

int main() {
    printf("Hello world\n");
    return 0;
}
```

Ένα απλό πρόγραμμα C (2/3)

- Το **#include** χρησιμοποιείται για να συμπεριλάβουμε ένα ολόκληρο άλλο αρχείο στο τρέχον αρχείο κώδικα ώστε να χρησιμοποιήσουμε τη λειτουργικότητά του
  - Όμοια με το include statement που είδαμε στην PHP
- Στο παράδειγμα κάνουμε #include το αρχείο stdio.h που είναι η βιβλιοθήκη της C για είσοδο και έξοδο (standard I/O)
  - Σκεφτείτε το αντίστοιχα με το import της Python
- Η δήλωση συνάρτησης δεν χρειάζεται κάποια λέξη-κλειδί, ορίζουμε τον τύπο της συνάρτησης, το όνομά της και τα ορίσματα που αυτή δέχεται
- Μέσα στη συνάρτηση main χρησιμοποιούμε τη συνάρτηση **printf** για να τυπώσουμε ένα μήνυμα στην κονσόλα
  - Το μήνυμα είναι το string “Hello world” ακολουθούμενο από αλλαγή γραμμής \n
- Η main επιστρέφει μια ακέραια τιμή (τύπου **int**)
  - Επιστρέφουμε 0 από το **return** statement σηματοδοτώντας στο λειτουργικό σύστημα ότι το πρόγραμμα εκτελέστηκε επιτυχώς
  - Διαφορετικά επιστρέφουμε μη μηδενική τιμή

```c
#include <stdio.h>

int main() {
    printf("Hello world\n");
    return 0;
}
```

Ένα απλό πρόγραμμα C (3/3)

- Βήματα για εκτέλεση ενός προγράμματος C
  - Γράφουμε τον κώδικα με κάποιο κειμενογράφο και τον αποθηκεύσουμε σε αρχείο με κατάληξη .c, π.χ. **test.c**
  - Χρησιμοποιούμε ένα μεταγλωττιστή (compiler) για να μεταγλωττίσουμε το αρχείο κώδικα σε ένα εκτελέσιμο αρχείο που περιέχει κώδικα μηχανής
    - Στα linux της σχολής υπάρχει ο compiler gcc τον οποίο τρέχουμε ως εξής: **gcc test.c**
    - Έτσι παράγεται το εκτελέσιμο αρχείο **a.out**
  - Τρέχουμε το εκτελέσιμο ως εξής: **./a.out**
    - Σε αντίθεση με την python δεν τρέχουμε κάποιον interpreter με το πρόγραμμά μας, αλλά απευθείας το εκτελέσιμο
    - Για την εκτέλεση αρκεί λοιπόν το εκτελέσιμο και δε χρειάζεται να δώσουμε τον πηγαίο κώδικα του προγράμματός μας

```
lilis@test-vm: ~/test
File  Edit  View  Search  Terminal  Help
lilis@test-vm:~/test$ more test.c
#include <stdio.h>

int main() {
        printf("Hello world!\n");
        return 0;
}
lilis@test-vm:~/test$ ls
test.c
lilis@test-vm:~/test$ gcc test.c
lilis@test-vm:~/test$ ls
a.out  test.c
lilis@test-vm:~/test$ ./a.out
Hello world!
lilis@test-vm:~/test$
```

Βασικό συντακτικό C (1/6)

- Συντακτικά (αλλά και σημασιολογικά) η C έχει αρκετές ομοιότητες με τις JavaScript και PHP
  - Η C προϋπήρχε και αυτές ακολούθησαν παρόμοιο συντακτικό
- Και πάλι βασιζόμαστε σε statements και συναρτήσεις που συντίθενται χρησιμοποιώντας
  - Τιμές (π.χ. αριθμοί, strings)
  - Τελεστές (π.χ. αριθμητικοί τελεστές, εκχώρηση)
  - Εκφράσεις (συνδυασμός τιμών και άλλων εκφράσεων με τελεστές, κλήσεις συναρτήσεων, κλπ.)
  - Λέξεις κλειδιά (π.χ. if, else, for)
- Οι μεταβλητές έχουν **στατικούς τύπους**
  - Υποστηρίζονται βασικοί τύποι δεδομένων (π.χ. αριθμοί, χαρακτήρες) καθώς και πίνακες και δομές δεδομένων (structs)
- Τα whitespaces (κενά, newlines, tabs) αγνοούνται
- Μπορούμε επίσης να έχουμε σχόλια - comments
  - Block comments `/* block comments */`
  - Line comments `// line comments`

Βασικό συντακτικό C (2/6)

- Βασικοί τύποι δεδομένων
  - Ακέραιοι - **int**
  - Πραγματικοί απλής ακρίβειας – **float**
  - Πραγματικοί διπλής ακρίβειας – **double**
  - Χαρακτήρες – **char**
  - Πίνακες – **arrays**
  - Απουσία τύπου – **void**
- Δομές δεδομένων που ορίζονται από το χρήστη – **struct**, **enum**, **union**
- Μεταβλητές
  - Ορίζονται δίνοντας τον τύπο και το όνομα τους
  - Κανόνες για τα ονόματα
    - Δεν πρέπει να είναι κάποια δεσμευμένη λέξη (π.χ. int, if, κλπ.)
    - Είναι case sensitive
    - Μπορούν να περιέχουν γράμματα, αριθμούς και underscores (_)
    - Ο πρώτος χαρακτήρας δε μπορεί να είναι αριθμός
  - Οι μεταβλητές **πάντα** έχουν τον τύπο με τον οποίο δηλώθηκαν, ενώ κατά την εκτέλεση μπορεί να αλλάξει η τιμή τους
  - Φροντίζουμε πάντα να αρχικοποιούμε μια μεταβλητή πριν τη διαβάσουμε γιατί διαφορετικά μπορεί να διαβάσουμε «σκουπίδια» (οποιαδήποτε τιμή)

```c
int x, X;
int y = 1, z = 2, _w12;
float f = 1.2f;
double d;
char c = 'a';

x = 0;
X = 2;
y = y + 1;
d = 0.1;

printf("x = %d\n", x);
printf("X = %d\n", X);
printf("y = %d\n", y);
printf("f = %f\n", f);
printf("d = %lf\n", d);
printf("c = %c\n", c);
printf("w = %d\n", _w12);
```

Βασικό συντακτικό C (3/6)

Βασικοί τελεστές

Αριθμητικοί τελεστές

Τελεστής	Περιγραφή
x + y	πρόσθεση
x – y	Αφαίρεση
x * y	Πολλαπλασιασμός
x / y	διαίρεση
x % y	υπόλοιπο ακέραιας διαίρεσης
x++, ++x	αύξηση κατά ένα
x--, --x	μείωση κατά ένα

Συσχετιστικοί τελεστές

Τελεστής	Περιγραφή
==	ισότητα
!=	ανισότητα
>	μεγαλύτερο
<	μικρότερο
>=	μεγαλύτερο ή ίσο
<=	μικρότερο ή ίσο

Τελεστές εκχώρησης

Τελεστής	Περιγραφή
x = y	εκχώρηση
x += y	x = x + y
x -= y	x = x - y
x *= y	x = x * y
x /= y	x = x / y
x %= y	x = x % y

Λογικοί τελεστές

Τελεστής	Περιγραφή
x && y	λογική σύζευξη
x || y	λογική διάζευξη
!x	λογική άρνηση

- Στη C δεν έχουμε ειδικό τύπο δεδομένων για booleans (τιμές αληθείας) ούτε true και false
  - True είναι κάθε μη μηδενικός αριθμός (συνήθως το 1)
  - False είναι μόνο το μηδέν
- Οι λογικοί τελεστές επιστρέφουν 1 για true και 0 για false

Βασικό συντακτικό C (4/6)

- Βασικά statements – Όλα τα παρακάτω δουλεύουν με παρόμοιο συντακτικό και σημασιολογία όπως και στις JavaScript, PHP
  - Το ερωτηματικό (semicolon **;**) τερματίζει ένα statement
  - **Block statement** {}
    - Ομαδοποιεί πολλαπλά statements
  - **if / else if / else**
    - Εκτέλεση υπό συνθήκη
  - **while, for** επανάληψη
  - **break**, **continue**
    - έξοδος από επανάληψη ή επιστροφή στην αρχή της
  - **switch**
    - επιλογή τμήματος κώδικα
    - Σε αντίθεση με τις JavaScript και PHP, μπορούμε να κάνουμε switch μόνο πάνω σε ακέραιες τιμές (π.χ. int, char)

```
int i;
for (i = 0; i < 10; i++) {
  if (i == 0) {
    printf("First iteration\n");
    continue;
  }
  else if (i == 9) {
    printf("Last iteration\n");
    break;
  }
  switch (i) {
    case 1:  printf("i is 1\n"); break;
    case 2:  printf("i is 2\n"); break;
    default: printf("i is not 1 or 2\n");
             break;
  }
}
while (i) //same as while(i != 0)
  printf("i = %d\n", i--);
```

Βασικό συντακτικό C (5/6)

- Συναρτήσεις (1/2)
  - Μια συνάρτηση περιέχει ένα σύνολο από εντολές που εκτελούνται όταν κληθεί
  - Η συνάρτηση έχει **επιστρεφόμενο τύπο** και όταν επιστρέφουμε κάποια τιμή με το return πρέπει να είναι συμβατή με αυτόν τον τύπο
    - Η επιστρεφόμενη τιμή μπορεί να είναι **void**
      - Τότε μπορούμε να επιστρέψουμε από τη συνάρτηση με σκέτο return; χωρίς έκφραση
  - Οι **παράμετροι** είναι λίστα από δηλώσεις μεταβλητών χωρισμένες με κόμμα
    - Κάθε παράμετρος περιέχει τον τύπο και το όνομά της
  - Ο επιστρεφόμενος τύπος μαζί με τις παραμέτρους μιας συνάρτησης ορίζουν την **υπογραφή** της συνάρτησης
  - Οι μεταβλητές που ορίζονται μέσα σε μια συνάρτηση είναι **τοπικές** (local) μεταβλητές και ορατές μόνο μέσα στη συνάρτηση
  - Οι μεταβλητές που ορίζονται έξω από συναρτήσεις είναι **καθολικές** (global) μεταβλητές και είναι ορατές σε όλες τις συναρτήσεις στο πρόγραμμα
  - Οι παράμετροι έχουν τις τιμές που δίνονται κατά την κλήση της συνάρτησης και συμπεριφέρονται ως τοπικές μεταβλητές
  - Η κλήση συνάρτησης είναι έκφραση και μπορεί να χρησιμοποιηθεί σε συνδυασμό με όλες τις εκφράσεις και τελεστές που έχουμε δει

```
// function definition
return_type name(parameters) {
  // code to be executed
  return expression;
}
// function call
x = name(argument1, argument2);
```

Βασικό συντακτικό C (6/6)

- Συναρτήσεις (2/2)
  - Σε ένα σημείο ενός προγράμματος C μπορούμε να «δούμε» μόνο τις συναρτήσεις και μεταβλητές που έχουν οριστεί πιο πάνω και όχι αυτές που ορίζονται πιο κάτω
  - Ειδικά για τις συναρτήσεις, αν θέλουμε να χρησιμοποιήσουμε μια συνάρτηση που ορίζεται πιο κάτω πρέπει να υπάρχει μια **δήλωσή** της
    - Η δήλωση συνάρτησης μοιάζει με τον ορισμό συνάρτησης αλλά χωρίς να υπάρχει σώμα
    - Ουσιαστικά δηλώνουμε την **υπογραφή** της συνάρτησης ώστε ο compiler να ξέρει πως χρησιμοποιείται και να μπορέσει να την καλέσει ακόμα και αν δεν έχει δει ακόμα τον ορισμό της
    - Προφανώς, η υπογραφή της δήλωσης θα πρέπει να ταυτίζεται με την υπογραφή του ορισμού

```c
#include <stdio.h>

//function definition
int max(int a, int b) {
  if (a > b) return a;
  else       return b;
}

//function declaration
float average(float x, float y);

int main() { //function definition
  printf("max(1, 2) = %d\n",
    max(1, 2));
  printf("average(1, 2) = %f",
    average(1, 2));
  return 0;
}

//function definition
float average(float x, float y)
  { return (x + y) / 2; }
```

Πίνακες – Arrays (1/3)

- Ο πίνακας είναι μια δομή δεδομένων που ομαδοποιεί πολλαπλά τιμές και επιτρέπει να τις προσπελάσουμε με βάση τη θέση τους
  - Σε αντίθεση με τις JavaScript και PHP, στη C δηλώνεται ο τύπος του πίνακα και όλα τα στοιχεία είναι **ομοειδή**
  - Επίσης, ο πίνακας έχει συγκεκριμένο αριθμό στοιχείων που ορίζεται κατά τη δήλωσή του
    - Δε μπορούμε αργότερα να προσθέσουμε στοιχεία στον πίνακα
  - Η προσπέλαση στοιχείων (indexing) γίνεται με τον τελεστή [] και η αρίθμηση ξεκινάει από μηδέν
    - Σε αντίθεση με την Python δεν επιτρέπονται αρνητικά indices

```
type arrayName [ size ];
type arrayName [ size ] =
 {value1, value2, value3 };
type arrayName [] =
 {value1, value2, value3 };
```

```c
int numbers[5];//5 elements, all uninitialized
int grades[5] = { 10, 8, 7, 6}; //5 elements, last uninitialized
int grades2[] = { 10, 8, 7, 6}; //4 elements
int i;
for (i = 0; i < 5; i++)
  numbers[i] = i + 5; //after for, numbers = 5, 6, 7, 8, 9
grades[4] = 9; //grades is now 10, 8, 7, 6, 9
```

Πίνακες – Arrays (2/3)

- Εκτός από απλούς πίνακες, μπορούμε να έχουμε πίνακες **πολλαπλών** διαστάσεων
  - Και πάλι ορίζεται ο τύπος που είναι ίδιος για όλα τα στοιχεία, καθώς και τα μεγέθη της κάθε διάστασης

```
type arrayName [size1][size2][size3];
```

```
int two_dim[5][10];
int three_dim[5][10][4];
double four_dim[2][3][4][5];
```

	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

```
#include <stdio.h>
int main() {
  int a[3][4] = {
     {0, 1, 2, 3} , /* first  row (index 0) */
     {4, 5, 6, 7} , /* second row (index 1) */
     {8, 9, 10, 11} /* third  row (index 2) */
  };
  int i, j;
  for (i = 0; i < 3; i++) {
    for (j = 0; j < 4; j++)
      printf("%d ", a[i][j]);
    printf("\n");
  }
  return 0;
}
```

Πίνακες – Arrays (3/3)

- Μπορούμε να δώσουμε ένα πίνακα ως όρισμα σε συνάρτηση
  - Η παράμετρος δίνεται όμοια με ορισμό ενός πίνακα, ως ***type name[]***
    - Δεν είναι λάθος να δώσουμε και το μέγεθος αλλά δε λαμβάνεται υπόψιν
  - Αν έχουμε πολλαπλές διαστάσεις πρέπει να δώσουμε τα μεγέθη όλων των διαστάσεων εκτός από την πρώτη
  - Στη C δεν υπάρχει κάποιος τελεστής ή συνάρτηση για να πάρουμε το **μέγεθος** του πίνακα
  - Όταν δίνουμε λοιπόν ένα πίνακα σε μια συνάρτηση, συνήθως δίνουμε και ένα επιπλέον όρισμα που περιέχει το πλήθος των στοιχείων του
- Δεν μπορούμε να επιστρέψουμε πίνακα από συνάρτηση
  - Θα δούμε πιο μετά πως μπορούμε να προσομοιώσουμε αυτή τη λειτουργικότητα με δείκτες (pointers)

```c
#include <stdio.h>

double average(int a[], int size) {
   int i;
   double sum = 0;

   for (i = 0; i < size; i++)
      sum += a[i];
   return sum / size;
}

int main() {
   int grades[] = { 10, 8, 7, 6};
   double avg = average(grades, 4);
   printf("Average = %lf\n", avg);
   return 0;
}
```

Συμβολοσειρές – Strings (1/2)

- Στη C δεν υπάρχει ξεχωριστός τύπος δεδομένων για συμβολοσειρές
- Τα strings είναι πίνακες από χαρακτήρες με τελευταίο στοιχείο τον ειδικό χαρακτήρα '\0'
  - Ο χαρακτήρας αναφέρεται και ως null character, και τα strings λέγονται null-terminated (τερματίζουν με null)
- Ως πίνακες, μπορούμε να διαβάσουμε και να γράψουμε τα δεδομένα τους με τον τελεστή []
- Στο header file <string.h> ορίζονται διάφορες συναρτήσεις βιβλιοθήκης για strings, κάποιες από τις οποίες αναφέρονται παρακάτω
  - Παρατηρείστε ότι κάποιες ενέργειες πάνω σε strings που σε JavaScript και PHP γίνοντας απλά με τελεστές (π.χ. αντιγραφή, συνένωση, κλπ.) στη C χρειάζονται ειδική συνάρτηση

Ιδιότητα / Μέθοδος	Περιγραφή
strlen(str)	Επιστρέφει το πλήθος των χαρακτήρων του string.
strcmp(s1, s2)	Συγκρίνει τα δύο strings s1 και s2 λεξικογραφικά. Επιστρέφει 0 αν είναι ίδια, αρνητική τιμή αν s1 < s2 και θετική τιμή αν s1 > s2.
strcpy(dest, src)	Αντιγράφει το string src στο string dest. Θα πρέπει να υπάρχει αρκετός χώρος (δηλαδή το string dest να έχει ίδιους ή περισσότερους χαρακτήρες από το src).
strcat(dest, src)	Προσθέτει το string src στο τέλος του string dest. Θα πρέπει να υπάρχει αρκετός χώρος στο dest.

Συμβολοσειρές – Strings (2/2)

- Παραδείγματα με strings

```c
#include <stdio.h>

int main () {
  char msg1[6] =
     {'H', 'e', 'l', 'l', 'o', '\0'};
  char msg2[] = "Hello";
  printf("Message1: %s\n", msg1);
  printf("Message2: %s\n", msg2);
  return 0;
}
```

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

int main () {
  char str1[12] = "Hello";
  char str2[12] = "World";
  char str3[12];
  int  len ;

  /* copy str1 into str3 */
  strcpy(str3, str1);
  printf("strcpy( str3, str1) :  %s\n", str3 );

  /* concatenates str1 and str2 */
  strcat( str1, str2);
  printf("strcat( str1, str2):   %s\n", str1 );

  /* total lenghth of str1 after concatenation */
  len = strlen(str1);
  printf("strlen(str1) :  %d\n", len );

  return 0;
}
```

Δομές – Structs (1/2)

- Με τα structs δημιουργήσουμε οντότητες με δεδομένα
  - Μοιάζουν με τα objects της JavaScript και της PHP, αλλά χωρίς να μπορούν να περιέχουν ενσωματωμένη λειτουργικότητα
    - Λειτουργικότητα μπορεί να παρέχεται ως ξεχωριστές συναρτήσεις με όρισμα structs
  - Ένα struct μπορεί να περιέχει οσαδήποτε δεδομένα και οποιουδήποτε τύπου
    - char, int, float, double, arrays, structs, κλπ.
  - Η προσπέλαση ενός μέλους του struct γίνεται με τον τελεστή ***member access operator*** **.**
  - Μπορούμε να αρχικοποιήσουμε διαδοχικά τα μέλη ενός struct με το συντακτικό {value1, … valueN}

```
struct structName {
  type1 memberName1;
  type2 memberName2;
  …
  typeN memberNameN;
};
struct structName name;
name.member = value;
```

```
struct X {
  int x;
}
struct StructOfStructs {
  struct Various v;
  struct Point p;
  struct X x;
}
```

```
struct Various {
  int a;
  char b;
  float c;
  double d;
  char e[10];
  int f[20];
};
```

```
#include <stdio.h>
struct Point {
 int x;
 int y;
};

int main() {
 struct Point p;
 struct Point o{0,0};
 p.x = 1;
 p.y = 2;
 printf("(%d,%d)\n",
  p.x, p.y);
 printf("(%d,%d)\n",
  o.x, o.y);
 return 0;
}
```

Δομές – Structs (2/2)

Παράδειγμα χρήσης struct

```c
#include <stdio.h>
struct Movie {
 char title[50];
 char director[50];
 int  year;
};

void print_movie(struct Movie movie)
  { printf("%s (%d), directed by %s\n", movie.title, movie.year, movie.director); }

int main() {
  struct Movie movie1 { "The Dark Knight", "Christopher Nolan", 2008};
  struct Movie movie2 { "Avengers: Endgame", "Anthony Russo & Joe Russo", 2019};
  struct Movie movies[] = {
    { "The Lord of the Rings:The Fellowship of the Ring", "Peter Jackson", 2001},
    { "The Lord of the Rings:The Two Towers", "Peter Jackson", 2002},
    { "The Lord of the Rings:The Return of the King", "Peter Jackson", 2003}
  };
  int i;
  print_movie(movie1); print_movie(movie2);
  for (i = 0; i < 3; i++)
    print_movie(movies[i]);
  return 0;
}
```

Είσοδος / Έξοδος

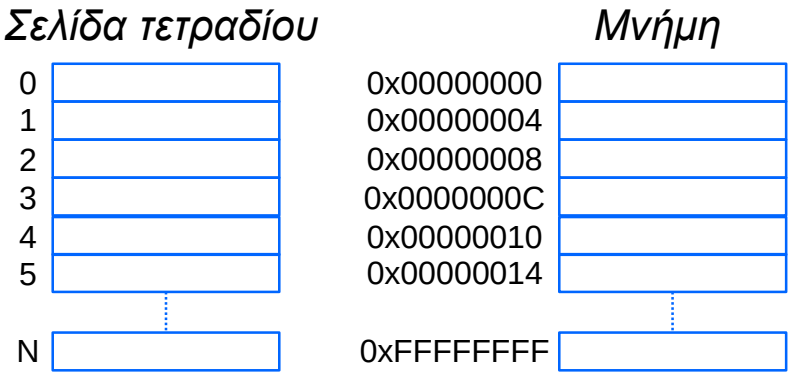
- Είδαμε ήδη ότι μπορούμε να τυπώσουμε στην κονσόλα χρησιμοποιώντας τη συνάρτηση **printf**
  - `printf("%d", value); //print integer`
- Αντίστοιχα, μπορούμε να ζητήσουμε είσοδο από το χρήστη με τη συνάρτηση **scanf**
  - `scanf("%d", &value); //read integer`
  - Ομοίως τυπώνουμε και για όλους τους τύπους βάζοντας το κατάλληλο γράμμα για το format (%c, %d, %f, %lf, κλπ.)
  - Η μόνη εξαίρεση είναι όταν διαβάσουμε string που το γράφουμε ως `scanf("%s", str); //read string`
  - Αντίστοιχα με την printf, μπορούμε να διαβάζουμε και πολλές τιμές με μια κλήση

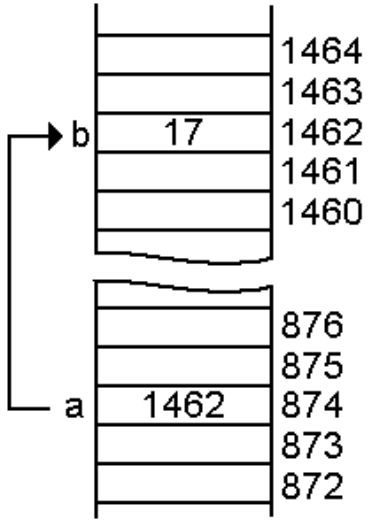
```c
#include <stdio.h>
int main( ) {
  double d;
  char str[100];
  int i;

  printf("Enter a double: ");
  scanf("%lf", &d);
  printf("d = %lf\n", d);

  printf("Enter a string and an integer: ");
  scanf("%s %d", str, &i);
  printf( "You entered: %s %d\n", str, i);

  return 0;
}
```

```
Enter a double: 123.456
d = 123.456000
Enter a string and an integer: Hello 789
You entered: Hello 789
```

Δείκτες – Pointers (1/8)

- Κάθε μεταβλητή αποθηκεύεται στη μνήμη και έχει μια συγκεκριμένη διεύθυνση
  - Σκεφτείτε τη μνήμη σαν μια πολύ μεγάλη σελίδα τετραδίου, όπου τα δεδομένα κάθε μεταβλητής γράφονται σε ξεχωριστή γραμμή
  - Η διεύθυνση μιας μεταβλητής είναι ο αριθμός της γραμμής στην οποία γράφονται τα δεδομένα της μέσα στη σελίδα
- Οι δείκτες είναι μεταβλητές που οι τιμές τους είναι διευθύνσεις μνήμης
  - «Δείχνουν» δηλαδή σε θέσεις στη μνήμη ή σε άλλες μεταβλητές

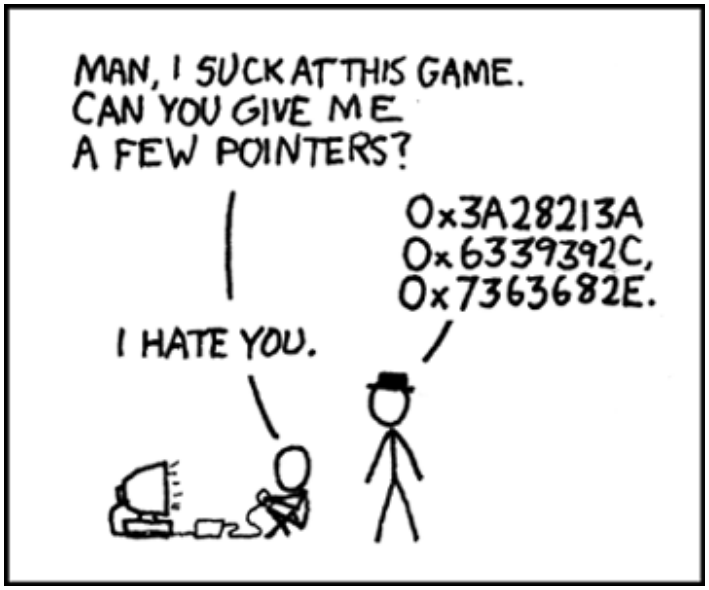




Δείκτες – Pointers (2/8)

- Όπως όλες τις μεταβλητές, έτσι και οι δείκτες δηλώνονται πριν τη χρήση με το συντακτικό `type * variable;`
- Ανεξάρτητα με τον τύπο type, η *τιμή* που έχει ένας δείκτης είναι ένα νούμερο που αναπαριστά μια διεύθυνση μνήμης
  - Τα νούμερα αυτά τα γράφουμε συνήθως σε δεκαεξαδική μορφή

```
int    *ip; /* pointer to an integer  */
double *dp; /* pointer to a double    */
float  *fp; /* pointer to a float     */
char   *ch  /* pointer to a character */
```



Δείκτες – Pointers (3/8)

- Για να είναι χρήσιμος ένας δείκτης πρέπει να δείχνει σε δεδομένα στα οποία έχουμε πρόσβαση και μας ενδιαφέρουν (π.χ. μεταβλητές)
  - Στην αναλογία με το τετράδιο, το να κρατάμε τη θέση μιας κενής γραμμής δε έχει ενδιαφέρον
- Για να πάρουμε τη διεύθυνση μιας μεταβλητής χρησιμοποιούμε τον τελεστή address of **&**
- Έχοντας ένα δείκτη, αυτό που μπορούμε να κάνουμε είναι να πάρουμε την τιμή της μεταβλητής στην οποία δείχνει με τον τελεστή dereference ***

```c
#include <stdio.h>

int main() {
  int *x = (int *) 123;
  int *y = (int *) 0x3A28213A;
  int var = 10;
  int *z = &var;

  printf("x = 0x%p\n", x); //0x0000007B
  printf("y = 0x%p\n", y); //0x3A28213A
  printf("z = 0x%p\n", z); //0x00EFFA54
  printf("*z = %d\n", *z); //10
  var = 11;
  printf("*z = %d\n", *z); //11

  return 0;
}
```

Δείκτες – Pointers (4/8)

- Ένας δείκτης που δεν έχει αρχικοποιηθεί μπορεί να έχει οποιαδήποτε τιμή («σκουπίδια στη μνήμη») και να δείχνει οπουδήποτε στη μνήμη
  - Από την τιμή του όμως δεν μπορούμε να ξέρουμε αν του έχουμε αναθέσει να δείχνει σε δικιά μας μεταβλητή ή δείχνει κάπου στην τύχη
  - Αν πάμε να διαβάσουμε από μια διεύθυνση που δεν είναι δικιά μας το πρόγραμμά μας θα τερματίζεται με crash
    - Συνήθως θα σας βγάλει μήνυμα «Segmentation fault»
- Χρησιμοποιούμε την ειδική τιμή **NULL** για να ορίσουμε ότι ένας δείκτης δεν έχει πάρει ακόμα τιμή
  - Ουσιαστικά ο δείκτης δε «δείχνει» ακόμα κάπου
  - Το NULL αντιστοιχεί στη θέση μνήμης 0 που είναι δεσμευμένη από το λειτουργικό σύστημα
    - Μπορούμε να ελέγχουμε αν ένας δείκτης έχει αρχικοποιηθεί ή όχι χρησιμοποιώντας τον σε συνθήκες ή συγκρίνοντάς τον με το NULL

```c
#include <stdio.h>

int main () {
  int  *ptr = NULL;
  printf("ptr = %p\n", ptr); //prints 0
  if (ptr) //same as if (ptr != NULL)
    printf("ptr is not null");
  if (!ptr) //same as if (ptr == NULL)
    printf("ptr is null");
  return 0;
}
```

Δείκτες – Pointers (5/8)

- Στη C οι κλήσεις των συναρτήσεων γίνονται με βάση την τιμή (**call by value**)
  - Αποτιμάται η έκφραση και η τιμή του πραγματικού ορίσματος δίνεται στη συνάρτηση
  - Έτσι, δίνοντας μια μεταβλητή ως όρισμα σε συνάρτηση δε μπορούμε να αλλάξουμε την τιμή της
- Για να πετύχουμε κλήση με αναφορά σε μεταβλητές (**call by reference**), δηλαδή οι αλλαγές σε μεταβλητές να είναι ορατές έξω από τη συνάρτηση, χρησιμοποιούμε δείκτες

```c
#include <stdio.h>

// function to swap the values of x and
void swap1(int x, int y) {
  int temp = x; // save the value of x
  x = y;        // put y into x
  y = temp;     // put temp into y
}
void swap2(int *x, int *y) {
  int temp = *x; // save the value of x
  *x = *y;       // put y into x
  *y = temp;     // put temp into y
}

int main () {
  int a = 1, b = 2;
  printf("a = %d, b = %d\n", a, b); //1, 2
  swap1(a, b);
  printf("a = %d, b = %d\n", a, b); //1, 2
  swap2(&a, &b);
  printf("a = %d, b = %d\n", a, b); //2, 1
  return 0;
}
```

Δείκτες – Pointers (6/8)

- Μπορούμε να έχουμε δείκτες και σε structs
- Έχοντας ένα δείκτη σε struct, μπορούμε να πάρουμε το struct με τον τελεστή * και μετά να πάρουμε τα μέλη του με τον τελεστή .
- Εναλλακτικά, μπορούμε να πάρουμε ένα μέλος απευθείας από τον δείκτη με τον τελεστή ->

```
struct S {
  type1 memberName1;
  type2 memberName2;
};
struct S* ptr = …;
(*ptr).memberName1 = …;
ptr->memberName2 = …;
```

```
#include <stdio.h>
struct Point {
  int x;
  int y;
};

void move(struct Point *pointer, int dx, int dy) {
  pointer->x += dx; //same as (*pointer).x += dx;
  pointer->y += dy; //same as (*pointer).x += dx;
}

int main() {
  struct Point p{ 1, 2 };
  struct Point *a = &p;
  p.x += 1;
  (*a).x += 1;

  printf("(%d, %d)\n", p.x, p.y); //prints (3, 2)
  move(&p, 3, 5);
  printf("(%d, %d)\n", p.x, p.y); //prints (6, 7)
  printf("(%d, %d)\n", a->x, a->y); //prints (6, 7)
  return 0;
}
```

Δείκτες – Pointers (7/8)

- Δείκτες και πίνακες
  - Όταν δηλώνουμε ένα πίνακα δεσμεύουμε στη μνήμη τόσες συνεχόμενες θέσεις όσο το μέγεθος του πίνακα
  - Η τιμή που έχει η μεταβλητή που κρατάει τον πίνακα είναι ουσιαστικά ένας δείκτης στην αρχή των συνεχόμενων αυτών θέσεων μνήμης
  - Μπορούμε λοιπόν να αναθέσουμε ένα πίνακα σε ένα δείκτη και αντίστοιχα μπορούμε να χρησιμοποιήσουμε ένα δείκτη για να προσπελάσουμε παρακάτω θέσεις μνήμης σαν να ήταν πίνακας
  - Αντίστοιχα, οι συναρτήσεις που παίρνουν όρισμα πίνακα μπορούν να δηλωθούν με τον αντίστοιχο δείκτη

```c
#include <stdio.h>
double sum(double *a, int size) {
  double s = 0;
  int i;
  for (i = 0; i < size; i++)
    s += a[i];
  return s;
}
int main () {
  double balance[3] = {100.0, 3.4, 17.0};
  double *p = balance;
  int i;
  for (i = 0; i < 3; i++) {
    printf("p[%d] = %lf, *(p + %d) = %lf\n",
      i, p[i], i, *(p + i));
  }
  printf("sumBalance = %lf, sumP = %lf\n",
    sum(balance, 3), sum(p, 3));
  return 0;
}
```

```
p[0] = 100.000000, *(p + 0) = 100.000000
p[1] = 3.400000, *(p + 1) = 3.400000
p[2] = 17.000000, *(p + 2) = 17.000000
sumBalance = 120.400000, sumP = 120.400000
```

Δείκτες – Pointers (8/8)

- Δείκτες σε δείκτες
  - Δείκτη μπορούμε να έχουμε σε μεταβλητή οποιουδήποτε τύπου
  - Άρα μπορούμε να και δείκτες σε δείκτες
    - Και σε οποιοδήποτε βάθος
  - Το συντακτικό παραμένει το ίδιο, απλά προσθέτουμε στη δήλωση ένα ακόμα * για κάθε επίπεδο δείκτη
    - Το ίδιο ισχύει και για να πάρουμε την τιμή


```c
#include <stdio.h>

int main () {
   int var = 100;
   int *ptr = &var;
   int **pptr = &ptr;

   printf("var = %d\n", var);
   printf("*ptr = %d\n", *ptr);
   printf("**pptr = %d\n", **pptr);

   printf("*pptr = 0x%p\n", *pptr);
   printf("ptr = 0x%p\n", ptr);

   return 0;
}
```

```
var = 100
*ptr = 100
**pptr = 100
*pptr = 0x008FF8D8
ptr = 0x008FF8D8
```

Δυναμική Δέσμευση μνήμης (1/3)

- Για να δηλώσουμε ένα πίνακα πρέπει να ξέρουμε από πριν το μέγεθός του
- Σε περίπτωση που το μέγεθος εξαρτάται από την εκτέλεση του προγράμματος χρειαζόμαστε δυναμική δέσμευσης μνήμης
- Οι βασικές συναρτήσεις που παρέχει η C είναι οι:
  - **void *malloc(int bytes);**
    - Δεσμεύει τόσα bytes και επιστρέφει ένα pointer στη δεσμευμένη διεύθυνση
  - **void free(void *address);**
    - Αποδεσμεύει τη μνήμη που αντιστοιχεί στη διεύθυνση

```c
#include <stdio.h>
#include <stdlib.h>
double average(int a[], int size) {
  int i, sum = 0;
  for (i = 0; i < size; i++)
    sum += a[i];
  return (double) sum / size;
}
int main() {
  int i, n;
  int* grades;
  printf("Enter number of grades: ");
  scanf("%d", &n);
  grades = (int *) malloc(n * sizeof(int));
  for (i = 0; i < n; i++) {
    printf("Enter grades #%d: ", i + 1);
    scanf("%d", grades + i);
    //same as scanf("%d", &grades[i]);
  }
  printf("Average: %lf\n", average(grades, n));
  free(grades);
  return 0;
}
```

Δυναμική Δέσμευση μνήμης (2/3)

- Όταν δεσμεύουμε μνήμη για ένα πίνακα τύπου T και μεγέθους N ακολουθούμε το μοτίβο
  `T* p = (T*) malloc(N * sizeof(T));`

```
int *x = (int *) malloc(10 * sizeof(int)); //10 ints
struct Point *p = (struct Point *) malloc(sizeof(struct Point)); //1 struct Point
```

- Η malloc δέχεται αριθμό bytes, οπότε πολλαπλασιάζουμε το μέγεθος του πίνακα με το sizeof του τύπου, δηλαδή το χώρο που χρειάζεται για ένα στοιχείο του πίνακα
- Εκτός από πίνακες οποιουδήποτε μεγέθους, με κατάλληλες κλήσεις της malloc μπορούμε να δημιουργήσουμε δυναμικό αριθμό από structs
  - Αυτός είναι και ο τρόπος που υλοποιούνται οι δομές δεδομένων στη C, π.χ. λίστες, δέντρα, hashtables, κλπ.

Δυναμική Δέσμευση μνήμης (3/3)

```c
#include <stdio.h>
#include <stdlib.h>
struct list {
  int value;
  struct list* next;
};

void push(struct list* head, int value) {
  struct list* p = head;
  while(p->next != NULL) p = p->next;
  p->next = (struct list *)
    malloc(sizeof(struct list));
  p->next->value = value;
  p->next->next = NULL;
}

void print_list(struct list* head) {
  struct list* p;
  for (p = head; p != NULL; p = p->next) {
    if (p != head)
      printf(", ");
    printf("%d", p->value);
  }
}
```

```c
void free_list(struct list* head) {
  struct list* p = head, *previous=NULL;
  while (p) {
    if (previous) free(previous);
    previous = p;
    p = p->next;
  }
}
int main() {
  int n = 1;
  struct list* head = (struct list*)
    malloc(sizeof(struct list));
  printf("Enter list head: ");
  scanf("%d", &head->value);
  head->next = NULL;
  while (n != 0) {
    printf("Enter next element (0 to stop): ");
    scanf("%d", &n);
    if (n != 0) push(head, n);
  }
  print_list(head);
  free_list(head);
  return 0;
}
```

Προχωρημένο παράδειγμα - Λίστες