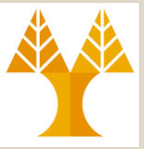
Τμήμα Πληροφορικής
Πανεπιστήμιο Κύπρου



ΕΠΛ232 – Προγραμματιστικές Τεχνικές και Εργαλεία

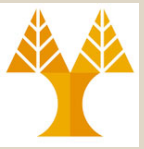
Διάλεξη 4: Οργάνωση Προγράμματος & Ανατομία Προγράμματος

(Κεφάλαιο 10, KNK-2ED)

Δημήτρης Ζεϊναλιπούρ

http://www.cs.ucy.ac.cy/courses/EPL232

Περιεχόμενο Διάλεξης 4



- **Οργάνωση Προγράμματος (Κεφ. 10)**
  - *Τοπικές Μεταβλητές* (αυτόματες, static), *Εξωτερικές Μεταβλητές* (block/file scope), Μπλοκ { }, *Εμβέλεια Μεταβλητών*
  - *Οργάνωση Προγράμματος* σε ένα αρχείο κώδικα.
- **Ανατομία Προγράμματος**
  - Εισαγωγή: Διεργασίες, Μνήμη & Διευθύνσεις
  - Ανατομία Προγράμματος
    - Στην Δευτερεύουσα Μνήμη (Disk)
    - Στην Κύρια Μνήμη (RAM)
- **(Προκαταρτική) Είσοδος/Έξοδος από Αρχεία**
  - Συναρτήσεις Βιβλιοθήκης fopen, fread, fwrite, παράδειγμα με ψευδοτυχαίους αριθμούς

Τοπικές (Αυτόματες) Μεταβλητές (Local Automatic Variables)

- Επαναληπτικά, αναφέρουμε ότι μια **μεταβλητή** που **δηλώνεται στο σώμα** μιας συνάρτησης λέγεται ότι είναι **τοπική (*local*)** στη συνάρτηση:

```
int sum_digits(int n) {
  int sum = 0;   /* local var */
  ...
}
```

```
void f(void)
{                          C99
  ...
  int i;
  ...                      scope of i
}
```

- **Ιδιότητες** τοπικών μεταβλητών (αλλά και παραμέτρων):
  - ***Αυτόματη αποθηκευτική διάρκεια (Automatic storage duration):*** **Δεσμεύεται (allocated)** «αυτόματα» ο χώρος όταν καλείται η συνάρτηση και **αποδεσμεύεται (deallocated)** αυτόματα όταν **ολοκληρωθεί** η εκτέλεση της.
  - ***Εμβέλεια Μπλοκ (Block scope):*** Μια τοπική μεταβλητή είναι **«ορατή»** από τη **δήλωση** της **μέχρι** το **τέλος της συνάρτησης** που την δηλώνει.

Τοπικές Στατικές Μεταβλητές (Local Static Variables)

- Προσθέτοντας **`static`** στη δήλωση μιας τοπικής μεταβλητής, καθιστά την μεταβλητή να έχει **στατική αποθηκευτική διάρκεια (*static storage duration.)***
- Δηλαδή, η static μεταβλητή διατηρεί την τιμή της μέχρι την **επόμενη κλήση …**

```
#include <stdio.h>
void foo() {
  static int i;
  printf("%d", i);
  i++;}
```

```
int main() {
  foo(); foo(); foo();
  return 0;}
```

Εκτυπώνει:
0 1 2

- **`...`** αλλά η **ορατότητα** της παραμένει σε **επίπεδο block** συνάρτησης, όπως τις **αυτόματες μεταβλητές**
  - Συνεπώς, η μεταβλητή δεν είναι ορατή σε άλλες συναρτήσεις.
- **Χρήση:** Μεταβλητές που θα χρησιμοποιούνται μεταξύ αλλεπάλληλων κλήσεων (για αποφυγή συνεχούς δημιουργίας / καταστροφής τους)
  - **Γενικά συνίσταται να χρησιμοποιείται με μεγάλη ΠΡΟΣΟΧΗ.**

Τοπικές Στατικές Μεταβλητές (Local Static Variables)

Το `static` μπορεί να χρησιμοποιηθεί με **πίνακες** και με **δείκτες** που θα δούμε στη συνέχεια

Πως δουλεύει ο ακόλουθος κώδικας;

```
#include <stdio.h> // printf
void foo(int i) {
  static char name[10]="variable";
  printf("%s\n", name);
  name[i]='0';
}
int main() {
  int i = 0;
  foo(i++); foo(i++); foo(i);
  return 0;
}
```

Αποτέλεσμα Προγράμματος:

```
variable
0ariable
00riable
```

Μια global μεταβλητή δεν χρειάζεται να είναι static όπως δείχνει η επόμενη διαφάνεια.

Εξωτερικές Μεταβλητές (External Variables)

- **Εξωτερικές Μεταβλητές (External Variables):** μεταβλητές που **ΔΕΝ** ορίζονται μέσα στο **σώμα** μιας συνάρτηση και **ΔΕΝ** είναι **παράμετροι** της.
  - Γνωστές ως **καθολικές μεταβλητές (*global variables*)**
- **Ιδιότητες Εξωτερικών Μεταβλητών:**
  - **Στατική Αποθηκευτική Διάρκεια (static storage duration)**
    - Όπως τις στατικές μεταβλητές δεν καταστρέφονται άμεσα
  - **Εμβέλεια Αρχείου (file scope)**
    - Μια μεταβλητή είναι **ορατή** από τη **δήλωση** της μέχρι το **τέλος του αρχείου** που **ορίζει** την **μεταβλητή**
- **Μειονεκτήματα Εξωτερικών Μεταβλητών:**
  - **Δύσκολο debugging, δύσκολη επαναχρησιμοποίηση κώδικα, σπατάλη μνήμης, για αυτό η χρήση τους να ΠΕΡΙΟΡΙΣΤΕΙ στο ελάχιστο στο μάθημα**

Εξωτερικές Μεταβλητές (External Variables)

- Τι πάει λάθος με το ακόλουθο κώδικα;

```
int i;

void print_one_row(void)
{
  for (i = 1; i <= 10; i++)
    printf("*");
}

void print_all_rows(void)
{
  for (i = 1; i <= 10; i++) {
    print_one_row();
    printf("\n");
  }
}
```

Αντί να εκτυπώνονται όλες οι γραμμές, εκτυπώνεται μόνο μια! ☹

Μπλοκ Εντολών (Blocks)

- Η C, όπως και άλλες γλώσσες, παρέχει την έννοια των **ρητών μπλοκ { }** για να **περιορίσει την εμβέλεια μεταβλητών**.
  - Λιγότερες συγκρούσεις ονομάτων (**name conflicts**)
  - Μείωση συνωστισμού δηλώσεων στην αρχή συναρτήσεων (**cluttering declarations**)

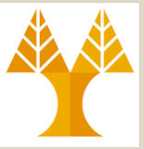
```
#include <stdio.h>
void foo(void) {
    // other statements ...
    {
        int i = 5;     // Automatic storage duration, block scope
        static int j; // Static storage duration, block scope
        printf("%d %d\n", i++, j++);
    }
    // printf("%d %d", i, j); - compile error: 'i' 'j' undeclared
   (first use in this function)
}
```

```
int main() {
    foo(); foo(); foo();
    return 0;
}
```

Εκτυπώνει:

```
5 0
5 1
5 2
```

Εμβέλεια Μεταβλητών (Variable Scope)



- **Αρχή Εμβέλειας:** Όταν γίνεται **δήλωση** μιας μεταβλητής που είναι **ήδη δηλωμένη** (ορατή), η νέα δήλωση «**αποκρύπτει**» την παλιά **δήλωση**.
  - Στο τέλος η **παλιά δήλωση** αποκτά ξανά το νόημα της.

```
int i ;                 /* Declaration 1 */

void f(int i )          /* Declaration 2 */
{
  i = 1;
}

void g(void)
{
  int i = 2;            /* Declaration 3 */

  if (i > 0) {
    int i ;             /* Declaration 4 */

    i = 3;
  }

  i = 4;
}

void h(void)
{
  i = 5;
}
```

Οργάνωση Προγράμματος (με 1 Αρχείο Πηγαίου Κώδικα *)

- * Αργότερα θα δούμε πως η σημαντική **αρχή της αφαιρετικότητας** μας επιβάλλει να **διασπάσουμε** ένα πρόγραμμα σε **πολλαπλά αρχεία** πηγαίου κώδικα.
- **Βασικά συστατικά** ενός **C προγράμματος** (σε σειρά εμφάνισης):
  - **Οδηγίες προεπεξεργασίας** μεταγλωττιστή (compiler preprocessing directives) such as `#include` and `#define`
  - Ορισμός **Νέων Τύπων** (Type definitions) - `typedef`
  - Δήλωση **Εξωτερικών Μεταβλητών** (Declarations of Ext. vars)
  - **Πρότυπα Συναρτήσεων** (function prototypes)
  - **Ορισμός Συναρτήσεων** (function definitions) - πριν την χρήση τους εναλλακτικά παίρνεται το λάθος `undefined symbol first referenced in file _print ...`
- Τύποι, μεταβλητές, κτλ., έχουν «ορατότητα» μόνο μετά τον ορισμό τους.

Οργάνωση Προγράμματος (με 1 Αρχείο Πηγαίου Κώδικα)

- **Βασικά Συστατικά Σχολίου:**
  - **Name:** Όνομα Συνάρτησης
  - **Description:** Τι ρόλο επιτελεί (Σύντομη και Εκτενέστερη Περιγραφή – Short/Long)
  - **Parameters:** Νόημα κάθε παραμέτρου (`@param`)
  - **Return Value:** Επεξήγηση της τιμής εξόδου, εάν υπαρχει (`@return`)
  - **Others:** Επεξήγηση οποιονδήποτε άλλων σημαντικών δεδομένων και χαρακτηριστικών, π.χ.,
    - περιγραφή των συνεπειών μεταβολής εξωτερικών μεταβλητών
    - αλγόριθμου, δομής δεδομένων, κτλ.

Οργάνωση Προγράμματος (με 1 Αρχείο Πηγαίου Κώδικα)

- Παράδειγμα Σχολίου για Χρήση με **Σύστημα Τεκμηρίωσης Κώδικα (Software Documentation System) Doxygen** (www.doxygen.org/)
  - Έχει μελετηθεί στα εργαστήρια.

Βασικά Συστατικά Σχολίου:

```
/**
 * <A short one line description>
 *
 * <Longer description>
 * <May span multiple lines or paragraphs as needed>
 *
 * @param  Description of method's or function's input parameter
 * @param  ...
 * @return Description of the return value
 */
```

Οργάνωση Προγράμματος (με 1 Αρχείο Πηγαίου Κώδικα)

Παράδειγμα Σχολίων για την Αρχή του Αρχείου

```
/**
 * @file
 * @author  John Doe <jdoe@example.com>
 * @version 1.0
 *
 * @section LICENSE
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License as
 * published by the Free Software Foundation; either version 2 of
 * the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
 * General Public License for more details at
 * http://www.gnu.org/copyleft/gpl.html
 *
 * @section DESCRIPTION
 *
 * The time module represents a moment of time.
 */
```

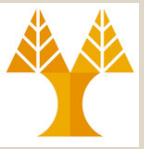
Οργάνωση Προγράμματος (με 1 Αρχείο Πηγαίου Κώδικα)

- **Σκελετός Προγράμματος (Skeleton Program):** Ένα υψηλού επιπέδου πρόγραμμα το όποιο περιέχει **κενό κώδικα (dummy code).**
  - **Χρησιμοποιείται** για να δημιουργηθεί η **βασική δομή** ενός **προγράμματος** και είναι **βασικό εργαλείο** για να **διαμεριστεί η** ανάπτυξη ενός λογισμικού σε μια **ομάδα προγραμματιστών**.
  - Περισσότερα μετά το **midterm** όταν θα μιλήσουμε για **SVN** και αρχές ανάπτυξης **μεγάλων προγραμμάτων**

```
/*************************************************************
 * main: Calls read_cards, analyze_hand, and print_result    *
 *       repeatedly.                                         *
 *************************************************************/
int main(void)
{
  for (;;) {
    read_cards();
    analyze_hand();
    print_result();
  }
}
```

```
/*************************************************************
 * read_cards:  Reads the cards into external variables;     *
 *              checks for bad cards and duplicate cards.    *
 *************************************************************/
void read_cards(void)
{
  …
}
```

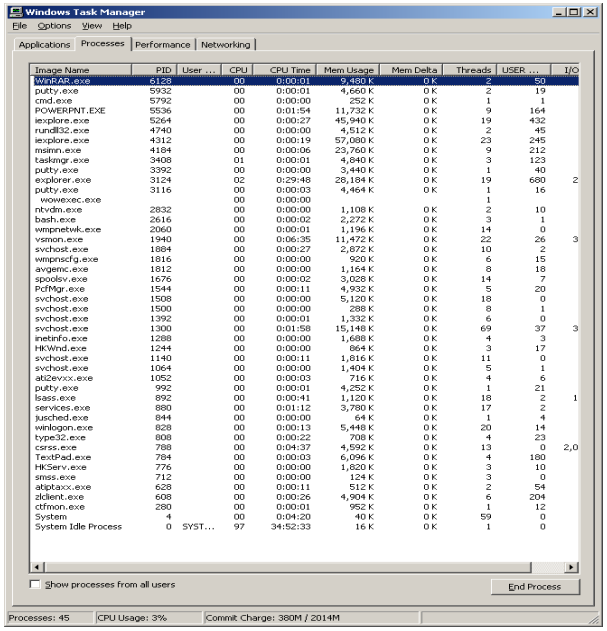
Περιεχόμενο Διάλεξης

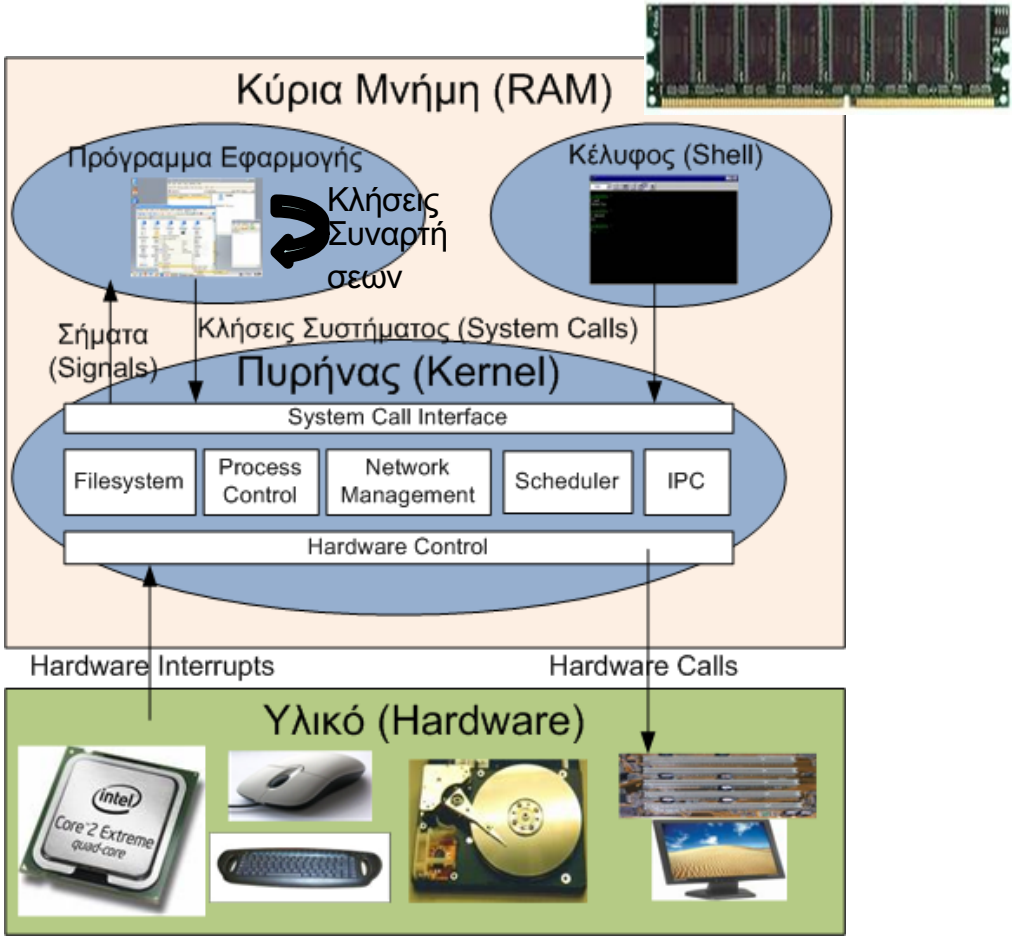


- **Οργάνωση Προγράμματος (Κεφ. 10)**
  - *Τοπικές Μεταβλητές* (αυτόματες, static), *Εξωτερικές Μεταβλητές* (block/file scope), Μπλοκ { }, *Εμβέλεια Μεταβλητών*
  - *Οργάνωση Προγράμματος* σε ένα αρχείο κώδικα.
- **Ανατομία Προγράμματος**
  - Εισαγωγή: Διεργασίες, Μνήμη & Διευθύνσεις
  - Ανατομία Προγράμματος
    - Στην Δευτερεύουσα Μνήμη (Disk)
    - Στην Κύρια Μνήμη (RAM)
- **(Προκαταρτική) Είσοδος/Έξοδος από Αρχεία**
  - Συναρτήσεις Βιβλιοθήκης fopen, fread, fwrite, παράδειγμα με ψευδοτυχαίους αριθμούς

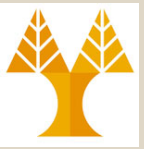
Ανατομία Συστήματος

- **Αριστερά:** Οι διεργασίες ενός συστήματος Windows. **Δεξιά**: Διεργασίες και Πυρήνας



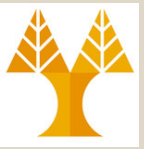


Ανατομία Συστήματος



- Γνωρίζουμε ότι ένα πρόγραμμα υπό εκτέλεση ονομάζεται **διεργασία (process)**
- Σε ένα Λειτουργικό Σύστημα εκτελούνται «**ψευδό-παράλληλα**» πολλές διεργασίες κάτω από τον **συντονισμό** της διεργασίας **πυρήνα (kernel)**
  - Σε συστήματα **πολλαπλών επεξεργαστών** (ή επεξεργαστές με **πολλαπλά νήματα** ελέγχου) εκτελούνται πραγματικά παράλληλα!
- Οι **διεργασίες** αυτές έχουν κάποιο **χώρο (μνήμη)** στον οποίο **τοποθετούν δεδομένα** (μεταβλητές, κτλ.)

Μνήμη & Διευθύνσεις

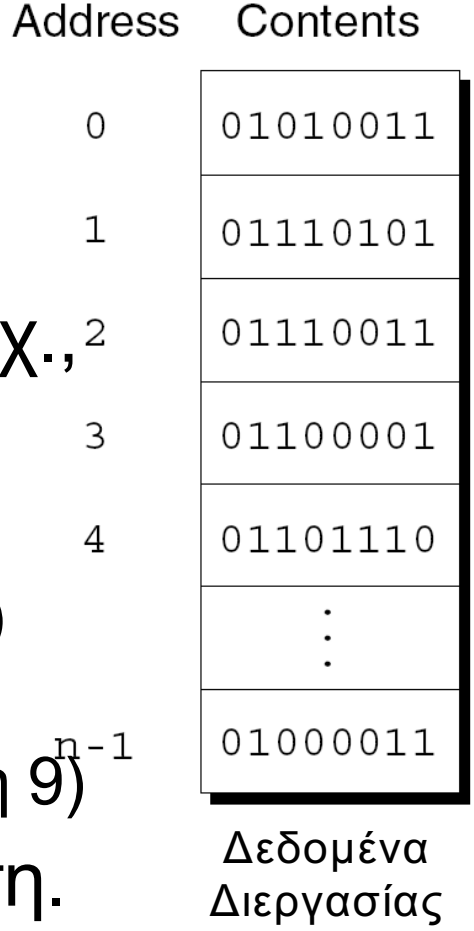


- Την μνήμη μπορούμε να την φανταστούμε ως ένα μεγάλο **πίνακα από bytes (8 bits)**, π.χ.,
  - 2 gigabytes = 2 147 483 648 Bytes
  - 4 gigabytes = 4 294 967 296 Bytes (max για 32-bit συστήματα)
- Τα bytes αυτά παραχωρούνται στις **διεργασίες** από τον **πυρήνα**.

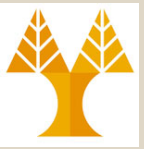
- Κάθε **byte** έχει μια μοναδική **διεύθυνση** (***address)***
- *Οι **διευθύνσεις** της κάθε διεργασίας (προγράμματος) είναι **ανεξάρτητες μεταξύ τους.***
  - *Δηλαδή, κάθε διεργασία έχει διευθύνσεις μνήμης που εκτείνονται από το **0** μέχρι το **n-1**.*
  - *Λόγω της **νοητής μνήμης (virtual memory),** ο πίνακας αυτός μπορεί να είναι όσο μεγάλη είναι η δευτερεύουσα σας μνήμη!*

Μνήμη & Διευθύνσεις

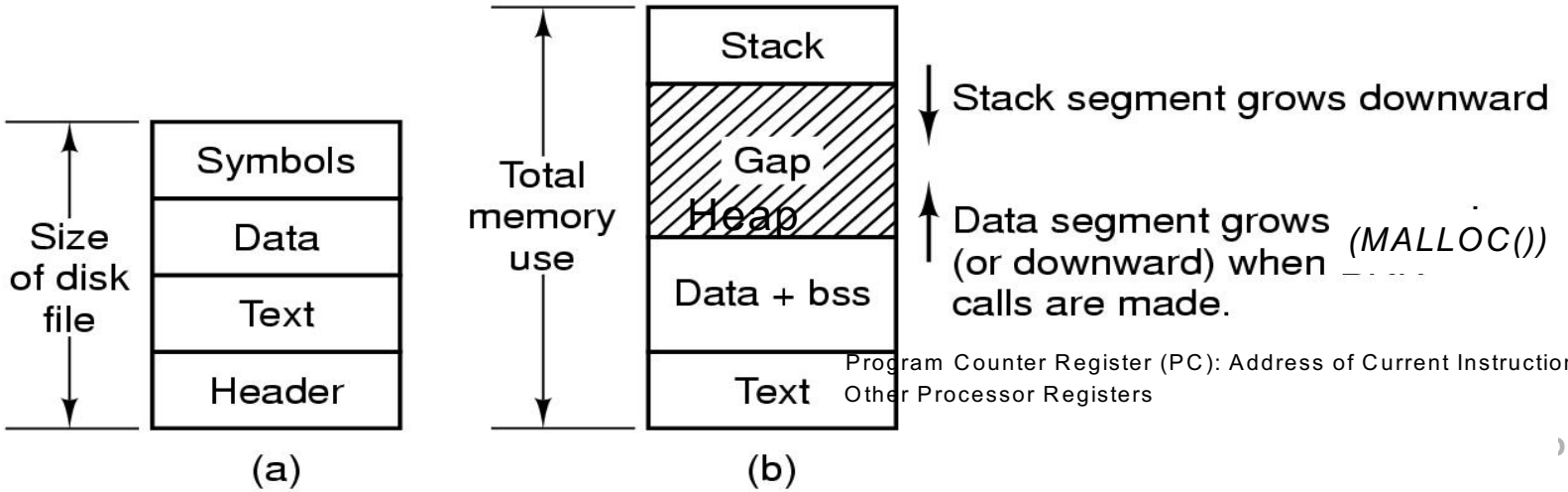
- Μια διεργασία λοιπόν **αναπαριστάται** από ένα **πίνακα bytes**, ο οποίος αποθηκεύει ότι έχει σχέση με την διεργασία (**process memory space**) π.χ.,
  - **Κώδικα** (οι εντολές του προγράμματος)
  - **Δεδομένα** (π.χ., σταθερές, κτλ.)
  - **Στοίβα** (π.χ., τιμές μεταβλητών συναρτησ..)
  - **Σωρός** (επιτρέπει την δυναμική προσθήκη επιπλέον μνήμης στο πρόγραμμα – Διάλεξη 9)
- Κάθε θέση του πίνακα έχει μια διεύθυνση.



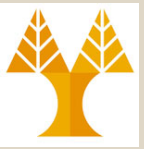
Ανατομία Προγράμματος



- Προτού δούμε περισσότερα για την **οργάνωση προγράμματος** ας δούμε λίγο πως **αναπαρίσταται** ένα πρόγραμμα στη μνήμη (**διεργασία ή process**).
- Αρχικά το **εκτελέσιμο αρχείο** (στην δευτερεύουσα μνήμη) έχει μια συγκεκριμένη δομή (**αριστερό σχήμα**) όπως αυτή διαμορφώνεται μετά την **μεταγλώττιση**.
- Όταν φορτωθεί στην **κύρια μνήμη** το πρόγραμμα τότε έχει την δομή που φαίνεται στα δεξιά.



Ανατομία Προγράμματος (στη Δευτερεύουσα Μνήμη)



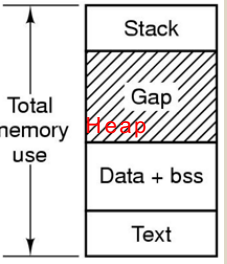
- Ανάλυση δομής εκτελέσιμων αρχείων (από την δευτερεύουσα μνήμη) με την εντολή size (bytes).

```
$ size /bin/bash /bin/tcsh  /bin/sort /bin/ls ~/.profile
```

text	data	bss	dec	hex	filename
600070	22872	17044	639986	9c3f2	**/bin/bash**
299571	14896	196864	511331	7cd63	**/bin/tcsh**
38946	888	3940	43774	aafe	**/bin/sort**
65254	1108	872	67234	106a2	**/bin/ls**

```
size: /home/faculty/dzeina/.profile: File format not recognized
```

- Η τέταρτη και πέμπτη στήλη είναι το σύνολο των πρώτων τριών στηλών σε δεκαδική και δεκαεξαδική μορφή.
- Η εντολή δεν εκτελείται πάνω σε αρχεία δεδομένων



Ανατομία Προγράμματος (στη Κύρια Μνήμη)

a) **Text:** Οι εντολές μηχανής τις οποίες θα εκτελέσει το CPU. Συνήθως αυτή η περιοχή (read-only) μπορεί να μοιραστεί μεταξύ διεργασιών.

b) **(Initialized) Data Segment:** Περιέχει τιμές οι οποίες έχουν ανατεθεί ρητά στον πηγαίο κώδικα π.χ. **int max = 99;** Σταθερές και αρχικοποιημένα static οπουδήποτε**, π.χ., static int max = 99;**

c) **(Uninitialized) Data Segment (BSS - Block-Started-By-Symbol):** Περιέχει καθολικά δηλωμένες μεταβλητές (εκτός των συναρτήσεων) **π.χ. long sum[1000]; και static μη-αρχικοποιημένες μεταβλητές.** Οι τιμές σε αυτή την περιοχή **αρχικοποιούνται από τον πυρήνα σε 0 κατά την διάρκεια της εκτέλεσης**. Οι μεταβλητές **μέσα στις συναρτήσεις** δεν δεσμεύονται εδώ αλλά στην στοίβα του προγράμματος κατά την κλήση της κάθε συνάρτησης.

d) **Stack (Στοίβα Προγράμματος):** Κάθε κλήση συνάρτησης σε ένα πρόγραμμα φυλάσσει όλες τις πληροφορίες (μεταβλητές, διεύθυνση επιστροφής κτλ.) στη στοίβα του προγράμματος. Οι μεταβλητές δεσμεύονται αυτόματα (automatic variables). Το **μέγεθος** της στοίβας έχει άμεση σχέση με το βάθος της **υποστηριζόμενης αναδρομής**!

e) **Heap (Σωρός Προγράμματος):** Οτιδήποτε δεσμεύει δυναμικά μνήμη (π.χ., με τις εντολές malloc(), calloc(), realloc() που θα δούμε στη διάλεξη 9 φυλάγετε εδώ). Δεδομένα στην σωρό μπορεί να προσπελαστούν μέσω της διεύθυνσης τους.

Ανατομία Προγράμματος (στη Κύρια Μνήμη)

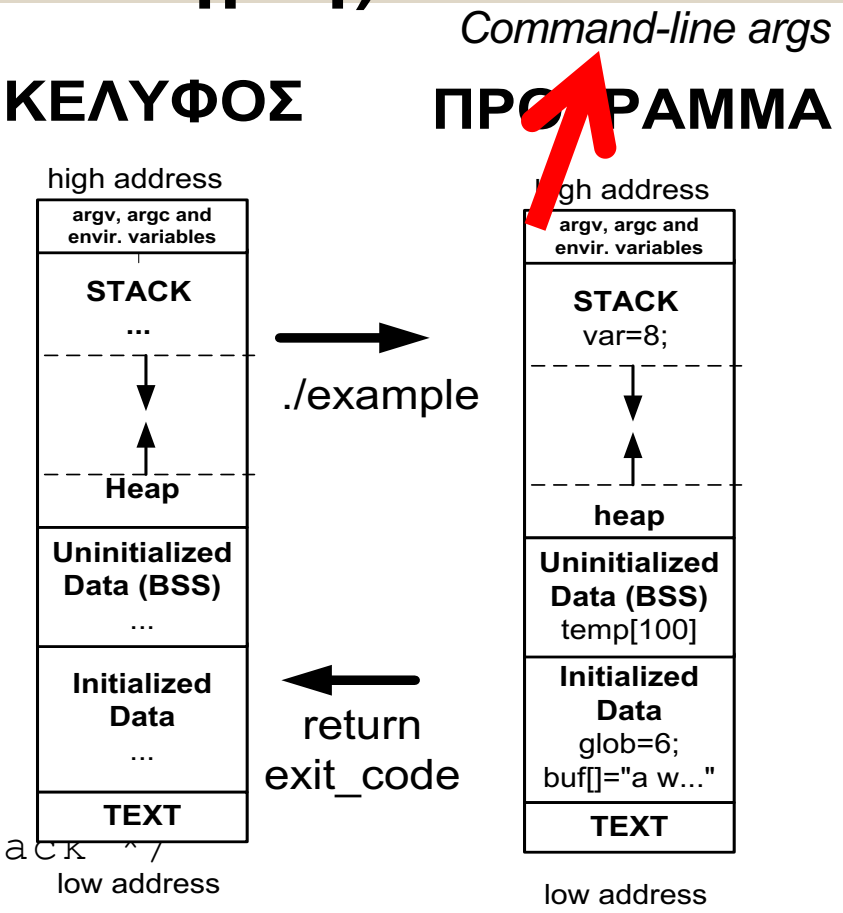
```
#include <stdio.h>

/* variable in initialized data */
int glob = 6;

/* variable in initialized data */
char buf[] = "a write to stdout\n";

/* variable in uninitialized data*/
char temp[100];


int main(void) {
    /* automatic variable in the stack */
    int var = 88;
    printf("%s", buf);
    return 0;
}
```



Ανατομία Προγράμματος (Όρια Προγραμμάτων)

Παρουσίαση Ορίων: **ulimit –a**

```
Terminal — ssh — 77×24
1)dzeina@b103ws1> ulimit -a
core file size          (blocks, -c) 0
data seg size           (kbytes, -d) unlimited
scheduling priority             (-e) 0
file size               (blocks, -f) unlimited
pending signals                 (-i) 29944
max locked memory       (kbytes, -l) 64
max memory size         (kbytes, -m) unlimited
open files                      (-n) 1024
pipe size            (512 bytes, -p) 8
POSIX message queues     (bytes, -q) 819200
real-time priority              (-r) 0
stack size              (kbytes, -s) 10240
cpu time               (seconds, -t) unlimited
max user processes              (-u) 1024
virtual memory          (kbytes, -v) unlimited
file locks                      (-x) unlimited
```

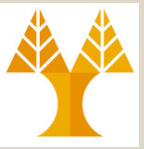
Ανάθεση Heap 10MB: **ulimit –m 102400**

Ανάθεση Stack 10MB: **ulimit –s 102400**

(works on laboratory machines!)

Reset to default: Επανεκκίνηση Τερματικού

Περιεχόμενο Διάλεξης



- **Οργάνωση Προγράμματος (Κεφ. 10)**
  - *Τοπικές Μεταβλητές* (αυτόματες, static), *Εξωτερικές Μεταβλητές* (block/file scope), Μπλοκ { }, *Εμβέλεια Μεταβλητών*
  - *Οργάνωση Προγράμματος* σε ένα αρχείο κώδικα.
- **Ανατομία Προγράμματος**
  - Εισαγωγή: Διεργασίες, Μνήμη & Διευθύνσεις
  - Ανατομία Προγράμματος
    - Στην Δευτερεύουσα Μνήμη (Disk)
    - Στην Κύρια Μνήμη (RAM)
- **(Προκαταρτική) Είσοδος/Έξοδος από Αρχεία**
  - Συναρτήσεις Βιβλιοθήκης fopen, fread, fwrite, παράδειγμα με ψευδοτυχαίους αριθμούς

Είσοδος/Έξοδος από Αρχεία

Εισαγωγική Επισήμανση

- Για **επεξεργασία αρχεία** θα χρειαστούμε **δείκτες** (*) τους οποίους θα δούμε στις **ερχόμενες 2 διαλέξεις**.
- Για να μπορέσετε **μέχρι τότε** να δουλεύετε με αρχεία, στα πλαίσια της **AS1**, θα σας **υποδειχτεί** τώρα ο **βασικός τρόπος** και η **αναλυτική επεξήγηση** θα γίνει **αργότερα** στην Διάλεξη 9
  - Όπου θα δούμε το File I/O σε βάθος.

Είσοδος/Έξοδος από Αρχεία (Άνοιγμα/Κλείσιμο)

Άνοιγμα/Κλείσιμο Αρχείου fopen (<stdio.h>)

FILE * fopen(char *filename, char *mode);

- Η παράμετρος *filename* υποδεικνύει το όνομα του αρχείου που επιθυμούμε να ανοίξουμε
- Το Mode υποδεικνύει το είδος της πράξης (π.χ. Read, write, read-write, etc)

```
FILE *fp = NULL;
fp = fopen("myfile.txt", "r");
...
fclose(fp);
```

Τύπος Ανοίγματος

(εδώ READ)

Όνομα Αρχείου

Οντότητα Διαχείρισης Αρχείου. (file pointer)

Είσοδος/Εξοδος από Αρχεία (Τρόπος Προσπέλασης)

Η παράμετρος mode υποδεικνύει τον τρόπο με τον οποίο θέλουμε να προσπελάσουμε το αρχείο

Mode	**Σημασία**
r	Μόνο **Ανάγνωση**. Αν το αρχείο δεν υπάρχει, επιστρέφεται NULL
w	Μόνο **Εγγραφή**. Αν το αρχείο δεν υπάρχει δημιουργείται, αν υπάρχει τα περιεχόμενά του καταστρέφονται.
a	**Προσθήκη**. Αν το αρχείο δεν υπάρχει, δημιουργείται.
r+	**Ανάγνωση** και **εγγραφή**. Αν το αρχείο δεν υπάρχει, επιστρέφεται NULL.
w+	**Ανάγνωση** και **εγγραφή**. Αν το αρχείο δεν υπάρχει δημιουργείται, αν υπάρχει τα περιεχόμενά του καταστρέφονται.
a+	**Προσθήκη** και **ανάγνωση**. Αν το αρχείο δεν υπάρχει, δημιουργείται.

Είσοδος/Εξοδος από Αρχεία (Ανάγνωση / Εγγραφή Τιμών)

```
#include <stdio.h>  // fopen, fscanf, fprintf, fclose, printf
#include <stdlib.h> // EXIT_FAILURE
char filename[]="test.txt";

int main()
{
    FILE *fp = NULL;
    int a;

    if ((fp = fopen(filename, "r")) == NULL)        {
        printf("Error: Unable to open %s\n", filename);
        exit(EXIT_FAILURE);
    }

    // Read an integer from the file
    fscanf(fp, "%d", &a);
    printf("%d",a); // ή fprintf(fp, "Duplicate:%d", a); Με "w" | "a" mode

    // Close the file
    fclose(fp);

    return 0;

}
```

Για χρήση **άλλων τύπων** χρησιμοποιούνται αντίστοιχα ορίσματα με την **printf()**

Είσοδος/Εξοδος από Αρχεία (Ανάγνωση / Εγγραφή Τιμών)

```
#include <stdio.h>  // fopen, fscanf, fprintf, fclose, printf
#include <stdlib.h> // EXIT_FAILURE
#include <time.h>   // time
char filename[]="test.txt";

int main()
{
   FILE *fp = NULL;
   int i;
   srand(time(NULL));

   if ((fp = fopen(filename, "w")) == NULL)          {
      printf("Error: Unable to open %s\n", filename);
      exit(EXIT_FAILURE);
   }

   for (i=0;i<10;i++) {
      fprintf(fp, "%d\n", rand() % 1000 + 1);
   }

   // Close the file
   fclose(fp);
   return 0;
}
```

Τι μπορεί να κάνει το ακόλουθο πρόγραμμά;

Γιατί χρειάζεται μια συνάρτηση όπως την srand();

Χαρακτήρας Τερματισμού Γραμμής End of Line (EOL)

- *Το newline character είναι διαφορετικό σε διαφορετικά Λειτουργικά Συστήματα!*
  - **LF (\n):** Multics, Unix and Unix-like systems (Linux, OS X, FreeBSD, AIX, Xenix, etc.), BeOS, Amiga, RISC OS, and other
  - ***CR+LF (\r\n):* Microsoft Windows**, *DOS (MS-DOS, PC DOS, etc.), DEC TOPS-10, RT-11, CP/M, MP/M, Atari TOS, OS/2, Symbian OS, PalmOS, Amstrad CPC, and most other early non-Unix and non-IBM Oses*
  - **CR (\r):** Commodore 8-bit machines, Acorn BBC, ZX Spectrum, TRS-80, Apple II family, Oberon, **Mac OS up to version 9**, MIT Lisp Machine and OS-9
  - 0x9B (Atari 8-bit machines), LF+CR (RISC OS spooled text)
- `dos2unix` ***and*** `unix2dos` ***- DOS/MAC to UNIX text file format converter commands.***

Χαρακτήρας Τερματισμού Αρχείου; End of File (EOF)

- Δεν χρησιμοποιείται σε αρχεία αυτός ο χαρακτήρας!
- Το EOF==-1 και υποδηλώνει απλά ότι το αρχείο που διαβάζουμε δεν έχει άλλο χαρακτήρα, έτσι η βιβλιοθήκη stdio.h του glibc επιστρέφει -1

```
while (fscanf(in, "%c", &symbol) != EOF) { …}
```

- Στον πίνακα ASCII υπάρχει ο χαρακτήρας 25 EM (end of medium) ο οποίος χρησιμοποιήθηκε σε αρχεία τύπου Windows στο παρελθόν, αλλά πλέον δεν υπάρχει.
- Αργότερα θα μάθουμε πως με το `hexdump -C file.any` θα μπορούμε να βλέπουμε την byte ακολουθία οποιουδήποτε αρχείου και εκεί θα φανεί ακριβώς τι περιέχεται στο κάθε αρχείο (Άσκηση 4)