

# ΕΠΛ232 – Προγραμματιστικές Τεχνικές και Εργαλεία

Εργαλεία I (μέρος: Κεφάλαια 19.1-19.2, 24.1-24.2 ΚΝΚ-2ΕΔ)

Τμήμα Πληροφορικής, Πανεπιστήμιο Κύπρου

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

# Περιεχόμενο Διάλεξης 16

- **Μεγάλης Κλίμακας Λογισμικό (Large-scale SW)**
  - Μέγεθος Λογισμικού και Προβλήματα
  - Κύκλος Ανάπτυξης Λογισμικού
- **Μονάδες Προγράμματος (Modules)**
  - Εισαγωγή, Χαρακτηριστικά, Πλεονεκτήματα
  - Συνοχή (Cohesion) και Διασύνδεση (Coupling)
  - Παραδείγματα Μονάδων : glib-c, gzip, coreutils
  - Τύποι Μονάδων, Μονάδες και Βιβλιοθήκες
- **Έλεγχοι και Ανάλυση Κώδικα**
  - Επιβεβαίωση Ισχυρισμών `assert()`, `Cunit/Junit`
  - Έννοιες: Στατική & Δυναμική Ανάλυση Κώδικα
  - Εργαλεία: `Valgrind`, `gprof`, `JProfiler` (JAVA) και διασύνδεση με Eclipse



# Μεγάλης Κλίμακας Λογισμικό (Μέγεθος)

- Τα περισσότερα **πλήρη προγράμματα (full-featured programs)**, είναι χιλιάδες **Γραμμές Πηγαίου Κώδικα (Source Lines of Code - SLOC)**.
- **Παραδείγματα**
  - **gz124src.zip**: Το gzip εργαλείο σε UNIX (**42 αρχεία με 6,972 SLOC!**)
    - <http://www.gzip.org/>
  - **perl-5.14.2.tar.gz**: Ο μεταφραστής και οι βασικές βιβλιοθήκες της γλώσσας PERL (**3241 αρχεία και 667,323 SLOC!!**)
    - <http://www.perl.org/get.html>
  - **glibc-2.14.tar.gz**: ο κώδικας όλων των βιβλιοθηκών της γλώσσας C (**8552 αρχεία και 1,775,440 SLOC!!!**)
    - <http://ftp.gnu.org/gnu/glibc/>



# Μεγάλης Κλίμακας Λογισμικό (Μέγεθος)

- Παράδειγμα Εύρεσης SLOC με το εργαλείο **CLOC (Count Lines of Code\*)**

- Windows: <http://sourceforge.net/projects/cloc/files/cloc/v1.55/cloc-1.55.exe>
- Linux (με Perl): <http://sourceforge.net/projects/cloc/files/cloc/v1.55/cloc-1.55.pl>

```
$ ./cloc-1.55.pl gzip124/  
    89 text files.  
    82 unique files.  
    47 files ignored.
```

**Απλά τοποθετήστε το .pl ή .exe στον κατάλογο που θέλετε και μετά  
"./cloc-1.55.pl <directory> " ή απλά "./cloc-1.55.pl gzip.tar.gz"**

```
http://cloc.sourceforge.net v 1.55 T=0.5 s (84.0 files/s, 21174.0 lines/s)
```

Language	files	blank	comment	code
C	22	1010	1910	5129
C/C++ Header	12	161	186	680
Bourne Shell	2	85	99	670
Assembly	2	69	92	458
DOS Batch	1	0	0	18
Perl	1	1	2	12
Teamcenter def	2	0	0	5
SUM:	42	1326	2289	6972

- Το CLOC υπολογίζει τις κενές γραμμές, τις γραμμές σχολίου και τις φυσικές γραμμές πηγαίου κώδικα σε διάφορες γλώσσες προγραμματισμού.
- Υπολογίστε επίσης τις διαφορές στις γραμμές του κώδικα και τα σχόλια μεταξύ δύο εκδόσεων του κώδικα.



# Μεγάλης Κλίμακας Λογισμικό (Μέγεθος)

- Παραδείγματα SLOC γνωστών Λειτουργικών Συστημάτων.
  - Σύνολο πολλών πλήρων υπο-προγραμμάτων

Year	Operating System	SLOC (Million)
1993	Windows NT 3.1	4-5 <sup>[1]</sup>
1994	Windows NT 3.5	7-8 <sup>[1]</sup>
1996	Windows NT 4.0	11-12 <sup>[1]</sup>
2000	Windows 2000	more than 29 <sup>[1]</sup>
2001	Windows XP	45 <sup>[2]</sup>
2003	Windows Server 2003	50 <sup>[1]</sup>

Πηγή: [http://en.wikipedia.org/wiki/Source\\_lines\\_of\\_code](http://en.wikipedia.org/wiki/Source_lines_of_code)

Year	Operating system	SLOC (million)
2000	Debian 2.2	55-59 <sup>[7][8]</sup>
2002	Debian 3.0	104 <sup>[8]</sup>
2005	Debian 3.1	215 <sup>[8]</sup>
2007	Debian 4.0	283 <sup>[8]</sup>
2009	Debian 5.0	324 <sup>[8]</sup>
2012	Debian 7.0	419 <sup>[9]</sup>
2009	OpenSolaris	9.7
	FreeBSD	8.8
2005	Mac OS X 10.4	86 <sup>[10][n 1]</sup>
1991	Linux kernel 0.01	0.010239
2001	Linux kernel 2.4.2	2.4 <sup>[6]</sup>
2003	Linux kernel 2.6.0	5.2
2009	Linux kernel 2.6.29	11.0
2009	Linux kernel 2.6.32	12.6 <sup>[11]</sup>
2010	Linux kernel 2.6.35	13.5 <sup>[12]</sup>
2012	Linux kernel 3.6	15.9 <sup>[13]</sup>
2015-06-30	Linux kernel pre-4.2	20.2 <sup>[14]</sup>



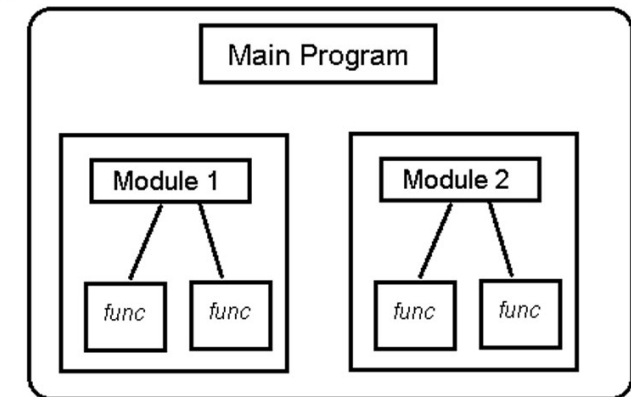
# Περιεχόμενο Διάλεξης 16

- **Μεγάλης Κλίμακας Λογισμικό (Large-scale SW)**
  - Μέγεθος Λογισμικού και Προβλήματα
  - Κύκλος Ανάπτυξης Λογισμικού
- **Μονάδες Προγράμματος (Modules)**
  - Εισαγωγή, Χαρακτηριστικά, Πλεονεκτήματα
  - Συνοχή (Cohesion) και Διασύνδεση (Coupling)
  - Παραδείγματα Μονάδων : glib-c, gzip, coreutils
  - Τύποι Μονάδων, Μονάδες και Βιβλιοθήκες
- **Έλεγχοι και Ανάλυση Κώδικα**
  - Επιβεβαίωση Ισχυρισμών `assert()`, Cunit/Junit
  - Έννοιες: Στατική & Δυναμική Ανάλυση Κώδικα
  - Εργαλεία: Valgrind, gprof, JProfiler (JAVA) και διασύνδεση με Eclipse



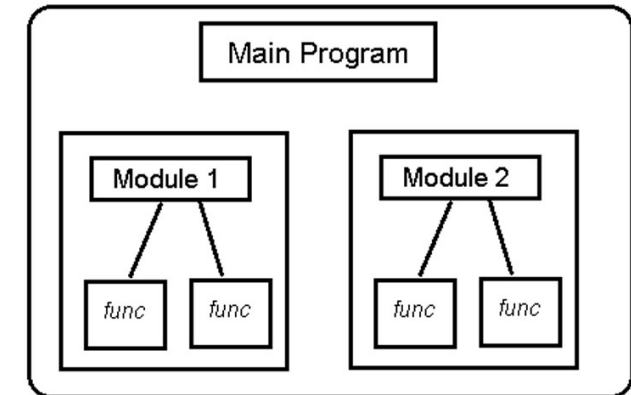
# Μονάδες – Modules (Εισαγωγή)

- Η αρθρωτότητα στον προγραμματισμό ενθαρρύνει τον διαχωρισμό της λειτουργικότητας ενός προγράμματος σε ξεχωριστές και ανεξάρτητες μονάδες, έτσι ώστε κάθε μονάδα να έχει όλα όσα απαιτούνται για να εκτελέσει από μόνη της.
- Τα στοιχεία στη διασύνδεση μιας λειτουργικής μονάδας μπορούν να εντοπιστούν και σε άλλες ενότητες του προγράμματος.
- Ο αρθρωτός προγραμματισμός σήμερα ασχολείται με την αποσύνθεση υψηλού επιπέδου του κώδικα ενός ολόκληρου προγράμματος σε λειτουργικές και ανεξάρτητες μονάδες.



# Μονάδες – Modules (Επανάληψη)

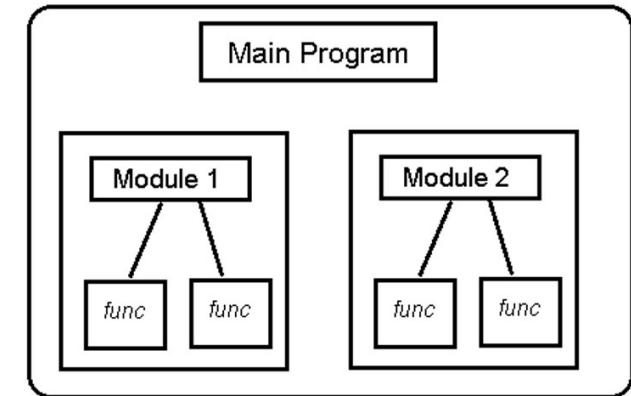
- Δυνατότητα επαναχρησιμοποίησης κώδικα
  - Η αρθρωτότητα επιτρέπει στους προγραμματιστές να χρησιμοποιούν τον ίδιο κώδικα στη λειτουργική μονάδα ξανά και ξανά, σε διαφορετικές θέσεις του προγράμματος. Η δημιουργία μονάδων ή κλάσεων το εξασφαλίζει αυτό.
- Εύκολος εντοπισμός σφαλμάτων
  - Ο εντοπισμός σφαλμάτων σε ένα πρόγραμμα με εκατομμύρια γραμμές κώδικα αποτελεί πρόκληση. Εξασφαλίζοντας πως κάθε εργασία έχει διακριτή μονάδα κάνει τη διαδικασία πολύ ευκολότερη. Μια ελαττωματική λειτουργία μπορεί εύκολα να ελεγχθεί για σφάλματα όταν βρίσκεστε σε μια ανεξάρτητη λειτουργική μονάδα.
- Απλότητα και ευκολία συντήρησης
  - Ο αριθμός των γραμμών του κώδικα μειώνεται, αποφεύγοντας την επανάληψη παρόμοιου κώδικα.
- Οργάνωση
  - Η ανάγνωση ενός προγράμματος είναι ευκολότερη και πιο κατανοητή. Αυτό κάνει τον προγραμματιστή πιο οργανωμένο στην κωδικοποίηση του, το οποίο βοηθά όταν γίνεται αναφορά. Αυτό κάνει επίσης άλλους προγραμματιστές να μπορούν να διαβάσουν τον κώδικα ευκολότερα.
- Ενθαρρύνει την ομαδική εργασία και τη συνεργασία
  - Η αρθρωτή σχεδίαση σε μια ανάπτυξη προγράμματος βελτιώνει τις προσπάθειες συνεργασίας από διαφορετικούς προγραμματιστές. Αυτό χτίζει το ομαδικό πνεύμα που απαιτείται για την ολοκλήρωση ενός μεγάλου έργου που απαιτεί χιλιάδες γραμμές κώδικα.





# Μονάδες – Modules (Εισαγωγή)

- Ζητήματα που προκύπτουν κατά τη σύνταξη ενός μεγάλου προγράμματος:
  - Στυλ
  - Τεκμηρίωση
  - Συντήρηση
  - Σχεδίαση

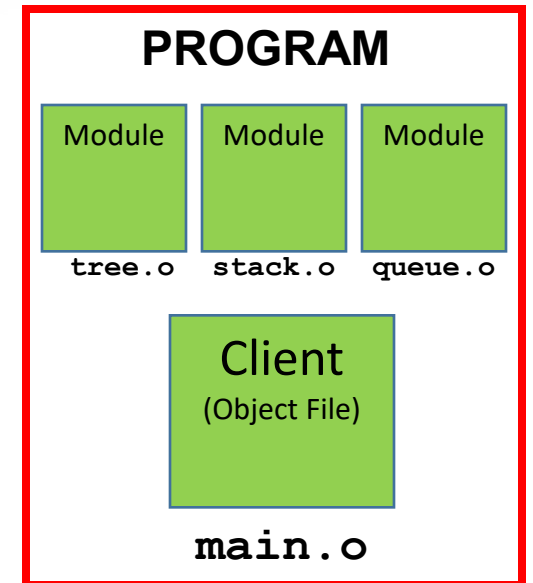


# Μονάδες – Modules (Εισαγωγή)

- Όπως αναφέραμε ήδη, είναι ευκολότερο να βλέπουμε ένα πρόγραμμα ως μια συλλογή από **ανεξάρτητες μονάδες (modules)**.

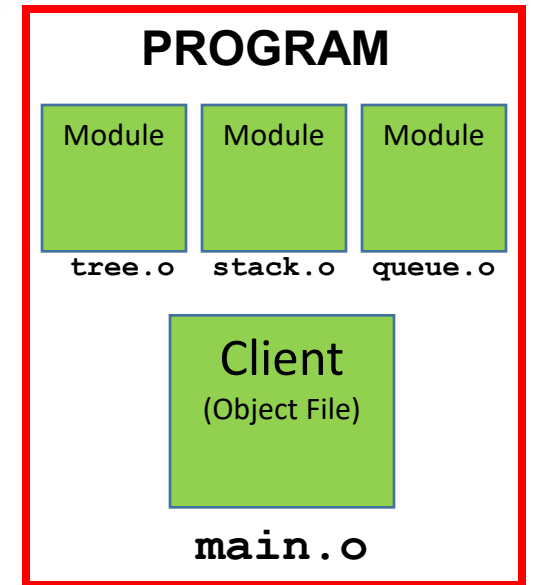
• **Μονάδα Προγράμματος (Module):** Συλλογή από **"υπηρεσίες" (συναρτήσεις)** οι οποίες είναι διαθέσιμες σε άλλα μέρη του προγράμματος, τα οποία ονομάζονται **χρήστες (clients)**.

- **Ορολογία C: Module ή Object** με `.o` extension (κώδικας μηχανής που προκύπτει από την μεταγλώττιση)
- **Ορολογία JAVA: Class ή Object** με `.class` extension (ενδιάμεσος κώδικας που προκύπτει από την μεταγλώττιση – απαιτεί JRE για εκτέλεση – μπορεί να γίνει decompile, π.χ., ψάξε `jad` ή `JD` στο google.)



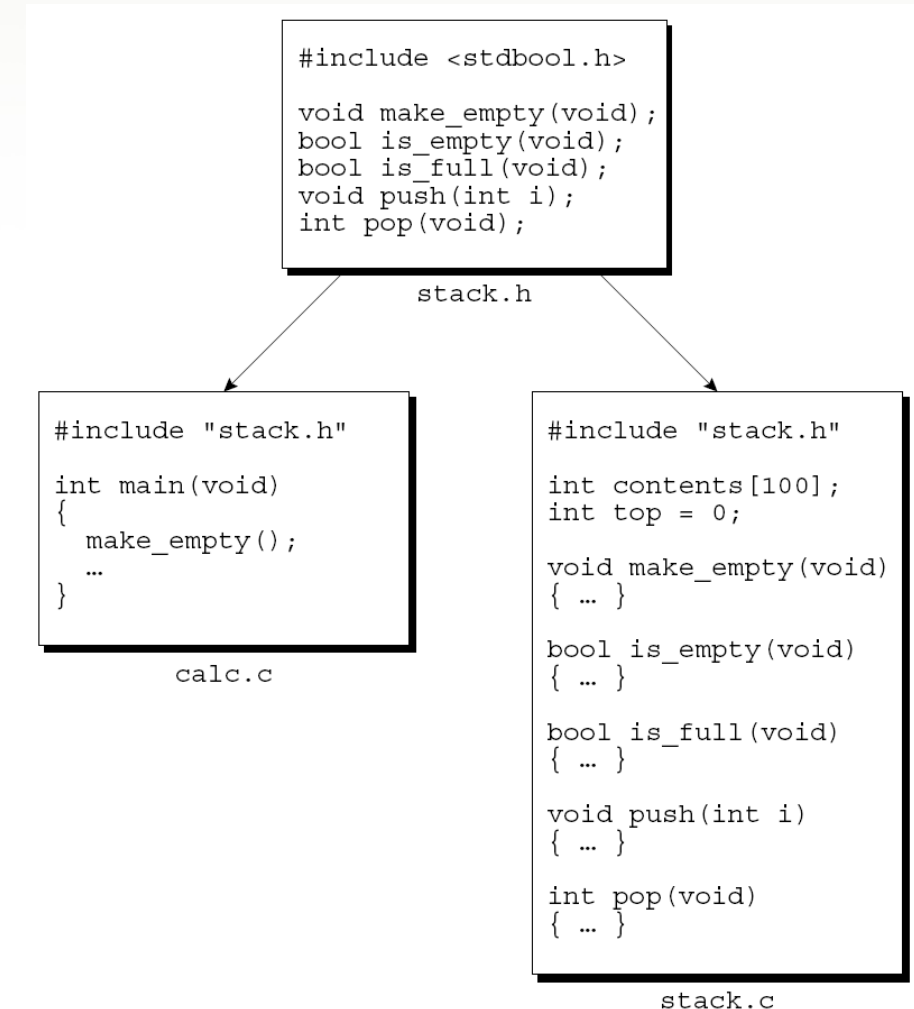
# Μονάδες – Modules (Χαρακτηριστικά)

- Κάθε module έχει μια **διεπαφή (interface)**, η οποία περιγράφει τις **διαθέσιμες "υπηρεσίες"** του module. Η διεπαφή αυτή την ονομάστηκε **αρχείο κεφαλίδας**.
- Κάθε module έχει την **υλοποίηση (implementation)**. Η υλοποίηση αυτή ονομάστηκε **πηγαίος κώδικας**.
- Κάθε Module **αναπτύσσεται ξεχωριστά** από τα υπόλοιπα modules.
- Κάθε Module **μπορεί να αναπτύσσεται από διαφορετικούς προγραμματιστές** (ή ζεύγη προγραμματιστών – extreme programming)
- Κάθε Module **δοκιμάζεται και συντηρείται ξεχωριστά** από τα άλλα module (με τους οδηγούς δοκιμής που διερευνούν όλα τα μονοπάτια εκτέλεσης ενός προγράμματος)



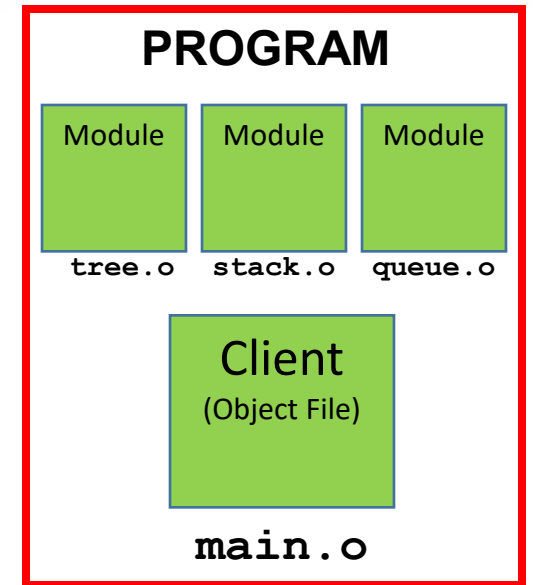
# Μονάδες – Modules (Χαρακτηριστικά)

- Η `calc.c`, περιέχει την `main` συνάρτηση
- Μια μονάδα `stack`, αποθηκεύεται στην `stack.h` και `stack.c`
  - `calc.c` είναι ο *χρήστης (client)* της μονάδας `stack`.
  - `stack.h` είναι η *διεπαφή (interface)* της μονάδας `stack`.
  - `stack.c` είναι η *υλοποίηση (implementation)* της μονάδας.



# Μονάδες – Modules (Πλεονεκτήματα)

- **Πλεονεκτήματα διάσπασης ενός προγράμματος σε μονάδες:**
  - **Αφαιρετικότητα (Abstraction)**
    - Γνωρίζουμε τι κάνει **χωρίς** να ξέρουμε πως.
  - **Επαναχρησιμοποίηση Κώδικα (Reusability)**
    - Κάθε μονάδα (π.χ., στοίβα) που παρέχει υπηρεσίες μπορεί εύκολα να είναι χρήσιμη σε άλλους πελάτες.
  - **Συντήρηση Κώδικα (Maintainability)**
    - Ένα **σφάλμα** βρίσκεται σε μια μόνο μονάδα, επομένως διευκολύνεται ο **εντοπισμός & διόρθωση** του προβλήματος.
    - Η **επανα-μεταγλώττιση** γίνεται γρήγορα (ένα αρχείο).
    - Μπορούμε να αλλάξουμε την υλοποίηση μιας μονάδας χωρίς να αλλάξει το υπόλοιπο πρόγραμμα (εφόσον η διεπαφή δεν αλλάζει πρότυπο)



# Μονάδες – Modules (Ερωτηματικά)

- Κατά τη **σχεδίαση** ενός **αρθρωτού μεγάλου** προγράμματος προκύπτουν κάποια **ερωτήματα**:
  - Ποιες και πόσες **μονάδες** πρέπει να έχει ένα πρόγραμμα;
  - Τι **υπηρεσίες** πρέπει να **προσφέρει** μια μονάδα;
  - Ποιες **υπηρεσίες** να είναι **εσωτερικές** και ποιές **εξωτερικές**;
  - Τι **αλληλεξαρτήσεις** πρέπει να έχουν οι μονάδες;
- Σίγουρα, θα έχετε **προβληματιστεί** για τα **ερωτήματα** αυτά, από την **έως τώρα τριβή** σας με τις **εργασίες**.

# Cohesion and Coupling

## Συνοχή και Διασύνδεση

- Σε ένα καλά **σχεδιασμένο πρόγραμμα**, οι μονάδες πρέπει να έχουν τα ακόλουθα χαρακτηριστικά:
  - **Ψηλή Συνοχή (High cohesion)**: Τα **στοιχεία** (συναρτήσεις, δεδομένα, κτλ) κάθε μονάδας πρέπει να **συσχετίζονται** όσο το **δυνατό περισσότερο**
  - **Χαμηλή Διασύνδεση (Low coupling)**: Τα **Modules** πρέπει να είναι όσο πιο **ανεξάρτητα** γίνεται (χωρίς δηλαδή να κάνει το ένα **include** το άλλο).
- Ας δούμε τώρα κάποια πραγματικά παραδείγματα από τα προγράμματα
  - glib-c (βιβλιοθήκες gnuC), gzip (εργαλείο C), coreutils (βασικές εντολές UNIX)



# Παραδείγματα

## Ψηλή Συνοχή και Χαμηλή Διασύνδεση

- Οι μονάδες του προγράμματος **gzip124** το οποίο υλοποιεί διάφορους **αλγόριθμους συμπίεσης / αποσυμπίεσης**
- Παρατηρούμε ότι έχουμε μια μονάδα (module) ανά αλγόριθμο, ένα πελάτη και άλλα
  - `gzip.c` (Client προγράμματος)
  - **`trees.c` / `unpack.c` (Huffman αλγόριθμος!)**
  - `zip.c` / `unzip.c` (PKZIP αλγόριθμος)
  - `lzw.c` / `unlzw.c` / `lzw.h` (LZW αλγ.)
  - `inflate.c` / `deflate.c` (LZ77 + Huffman)
  - **Αλλα:** `getopt` (command options), κτλ.
- Το αρχείο κεφαλίδας των πλείστων αρχείων είναι το `gzip.h` (κάθε συνάρτηση μπορεί να υλοποιείται από ξεχωριστό προγραμματιστή)
- Οι οδηγοί χρήσης μπορεί να είναι υλοποιημένοι εξωτερικά (εκτός κώδικα, το οποίο είναι εφικτό)

assembly





# Παραδείγματα

## Ψηλή Συνοχή και Χαμηλή Διασύνδεση

- Υποσύνολο των μονάδων για τα core utilities (βασικές εντολές) του UNIX
  - Διαθέσιμο μέσω : `http://ftp.gnu.org/gnu/coreutils/coreutils-8.31.tar.xz`
- **Τι παρατηρείται; Με ποια λογική έχουν δημιουργηθεί οι μονάδες;**
- Τα δικά μας προγράμματα μέχρι στιγμής ήταν "**πολύ μικρά**" για αυτό **σκόπιμα δημιουργήσαμε και μονάδες που ΔΕΝ θα χρειάζονταν στην πραγματικότητα.**
  - Σε μεγαλύτερα προγράμματα θα ήταν πιο ξεκάθαρο πώς να χωριστεί ο κώδικας σε μονάδες (όπως τα παραδείγματα)

```
chown-core.c
chown.c
chroot.c
cksum.c
comm.c
copy.c
cp-hash.c
cp.c
csplit.c
cut.c
date.c
dd.c
df.c
dircolors.c
dirname.c
du.c
echo.c
env.c
expand.c
expr.c
```



# Τύποι Μονάδων (Types of Modules)

- Μέχρι στιγμής είχαμε πει ότι μια **Μονάδα (Module)** θα **αποτελείται** από ένα .c και .h αρχείο.
- Τώρα είμαστε πιο **έτοιμοι (ώριμοι)** για να **χαλαρώσουμε** αυτό τον **περιορισμό**.
- Γενικότερα, μπορούμε να πούμε ότι οι **μονάδες** ανήκουν στις **ακόλουθες κατηγορίες**:
  - A. Συλλογές Δεδομένων (Data pools)
  - B. Βιβλιοθήκες (Libraries)
  - C. Αφαιρετικά Αντικείμενα (Abstract objects)
  - D. Αφηρημένος τύπος δεδομένων (Abstract data type) – **μη εξεταστέα**



# Τύποι Μονάδων (Types of Modules)

**A. Συλλογή Δεδομένων (*Data Pool*):** συσχετιζόμενες μεταβλητές ή/και σταθερές.

- Στη C, μια μονάδα αυτού του τύπου μπορεί να είναι ένα απλό αρχείο κεφαλίδας, π.χ., `<float.h>`, `<limits.h>` ή το `"def.h"`

**B. Βιβλιοθήκη (*Library*):** μια συλλογή από συσχετιζόμενες συναρτήσεις.

- `<string.h>` είναι η διεπαφή (interface) της βιβλιοθήκης που αφορά τις συναρτήσεις επεξεργασίας συμβολοσειρών.

Βιβλιοθήκη  
glibc-2.14/string

- h string.h
- c strlen.c
- c strncase\_l.c
- c strncase.c
- c strncat.c
- c strncmp.c
- c strncpy.c
- c strndup.c
- c strnlen.c
- c strpbrk.c
- c strrchr.c
- c strsep.c
- c strsignal.c
- c strspn.c
- c strstr.c
- c strtok\_r.c
- c strtok.c



# Μονάδες και Βιβλιοθήκες (Module and Library)

- Όπως αναφέραμε, η βιβλιοθήκη C (glibc), είναι η ίδια μια συλλογή από modules.
- Κάθε αρχείο **κεφαλίδας (header)** μέσα στη βιβλιοθήκη περιέχει την περιγραφή της **διεπαφής (interface)** για μια **μονάδα (module)** της βιβλιοθήκης., π.χ.,
  - `<stdio.h>` είναι το interface μιας μονάδας που περιέχει συναρτήσεις I/O.
  - `<string.h>` είναι το interface μιας μονάδας που περιέχει συναρτήσεις συμβολοσειρών.
  - κτλ

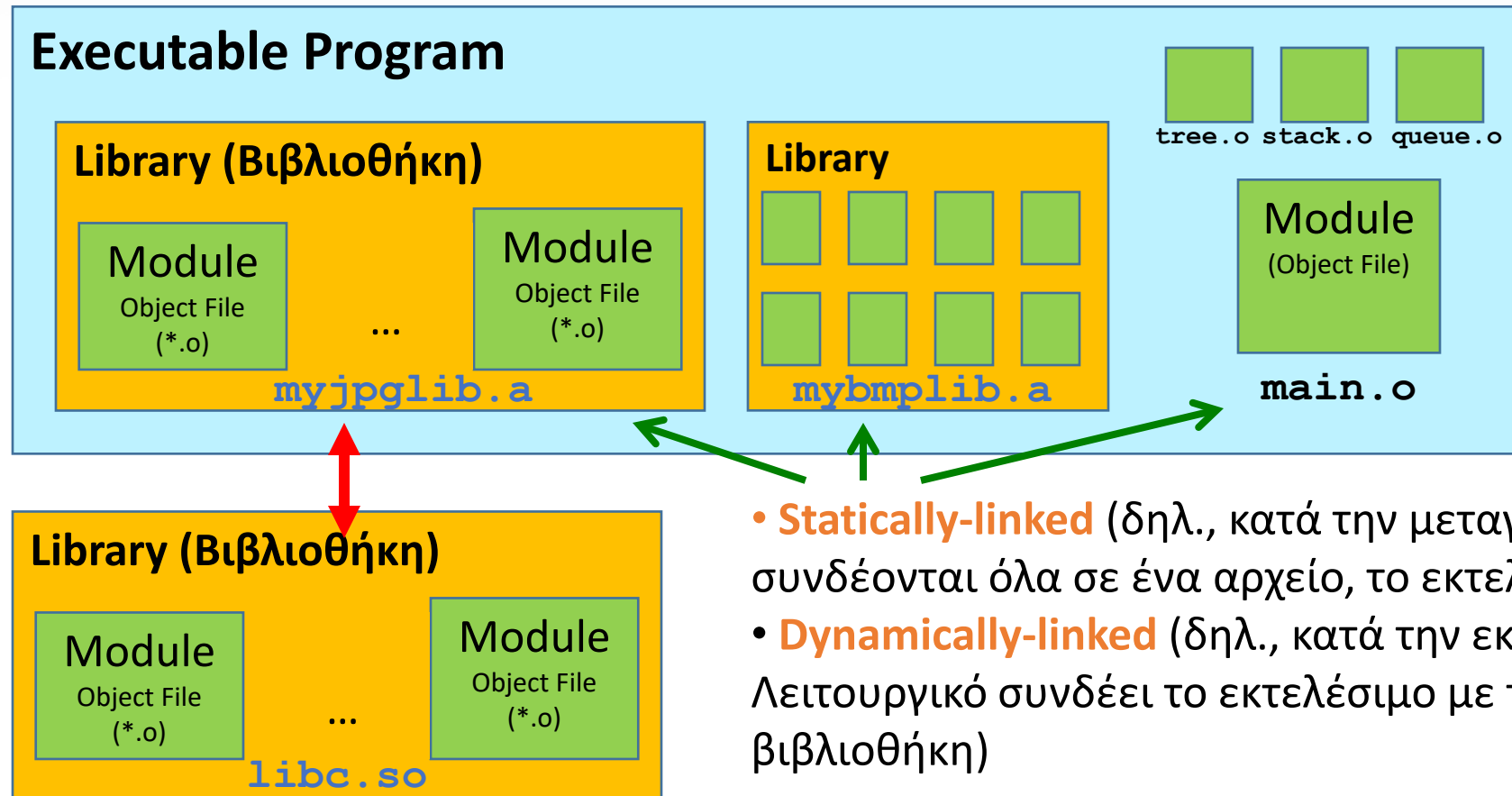
# Μονάδες και Βιβλιοθήκες (Module and Library)

- **Βιβλιοθήκη (Library):** Μια συλλογή από modules (αντίστοιχο της έννοιας των JAR αρχείων στη JAVA)
- Οι βιβλιοθήκες ομαδοποιούν πολλαπλά αρχεία από κώδικα γνωστών αντικειμένων σε ένα μόνο αρχείο. Συνήθως συναρτήσεις που μπορούν να μοιραστούν από περισσότερες από μία εφαρμογές χωρίζονται από τον πηγαίο κώδικα της εφαρμογής, και συγκεντρώνονται/ομαδοποιούνται σε μια βιβλιοθήκη.
- Το όφελος χρήσης βιβλιοθηκών είναι ότι κάθε αρχείο αντικειμένου δεν χρειάζεται να δηλώνεται κατά τη σύνδεση, επειδή ο προγραμματιστής μπορεί να αναφέρει τη συλλογική βιβλιοθήκη.
  - Αυτό απλοποιεί την πολλαπλή χρήση και κοινή χρήση στοιχείων λογισμικού μεταξύ εφαρμογών.
  - Τα στοιχεία που είναι μεγάλα μπορούν να δημιουργηθούν για **δυναμική χρήση**, έτσι η βιβλιοθήκη μπορεί να παραμείνει ξεχωριστή από το εκτελέσιμο μειώνοντας το μέγεθός της και ως εκ τούτου λιγότερο χώρο στο δίσκο χρησιμοποιείται για την εφαρμογή. Τα στοιχεία της βιβλιοθήκης καλούνται στη συνέχεια από διάφορες εφαρμογές για χρήση όταν χρειάζεται.



# Μονάδες και Βιβλιοθήκες (Module and Library)

- **Βιβλιοθήκη (Library):** Μια συλλογή από modules (αντίστοιχο της έννοιας των JAR αρχείων στη JAVA)



- **Statically-linked** (δηλ., κατά την μεταγλώττιση συνδέονται όλα σε ένα αρχείο, το εκτελέσιμο!)
- **Dynamically-linked** (δηλ., κατά την εκτέλεση το λειτουργικό συνδέει το εκτελέσιμο με την βιβλιοθήκη)



# Μονάδες και Βιβλιοθήκες (Module and Library)

Παράδειγμα Δημιουργίας / Σύνδεσης με Στατική Βιβλιοθήκης:

A) Δημιουργία Αντικειμενικών Αρχείων:

```
gcc -Wall -c f1.c f2.c
```

B) Δημιουργία Βιβλιοθήκης "mylibc.a" // ar -- create and maintain library archives

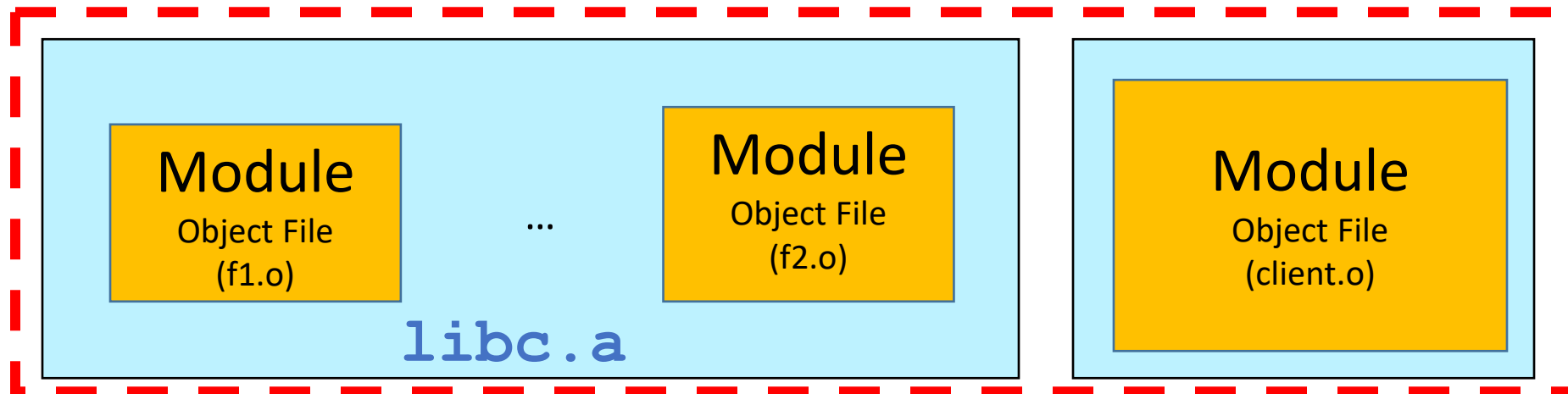
```
ar -cvq libc.a f1.o f2.o
```

Γ) Παρουσίαση Αρχείων στη Βιβλιοθήκη:

```
ar -t libc.a
```

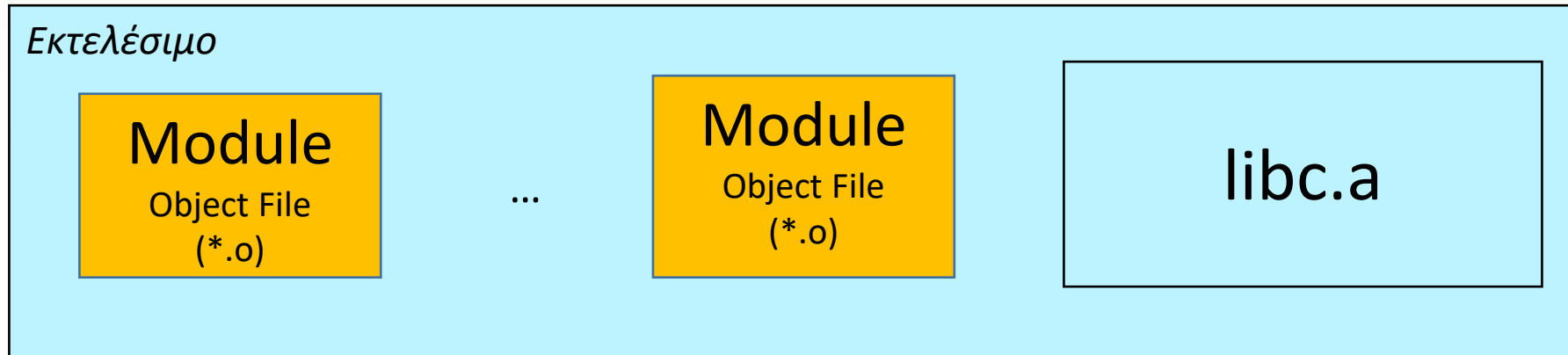
Δ) Σύνδεση προγράμματος πελάτη με τη Βιβλιοθήκη

```
gcc -o game client.c libc.a
```



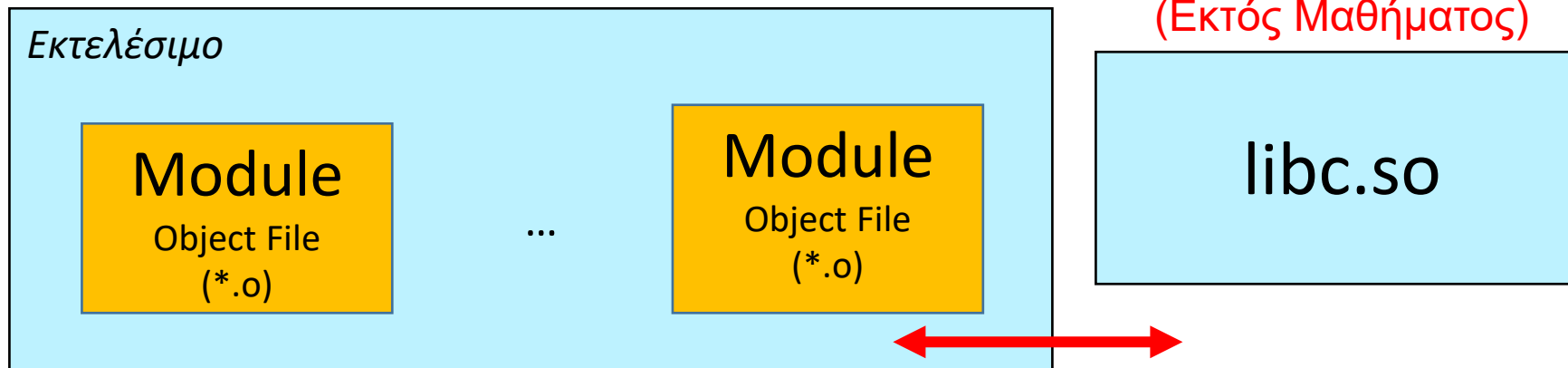
# Μονάδες και Βιβλιοθήκες (Module and Library)

## Σύνδεση με Στατική Βιβλιοθήκη (Static Library)



**Static libraries (.a):** Βιβλιοθήκη αντικειμένου κώδικα που συνδέεται και γίνεται μέρος της εφαρμογής.

## Σύνδεση με Δυναμική Βιβλιοθήκη (Dynamic Library)



**Dynamically linked shared object libraries (.so):** Οι βιβλιοθήκες πρέπει να είναι διαθέσιμες κατά τη φάση μεταγλώττισης/σύνδεσης. Τα κοινόχρηστα αντικείμενα δεν συμπεριλαμβάνονται στο εκτελέσιμο στοιχείο, αλλά συνδέονται με την εκτέλεση.

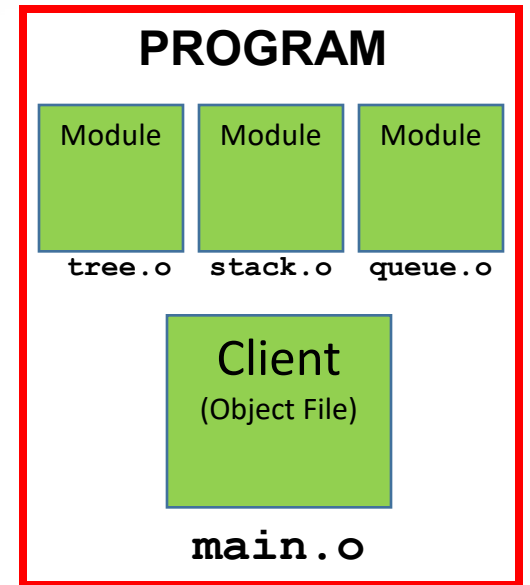
Example: <http://www.yolinux.com/TUTORIALS/LibraryArchives-StaticAndDynamic.html>





# Απόκρυψη Πληροφορίας (Information Hiding)

- Μια καλά σχεδιασμένη μονάδα **αποκρύπτει** πληροφορίες από τους πελάτες της (γνωστό ως **απόκρυψη πληροφορίας – information hiding**)
  - Π.χ., Οι πελάτες της στοίβας δε χρειάζεται να γνωρίζουν εάν η στοίβα αποθηκεύεται σε πίνακα, συνδεδεμένη λίστα ή αλλιώς.
- **Λόγοι Απόκρυψης:**
  - **Ασφάλεια (Security).** Εάν οι πελάτες δεν γνωρίζουν πως αποθηκεύονται τα δεδομένα εσωτερικά δεν μπορούν να την μεταβάλουν συνειδητά ή ασυνείδητα.
  - **Ευελιξία (Flexibility).** Η **αλλαγή** στα εσωτερικά ενός module δεν θα είναι δύσκολη από τον προγραμματιστή εφόσον το module εκθέτει (διαφημίζει) μόνο τη διεπαφή της!



# Απόκρυψη Πληροφορίας (Information Hiding)

- Στη C, το βασικό εργαλείο για απόκρυψη πληροφορίας είναι η χρήση του `static`
  - Μια **static μεταβλητή** με εμβέλεια αρχείου, είναι "ορατή" μόνο **εσωτερικά του αρχείου** (ούτε και οι πελάτες της μονάδας δεν το βλέπουν).
  - Μια **static συνάρτηση** μπορεί να καλείται απευθείας μόνο από άλλες **συναρτήσεις μέσα στο ίδιο αρχείο**.

```
def.h
#define PUBLIC // Ορατή Εκτός Αρχείου
#define PRIVATE static // Ορατή Εντός Αρχείου
```

- Η λέξη `static` έχει περισσότερες από μια χρήση στη C; `PRIVATE` καθιστά σαφές ότι το χρησιμοποιούμε για να επιβάλλουμε πληροφορίες που κρύβονται.



# Τύποι Μονάδων (Types of Modules)

**C. Αφηρημένο Αντικείμενο (Abstract Object):** Συλλογή από συναρτήσεις που εργάζονται πάνω σε κάποια δομή η οποία είναι εσωτερική (*hidden*).

- Στη διάλεξη 11 χρησιμοποιήσαμε το *static* μπροστά από τις εσωτερικές μεταβλητές της στοίβας (η οποία ήταν αφηρημένο αντικείμενο)

*stack.h*

```
static int sp = 0;  
static double val[100];
```

Προβλήματα;

Επόμενη Διαφάνεια ...

*main.c*

```
#include "stack.h"  
int main() {  
    STACK stack; int x = 1;  
    push(&stack, x); // OK  
    sp = 2; // ERROR
```



# Τύποι Μονάδων (Types of Modules)

- **Πρόβλημα 1:** Συγκρούσεις στα Ονόματα (π.χ., include "queue.h" και "stack.h" και τα δυο περιέχουν συνάρτηση top())
  - **Λύση:** Πρόθεμα στις συναρτήσεις stack\_top() και queue\_top().
- **Πρόβλημα 2:** Δεν θα μπορούσαμε να "κρύψουμε" τα δεδομένα του stack εάν είχαμε typedef
  - Οι αντικειμενοστραφείς γλώσσες (JAVA, C++, C#) υποστηρίζουν ευκολότερα τους Αφηρημένους Τύπους Δεδομένων (ΑΤΔ)

## stack.h

```
typedef struct {  
    int sp = 0;  
    double val[100];  
} STACK;
```

## main.c

```
#include "stack.h"  
int main() {  
    STACK stack; int x = 1;  
    push(&stack, x); // OK  
    stack.sp = 2;    // OK!!!
```



# Περιεχόμενο Διάλεξης 16

- **Μεγάλης Κλίμακας Λογισμικό (Large-scale SW)**
  - Μέγεθος Λογισμικού και Προβλήματα
  - Κύκλος Ανάπτυξης Λογισμικού
- **Μονάδες Προγράμματος (Modules)**
  - Εισαγωγή, Χαρακτηριστικά, Πλεονεκτήματα
  - Συνοχή (Cohesion) και Διασύνδεση (Coupling)
  - Παραδείγματα Μονάδων : glib-c, gzip, coreutils
  - Τύποι Μονάδων, Μονάδες και Βιβλιοθήκες
- **Έλεγχοι και Ανάλυση Κώδικα**
  - Επιβεβαίωση Ισχυρισμών `assert()`, Cunit/Junit
  - Έννοιες: Στατική & Δυναμική Ανάλυση Κώδικα
  - Εργαλεία: Valgrind, gprof, JProfiler (JAVA) και διασύνδεση με Eclipse



# Έλεγχοι Μονάδας ΧΩΡΙΣ Εργαλεία (Assertions)

- Μια εναλλακτική (ή συμπληρωματική) πρακτική για τον έλεγχο μονάδων είναι αυτή της **Επιβεβαίωσης Ισχυρισμών (Assertions)**
  - Υποστηρίζεται από όλες τις δημοφιλείς γλώσσες!
  - Επιτρέπει στο πρόγραμμα να ελέγξει τη συμπεριφορά του και να εντοπίσει τυχόν λάθη

- **Παράδειγμα**

```
#include <assert.h> // assert()
#include <stdlib.h> // EXIT_SUCCESS
int main() {
    int A[10], i = 15;
    assert(0<=i && i<10);
    A[i] = 0; return EXIT_SUCCESS;
}
```

*Το assert λοιπόν μπορεί να το φανταστεί κανείς ως μια **συνθήκη ελέγχου if**, η οποία **τερματίζει τη ροή εκτέλεσης** ενός προγράμματος εάν δεν ικανοποιείται η συνθήκη.*

- **Εκτέλεση** **Assertion failed: (0<=i && i<10), function main, file new.c, line 6.  
Abort trap**



# Έλεγχοι Μονάδας ΧΩΡΙΣ Εργαλεία (Assertions)

- **Μειονέκτημα assert:** Ο Χρόνος εκτέλεσης του προγράμματος αυξάνεται από τον επιπλέον έλεγχο, `assert (condition)`.
- Συνεπώς, πολλοί χρησιμοποιούν το `assert` μόνο κατά την φάση της δοκιμής / αποσφαλμάτωσης με την ακόλουθη λογική:
  - Εισαγωγή της `assert.h` αφού πρώτα γίνει `def` το `NDEBUG`.

```
#define NDEBUG
#include <assert.h>
```
  - Στην περίπτωση μας θα μπορούσαμε να συνδυάσουμε με τη `DEBUG` μεταβλητή του `makefile` μας.
  - ```
#ifndef DEBUG
    #define NDEBUG
    #include <assert.h>
#endif
```







# Το Εργαλείο CUnit

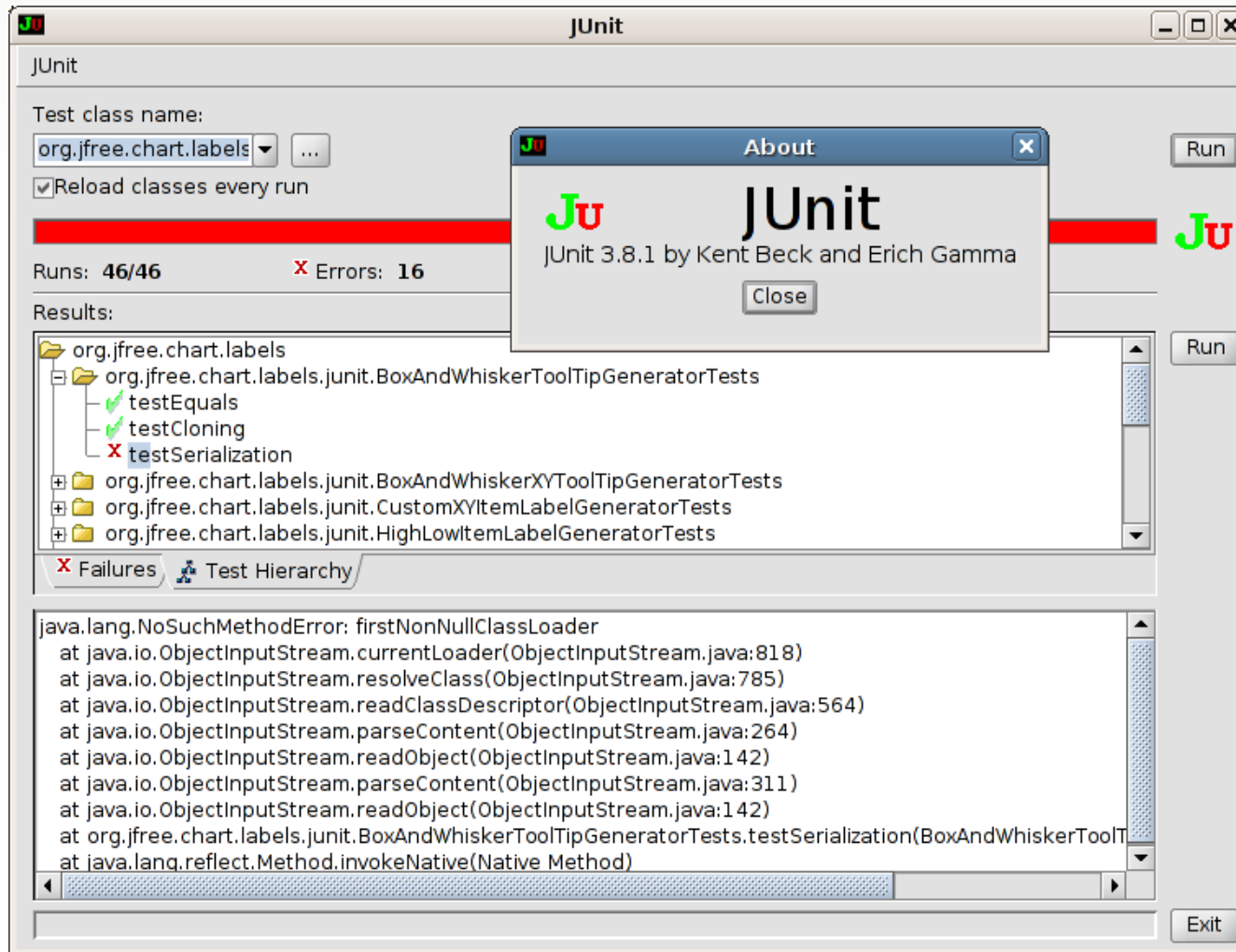
## Έλεγχοι Μονάδας με Εργαλεία

- CUnit είναι ένα ελαφρύ σύστημα για τη συγγραφή, τη διαχείριση και την εκτέλεση δοκιμών μονάδας στη C. Παρέχει στους προγραμματιστές C μια βασική λειτουργικότητα δοκιμών με μια ευέλικτη ποικιλία διασυνδέσεων χρήστη.
- CUnit είναι ενσωματωμένη ως στατική βιβλιοθήκη, η οποία συνδέεται με τον κώδικα δοκιμής του χρήστη. Χρησιμοποιεί ένα απλό πλαίσιο για τις δομές δοκιμών δόμησης και παρέχει ένα πλούσιο σύνολο ισχυρισμών για τον έλεγχο κοινών τύπων δεδομένων.
- Παραδείγματα: <http://cunit.sourceforge.net/example.html>



# Το Εργαλείο CUnit

## Έλεγχοι Μονάδας με Εργαλεία



Χρήση JUnit για τον  
έλεγχο μονάδας  
κλάσεων JAVA



# Στατική Ανάλυση Κώδικα (Static Code Analysis)

- **Στατική Ανάλυση Κώδικα (Static Code Analysis):** Η ανάλυση ενός κώδικα **ΧΩΡΙΣ** να εκτελεστεί (δηλ., σε επίπεδο **πηγαίου κώδικα** ή **αντικειμενικού κώδικα**)
  - **Μέρος** τέτοιας **ανάλυσης** γίνεται σήμερα από τον **μεταγλωττιστή**, π.χ., unreachable code, variable-not-used, κτλ. (παλαιότερο από άλλα εργαλεία, π.χ, lint)
  - Σε αρκετά πεδία απαιτούνται σήμερα **τυπικές μέθοδοι (formal methods)** οι οποίες αποδεικνύουν με **μαθηματικό τρόπο** ότι ένα λογισμικό ανταποκρίνεται στις απαιτήσεις. (π.χ., έλεγχος μοντέλων, λογισμικό για ιατρικούς, πυρηνικούς σκοπούς,)



# Δυναμική Ανάλυση Κώδικα (Dynamic Code Analysis)

- **Δυναμική Ανάλυση Κώδικα (Dynamic Code Analysis):** Η ανάλυση ενός κώδικα **ΜΕΣΩ** εκτέλεσης του προγράμματος σε ένα **πραγματικό ή νοητό** επεξεργαστή
  - Κατά την εκτέλεση καταγράφονται στοιχεία για το πρόγραμμα (π.χ., χρήση μνήμης, επεξεργαστή, κτλ)
  - Το αποτέλεσμα αυτής της ανάλυσης επιτρέπει στον χρήστη να εντοπίσει:
    - Διαρροή Μνήμης (Memory Leaks)
    - Χρόνος Συναρτήσεων, Απαίτηση Πόρων, κτλ.



# Δυναμική Ανάλυση Κώδικα (Dynamic Code Analysis)

Χρόνος Συναρτήσεων με `gprof` (βλέπε Εργαστήριο)

```
gcc prog.c -pg -o prog;  
gprof -bc prog
```

80% του χρόνου!  
11.6 seconds\*

```
Each sample counts as 0.01 seconds.
```

| %<br>time | cumulative<br>seconds | self<br>seconds | calls     | self<br>s/call | total<br>s/call | name                 |
|-----------|-----------------------|-----------------|-----------|----------------|-----------------|----------------------|
| 80.87     | 11.61                 | 11.61           | 161798155 | 0.00           | 0.00            | isValidMove          |
| 15.14     | 13.79                 | 2.17            | 20224717  | 0.00           | 0.00            | findNextEntry        |
| 2.13      | 14.09                 | 0.31            | 1         | 0.31           | 14.21           | solvePuzzle          |
| 1.05      | 14.24                 | 0.15            | 1         | 0.15           | 0.15            | createFilename       |
| 0.35      | 14.29                 | 0.05            | 10112282  | 0.00           | 0.00            | pop                  |
| 0.21      | 14.32                 | 0.03            | 20224564  | 0.00           | 0.00            | IsEmptyStack         |
| 0.21      | 14.35                 | 0.03            | 10112435  | 0.00           | 0.00            | push                 |
| 0.14      | 14.37                 | 0.02            | 1         | 0.02           | 0.02            | allocateMemory       |
| 0.07      | 14.38                 | 0.01            | 1         | 0.01           | 0.01            | MakeEmptyStack       |
| 0.00      | 14.38                 | 0.00            | 206       | 0.00           | 0.00            | printNeatLockedValue |
| 0.00      | 14.38                 | 0.00            | 153       | 0.00           | 0.00            | printNeatDot         |
| 0.00      | 14.38                 | 0.00            | 153       | 0.00           | 0.00            | printNeatValue       |
| 0.00      | 14.38                 | 0.00            | 10        | 0.00           | 0.00            | printLine            |
| 0.00      | 14.38                 | 0.00            | 2         | 0.00           | 0.00            | displayPuzzle        |
| 0.00      | 14.38                 | 0.00            | 2         | 0.00           | 0.00            | isValidPuzzle        |
| 0.00      | 14.38                 | 0.00            | 1         | 0.00           | 0.00            | readPuzzle           |
| 0.00      | 14.38                 | 0.00            | 1         | 0.00           | 0.00            | writePuzzle          |

\* Οι χρόνοι καταγράφονται κάνοντας δειγματοληψία το πρόγραμμα σε διαφορετικά χρονικά σημεία.



# Δυναμική Ανάλυση Κώδικα (Dynamic Code Analysis)

## Εντοπισμός Διαρροών Μνήμης (Memory Leaks) με `valgrind` (βλέπε Εργαστήριο)

```
valgrind --tool=memcheck --leak-check=full --show-reachable=yes --num-callers=20 --track-fds=yes ./test
```

```
gcc -o test -g test.c
```

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

```
int main()
{
```

```
    char *p;
```

```
    // Allocation #1 of 19 bytes
    p = (char *) malloc(19);
```

```
    // Allocation #2 of 12 bytes
    p = (char *) malloc(12);
    free(p);
```

```
    // Allocation #3 of 16 bytes
    p = (char *) malloc(16);
```

```
    return 0;
```

```
}
```

```
==14341== HEAP SUMMARY:
```

```
==14341==    in use at exit: 35 bytes in 2 blocks
```

```
==14341== total heap usage: 3 allocs, 1 frees, 47 bytes allocated
```

```
==14341==
```

```
==14341== 16 bytes in 1 blocks are definitely lost in loss record 1 of 2
```

```
==14341==    at 0x4A0610C: malloc (vg_replace_malloc.c:195)
```

```
==14341==    by 0x40050E: main (test.c:16)
```

```
==14341==
```

```
==14341== 19 bytes in 1 blocks are definitely lost in loss record 2 of 2
```

```
==14341==    at 0x4A0610C: malloc (vg_replace_malloc.c:195)
```

```
==14341==    by 0x4004E9: main (test.c:9)
```

```
==14341==
```

```
==14341== LEAK SUMMARY:
```

```
==14341==    definitely lost: 35 bytes in 2 blocks
```

\* Εκτέλεση σε νοητό επεξεργαστή



# Ανάλυση Κώδικα στο Eclipse (To linuxtools plugin)

Κάνοντας install τα "**linuxtools**" στο Eclipse CDT μας δίνει την ευκαιρία χρήσης του **GProf**, **Valgrind** και άλλων εργαλείων απευθείας από το **eclipse CDT**.

