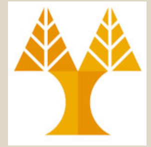


ΕΠΛ232 – Προγραμματιστικές Τεχνικές και Εργαλεία

Διάλεξη 12: Οργάνωση Προγραμμάτων σε Πολλαπλά Αρχεία II

(Κεφάλαια 15.2-15.4 &14.1,14.2,14.4, KNK2ED)

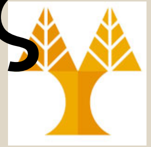
Δημήτρης Ζεϊναλιπούρ

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

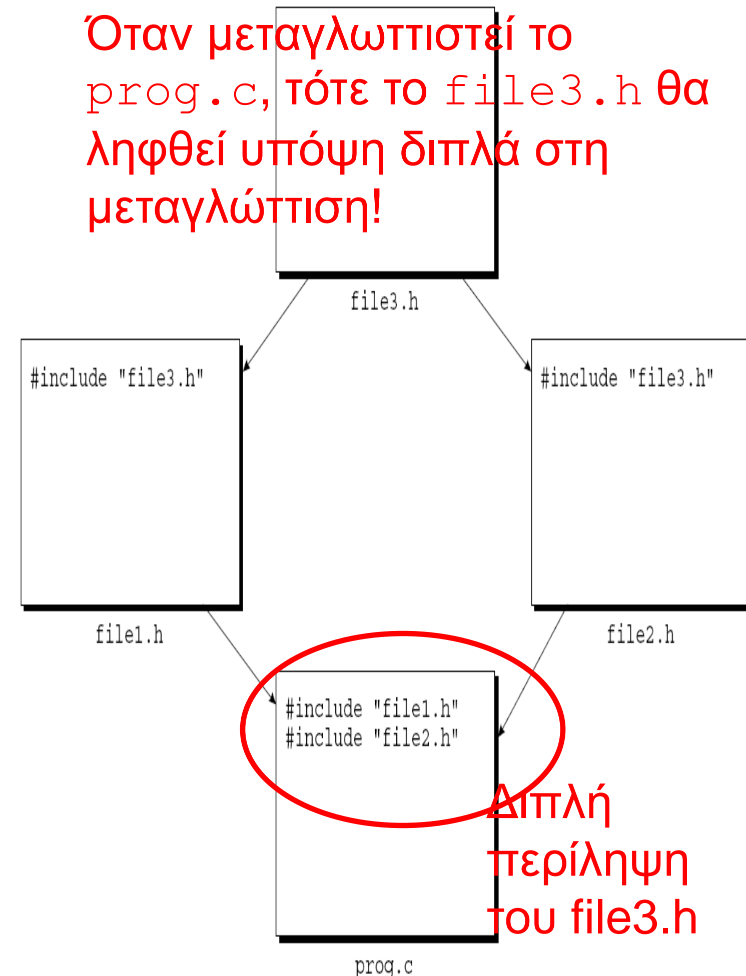
Περιεχόμενο Διάλεξης 12

- **Οργάνωση σε Πολλαπλά Αρχεία (15.2-15.3)**
  - Εμφωλευμένα Include
  - Προστασία Αρχείων Κεφαλίδας με `#ifndef … #endif`
  - Λογική Διάσπασης και Παράδειγμα: justify
- **Μεταγλώττιση με Makefiles (15.4)**
  - Makefiles και Αυτοματοποίηση Μεταγλώττισης
  - Γενικό (Generic) Makefile
  - Λογική Επανα-Μεταγλώττισης
- **Οδηγίες Προεπεξεργαστή (14.1,14.2,14.4)**
  - Δηλώσεις `#include` και `#define`, Παραδείγματα Χρήσης
  - Παραμετροποιημένες Μακροεντολές (Macros)
  - Δήλωση Μακροεντολών εκτός Προγράμματος

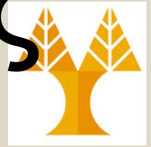
Προστασία Αρχείων Κεφαλίδες (Protecting Header Files)



- Εάν ένα αρχείο κώδικα (.c) περιέχει το **ΙΔΙΟ αρχείο κεφαλίδας 2 φορές**, τότε ενδέχεται να προκύψουν **προβλήματα** μεταγλώττισης.
  - Εάν το `file3.h` περιέχει μόνο define macros, function prototypes, και δηλώσεις μεταβλητών δεν δημιουργείται πρόβλημα
  - Εάν το `file3.h` περιέχει `typedef` θα πάρετε σφάλμα μεταγλώττισης



Προστασία Αρχείων Κεφαλίδες (Protecting Header Files)

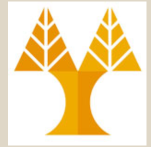


- Για να προστατέψουμε ένα αρχείο κεφαλίδας, μπορούμε να περιλάβουμε το περιεχόμενο του σε μια οδηγία Προεπεξεργαστή **#ifndef-#endif**.
- Παράδειγμα προστασίας `boolean.h`:

```
#ifndef BOOLEAN_H
#define BOOLEAN_H

#define TRUE 1
#define FALSE 0
typedef int Bool;

#endif
```

Ifndef: If not defined
Εάν και δεν είναι απαραίτητο, να δίνεται το όνομα ΠΑΝΤΑ βάσει του ονόματος του αρχείου για αποφυγή λαθών

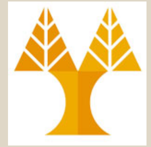
Διάσπαση Προγράμματος σε Πολλαπλά Αρχεία



Η Σχεδίαση ενός προγράμματος προϋποθέτει (8 βήματα):

1. **Find Functions:** εύρεση των συναρτήσεων που αποτελούν το πρόγραμμα.
2. **Group Functions:** Οι συναρτήσεις ομαδοποιούνται σε λογικά-σχετιζόμενες ομάδες (logically related groups).
3. **Design Header:** Κάθε **σύνολο συναρτήσεων (αντικείμενο)** πρέπει να έχει ένα αρχείο κεφαλίδας (`foo.h`):
   - **"Διαφήμιση" Εξωτερικών Προτύπων/Μεταβλητών/κτλ:** `foo.h` περιέχει τα πρότυπα των συναρτήσεων που ορίζονται στο `foo.c` αλλά και οτιδήποτε θέλουμε να έχει εμβέλεια **εκτός** του `foo.c`.
   - **"Απόκρυψη" Εσωτερικών Προτύπων/Μεταβλητών/κτλ:** Στοιχεία που θα χρησιμοποιηθούν μόνο στο `foo.c` ΔΕΝ πρέπει να δηλωθούν στο `foo.h` αλλά μόνο στο `foo.c` (με **`static`**.)
   - **Βιβλιοθήκες:** Συμπεριλάβε εδώ όσες βιβλιοθήκες χρειάζεστε για την αυτόνομη μεταγλώττιση / δοκιμή του αντικειμένου `foo.c`

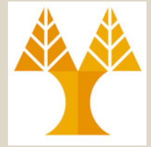
Διάσπαση Προγράμματος σε Πολλαπλά Αρχεία



Η Σχεδίαση ενός προγράμματος προϋποθέτει (συνέχεια):

4. **Code Source:** Υλοποιούμε το αρχείο `foo.c` βάσει των προτύπων που δόθηκαν στο αρχείο κεφαλίδας (`foo.h`).
   - `#include "foo.h"` στο `foo.c`
   - έτσι ώστε ο μεταγλωττιστής να μπορεί να ελέγξει ότι τα πρότυπα του αρχείου `foo.h` ταιριάζουν με αυτά του `foo.c`.
   - Το `.c` περιέχει και όλες τις ***"Εσωτερικές Συναρτήσεις"*** (π.χ., auxiliary συναρτήσεις που δεν θέλουμε να "διαφημιστούν" έξω)
5. **Include foo.h Elsewhere:** Το `foo.h` θα περιλαμβάνεται σε κάθε αρχείο το οποίο θα χρειαστεί να καλέσει συναρτήσεις που βρίσκονται στο `foo.c`.
6. **Συνάρτηση Main:** Η (μια και μοναδική συνάρτηση) `main` είναι καλό να τοποθετηθεί σε αρχείο που φέρει όνομα όμοιο με αυτό του προγράμματος.

Διάσπαση Προγράμματος σε Πολλαπλά Αρχεία



7. **Driver Αντικείμενου:** Κάθε `.c/.h` σύνολο **ΠΡΕΠΕΙ** να συνοδεύεται από μια συνάρτηση **driver (οδηγό χρήσης)**

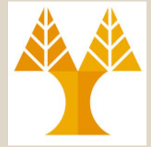
```
#ifdef DEBUG
/* Declaring Object Unit Tests */
static void tester1() { ... }
static void tester2() { ... }
int main(void) {
   tester1(); tester2(); //...
   return 0;
}
#endif
```

Χωρίς αυτό, συμπερίληψη του foo.h στο main.c θα έδειχνε 2 main() συναρτήσεις στον μεταγλωττιστή => ERROR

Εμβέλεια συνάρτησης στα πλαίσια του αρχείου μόνο (δηλ., PRIVATE)

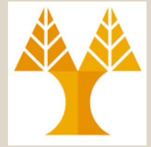
- Μια τέτοια συνάρτηση επιτρέπει να **αποσφαλματώσουμε** το αντικείμενο .c/.h **ξεχωριστά** από τα άλλα αντικείμενα.
  - **Χωρίς Makefile :** `gcc -DDEBUG=1 foo.c`
  - **Με Makefile (σε λίγο) :** `make CFLAGS=-DDEBUG=1 foo`
- Αργότερα θα μάθουμε πώς να γράφουμε drivers για κάλυψη **οριακών περιπτώσεων (unit tests)**

Παράδειγμα Οργάνωσης Προγράμματος



- Υποθέστε ένα πρόγραμμα το οποίο χωρίζεται στα ακόλουθα αρχεία:
  - `stack.c`: συναρτήσεις που σχετίζονται με τη στοίβα
    - Περιλαμβάνει main() μέσα σε **`#ifdef DEBUG`** … **`#endif`**
  - `queue.c`: συναρτήσεις που σχετίζονται με τη ουρά
    - Περιλαμβάνει main() μέσα σε **`#ifdef DEBUG`** … **`#endif`**
  - **`game.c`**: περιέχει τη συνάρτηση **`main`**
    - … το main() ΔΕΝ περιλαμβάνεται σε `ifdef`.
- Θα χρειαστούμε τουλάχιστον δυο αρχεία κεφαλίδας:
  - `stack.h`: πρότυπα συναρτήσεων του `stack.c`
  - `queue.h`: πρότυπα συναρτήσεων του `queue.c`
- **Βασικό Παράδειγμα για επίλυση AS3!**

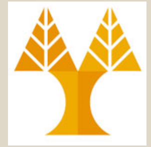
Παράδειγμα Οργάνωσης Προγράμματος



Διάγραμμα Συμπερίληψη Αρχείων Κεφαλίδας Προβλήματος

def.h → stack.h, queue.h
stack.h → game.c, stack.c
queue.h → game.c, queue.c

Παράδειγμα Οργάνωσης Προγράμματος



[def.h]

```
#ifndef DEF_H
#define DEF_H

// A) Libraries
// nothing

// B) Declarations
 typedef struct node {
        int            data;
        struct node *next;
} NODE;

// C) Function Prototypes
// nothing
#endif
```

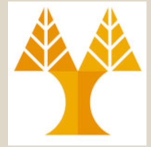
Κοινές Δηλώσεις για Stack.h και Queue.h

Μια σημείωση για το static

typedef static struct node { } NODE;
// ΛΑΘΟΣ – typedef δηλώνει τύπο.

static NODE node;
// OK, εφόσον ορίζουμε ότι το node θα έχει στατική αποθηκευτική διάρκεια (δηλ., "ορατότητα" στην εμβέλεια που ορίζεται, π.χ., αρχείου ή συνάρτησης)

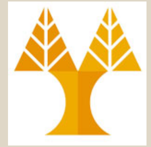
Παράδειγμα Οργάνωσης Προγράμματος



[stack.h]

```
#ifndef STACK_H
#define STACK_H

// A) Libraries
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include "def.h"

// B) Declarations
typedef struct {
    NODE  *top;
    int   size;
} STACK;

// C) Function Prototypes
/**
 * Doxygen Comment
 */
STACK *initStack(STACK *stack);
/* Comments*/
int initStack2(STACK **stack);
/* Comments */
bool IsEmptyStack(STACK *stack);
/* Comments */
void top(STACK *stack);
/* Comments */
int push(int value, STACK *stack);
/* Comments */
int pop(STACK *stack, int *retval);
#endif
```

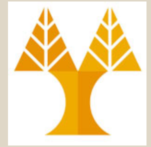
Παράδειγμα Οργάνωσης Προγράμματος



[queue.h]

```c
#ifndef QUEUE_H
#define QUEUE_H

// A) Libraries
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include "def.h"

// B) Declarations
typedef struct {
   NODE *head;
   NODE *tail;
   int length;
} QUEUE;

// C) Function Prototypes
/**
 * Doxygen Comment
 */
QUEUE *initQueue(QUEUE *queue);
int initQueue2(QUEUE **queue);
/* Comments */
bool isEmptyQueue(QUEUE *queue);
/* Comments */
void printHead(QUEUE *queue);
/* Comments */
void printTail(QUEUE *queue);
/* Comments */
int enQueue(int value, QUEUE *q);
/* Comments */
int deQueue(QUEUE *q, int *retval);
#endif
```

Παράδειγμα Οργάνωσης Προγράμματος



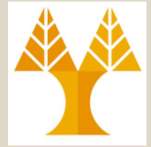
[game.c]

```
/* Formats a file of text */

#include <string.h>
#include "stack.h"
#include "queue.h"

#define MAX_WORD_LEN 20

int main(void)
{
    ...
}
```

Άσκηση για το Σπίτι: Οργανώστε την τελική έκδοση της RPN αριθμομηχανή

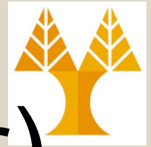
Μεταγλώττιση; Πολλά Αρχεία ☹ Δύσκολο να το χειριστούμε ...

Makefiles (Αυτοματοποίηση Μεταγλώττισης)



- Για την ευκολότερη μεταγλώττιση μεγάλων προγραμμάτων, το UNIX παρέχει την έννοια των ***makefile.***
  - Την οποία θα χρησιμοποιήσουμε για τις εργασίες 3-5
- Το **makefile** είναι ένα αρχείο κειμένου με ειδική σύνταξη η οποία επιτρέπει:
  - Τη **δήλωση των αρχείων** που αποτελούν ένα πρόγραμμα.
  - Την περιγραφή των **συσχετίσεων (*dependencies*)** ανάμεσα σε πηγαία αρχεία και αρχεία κεφαλίδας.
    - **Συσχετίσεις Μεταγλώττισης**
      - Υποθέστε ότι το `game.c` περιλαμβάνει το αρχείο `stack.h`.
      - Τυχών αλλαγή του `stack.h` προϋποθέτει την επανα-μεταγλώττιση του `game.c`
  - Τη δήλωση **ειδικών ορισμάτων μεταγλώττισης**.

Makefiles (Αυτοματοποίηση Μεταγλώττισης)



Target (Στόχος): Εξαρτήσεις:

- Makefile για το πρόγραμμα `game` :

```
game: game.o stack.o queue.o
          gcc -o game stack.o queue.o

game.o: game.c stack.h queue.h
          gcc -c game.c

stack.o: stack.c stack.h
          gcc -c stack.c

queue.o: queue.c queue.h
          gcc -c queue.c
```

link

Εντολές: *Compile but don't link*

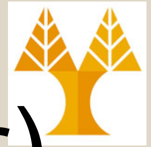
Κανόνες (Rules):
4 Ομάδες Εντολών

Προτιμήστε το γενικό makefile που ακολουθεί σε λίγες διαφάνειες!

Makefiles (Αυτοματοποίηση Μεταγλώττισης)

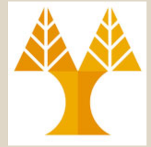
- Το **makefile** για ένα πρόγραμμα φτιάχνεται με ένα **κειμενογράφο** και φυλάσσεται σε αρχείο με όνομα **`Makefile`** ή **`makefile`**.
- Στη συνέχεια χρησιμοποιούμε την εντολή **make** για να μεταγλωττίσουμε (ή να επανα-μεταγλωττίσουμε) το πρόγραμμα
  - Εάν δεν υπάρχει το make εγκατεστημένο στο σύστημα σας θα πρέπει να το εγκαταστήσετε.
- Ελέγχοντας την **ώρα / ημερομηνία** μεταβολής του κάθε αρχείου **πηγαίου κώδικα** σας, το **make** είναι σε θέση να αναγνωρίζει ποιά αρχεία **πρέπει** να **επανα-μεταγλωττιστούν.**

Makefiles (Αυτοματοποίηση Μεταγλώττισης)



- Κάθε εντολή στο αρχείο **makefile** πρέπει να φέρει στην αρχή ένα tab (όχι μια ακολουθία από spaces), εναλλακτικά θα πάρετε κάποια λάθη.
  - $ make
  - Makefile:26: *** missing separator. Stop.
- Τα **makefile** μπορεί να είναι **πάρα πολύ περίπλοκα** (για μεταγλώττιση μεγάλων προγραμμάτων – ακόμη και ολόκληρων λειτουργικών συστημάτων).
  - Δείτε http://en.wikipedia.org/wiki/Make_(software)
- Στα πλαίσια του **μαθήματος** μπορούμε να **χρησιμοποιήσουμε** το γενικό **Makefile** της επόμενης διαφάνειας (να συνοδεύει όλες τις επόμενες εργασίες)
  - Το eClipse IDE παρέχει τη δυνατότητα χρήσης του Makefile που θα χρησιμοποιούμε στη γραμμή εντολών (δείτε εργαστήριο)

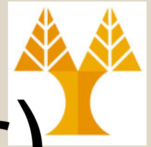
Ένα Γενικό Makefile για Όλα τα Προγράμματα



```
##############################################
# Makefile for compiling the program skeleton
# 'make'         build executable file 'PROJ'
# 'make doxy'    build project manual in doxygen
# 'make all'     build project + manual
# 'make clean'   removes all .o, executable and doxy log
##############################################
PROJ = as3     # the name of the project
CC   = gcc                # name of compiler
DOXYGEN = doxygen       # name of doxygen binary

# define any compile-time flags
CFLAGS = -std=c99 -Wall -O -Wuninitialized -Wunreachable-
    code –pedantic # there is a space at the end of this
LFLAGS = -lm

##############################################
# You don't need to edit anything below this line
##############################################

# list of object files
# The following includes all of them!
C_FILES := $(wildcard *.c)
OBJS := $(patsubst %.c, %.o, $(C_FILES))
```

```
# To create the executable file  we need the individual
# object files
$(PROJ): $(OBJS)
	$(CC) $(LFLAGS) -o $(PROJ) $(OBJS)

# To create each individual object file we need to
# compile these files using the following general
# purpose macro
.c.o:
	$(CC) $(CFLAGS)  -c $<
# there is a TAB for each identation.
# To make all (program + manual) "make all"
all :
	make
	make doxy

# To make all (program + manual) "make doxy"
doxy:
	$(DOXYGEN) doxygen.conf &> doxygen.log

# To clean .o files: "make clean"
clean:
	rm -rf *.o doxygen.log html
```

Makefiles στο eClipse (Αυτοματοποίηση Μεταγλώττισης)



New Project

Select a wizard

Creates a new Makefile project in a directory containing existing c

Wizards:

type filter text

- Java Project
- Java Project from Existing Ant Buildfile
- Plug-in Project
- General
  - Faceted Project
  - Project
- Android
  - Android Project
  - Android Sample Project
  - Android Test Project
- C/C++
  - C Project
  - C++ Project
  - Makefile Project with Existing Code

< Back | Next > | Cancel | Finish

C Project

C Project

Project name must be specified

Project name:

Use default location

Location: /Users/dzeina/CODE Browse...

Choose file system: default

Project type:

- Executable
- Executable (XL UPC)
- Shared Library
- Shared Library (XL UPC)
- Static Library
- Static Library (XL UPC)
- Executable (XL C/C++)
- Static Library(XL C/C++)
- Shared Library (XL C/C++)
- Executable (XL UPC)
- Static Library(XL UPC)
- Shared Library (XL UPC)
- Makefile project
  - Empty Project

Toolchains:

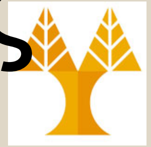
Project category is selected. Expand the category and select a concrete project type.

Show project types and toolchains only if they are supported on the platform

< Back | Next > | Cancel | Finish

To eClipse IDE παρέχει τη δυνατότητα χρήσης δικών μας (Standard Make) ή δημιουργίας (Managed Make) makefiles. Περισσότερα στο Εργαστήριο

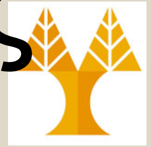
Δήλωση Μακροεντολών Εκτός Προγράμματος



- Οι πλείστοι μεταγλωττιστές (και ο GCC) υποστηρίζουν την επιλογή **–D**, η όποια επιτρέπει τον ορισμό της τιμής ενός macro στην γραμμή εντολών (χρήσιμο σε makefiles)
  - π.χ., ορισμός macro `DEBUG=1` στο `foo.c`

    `gcc -DDEBUG=1 foo.c` ή `gcc -DDEBUG foo.c`
- Σεβασμός Ορισμάτων CFLAGS του Makefile (με χρήση shell script command substitution):
  - **`CFLAGS="`cat Makefile | grep "CFLAGS =" | awk -F" = " '{print $2}'`"`**
    - Να δίνεται κάθε φορά που ανοίγεται το shell.
  - **`make CFLAGS="$CFLAGS -DDEBUG=1 -DTRACE=1 " stack`**
    - προσθέτει στα υφιστάμενα flags το -DDEBUG=1 και -DTRACE=1

Δήλωση Μακροεντολών Εκτός Προγράμματος



Παραδείγματα Χρήσης

```
make            # Σχόλιο: απλή μεταγλώττιση
make clean      # διαγραφή ενδιάμεσων αρχείων .o κτλ
make doxy       # δημιουργία doxygen
make all        # μεταγλώττιση όλων των αρχείων και doxygen
```

```
# Κάθε φορά που ανοίγετε το τερματικό
CFLAGS="`cat Makefile | grep "CFLAGS =" | awk -F" = " '{print $2}'`"
# Compile + TRACE
make CFLAGS="$CFLAGS -DTRACE=1"
```

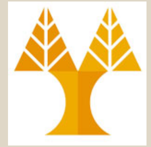
```
#ifdef TRACE
    printf(" > Pushing to stack");
#endif
```

```
# Compile + Doxygen + TRACE
make CFLAGS="$CFLAGS -DTRACE=1" all
```

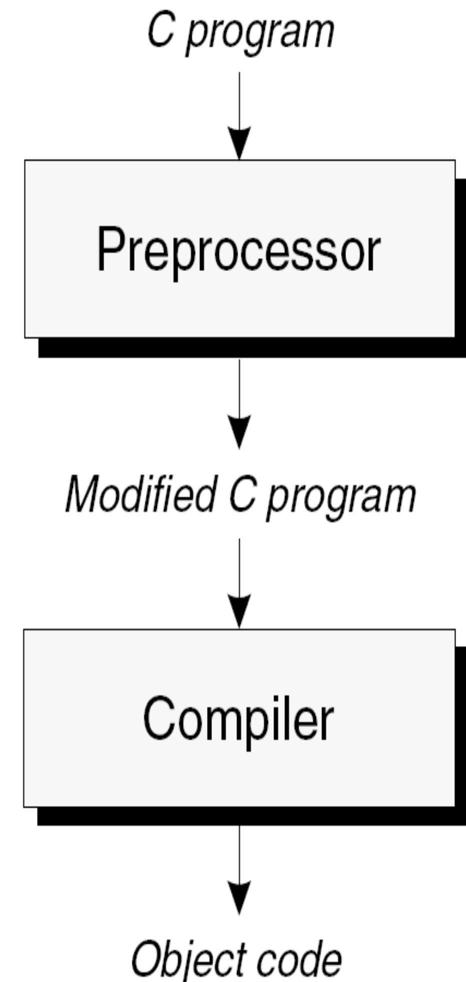
```
all :
        make
        make doxy
```

```
# Compile + DEBUG (stack) + TRACE
make CFLAGS="$CFLAGS -DTRACE=1 -DDEBUG=1" stack
```

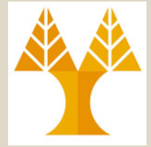
Οδηγίες Προεπεξεργαστή (Preprocessor Directives)



- Θυμίζουμε ότι ο **Προεπεξεργαστής** σε ένα πρόγραμμα εκτελείται ΠΡΙΝ την μετατροπή των **αντικειμενικών αρχείων**
  - Συμπερίληψη **δηλώσεων** από αρχεία **κεφαλίδας βιβλιοθηκών**
- Στη C είναι δυνατό να κωδικοποιούνται σχεδόν **ολόκληρα προγράμματα** με τον Προεπεξεργαστή!
  - Ωστόσο **ΠΡΕΠΕΙ** να αποφεύγεται η χρήση του στον **μεγαλύτερο δυνατό βαθμό**.
  - Αυτό εφόσον δεν έχει σχεδιαστεί για ανάπτυξη του ίδιου του κώδικα αλλά απλά για διαδικασίες προ-επεξεργασίας όπως δείχνουν τα ακόλουθα παραδείγματα



Οδηγίες Προεπεξεργαστή (Preprocessor Directives)



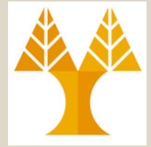
- Παράδειγμα Προεπεξεργασίας `#include <stdio.h>`:

Blank line
Blank line
Lines brought in from stdio.h
Blank line
Blank line
Blank line
Blank line

```
int main(void)
{
  float fahrenheit, celsius;

  printf("Enter Fahrenheit temperature: ");
  scanf("%f", &fahrenheit);

  celsius = (fahrenheit - 32.0f) * (5.0f / 9.0f);

  printf("Celsius equivalent is: %.1f\n", celsius);

  return 0;
}
```

Δείτε το μόνοι σας!
gcc –E example.c

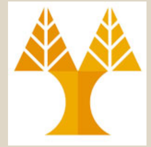
Οδηγίες Προεπεξεργαστή #include και #define



- **Χρήση 1:** Παράδειγμα Συμπερίληψης Διαφορετικού Αρχείου Κεφαλίδας για διαφορετικούς επεξεργαστές (ή διακοπή μεταγλώττισης – σε αρκετούς μεταγλωττιστές)

```
#if defined(IA32)
  #define CPU_FILE "ia32.h"
#elif defined(IA64)
  #define CPU_FILE "ia64.h"
#elif defined(AMD64)
  #define CPU_FILE "amd64.h"
#else
  #error No CPU_FILE found specified
#endif

#include CPU_FILE
```

Οδηγίες Προεπεξεργαστή `#include` και `#define`



- **Χρήση 2: Τοποθέτηση Σχόλιου γύρω από σχόλια:**
  - **Λανθασμένη Έκδοση**

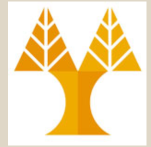
```
/* /* Comment */ */
```

```
error: expected identifier or '(' before '/' token
```

- **Διορθωμένη Έκδοση με `#IF`**

```
#if 0
/* Comment */
#endif
```

Οδηγίες Προεπεξεργαστή `#include` και `#define`



- **Χρήση 3:** Πρόγραμμα το οποίο είναι συμβατό με διαφορετικά λειτουργικά συστήματα:

```
#if defined(WIN32)
…
#elif defined(MAC_OS)
…
#elif defined(LINUX)
…
#endif
```

Χρήση 5: Απλές Συναρτήσεις

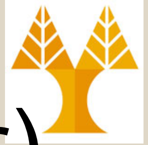
```
#define ADD(x,y)  (x+y)
```

- **Χρήση 4:** Ενότητες που θα εμφανίζονται μόνο με το τον ορισμό ειδικών σταθερών (DEBUG_CRITICAL, DEBUG_LOG, …)

```
#define DEBUG_LOG 1
#if defined(DEBUG_LOG)
printf("Value of i: %d\n", i);
#endif
```

Parenthesis Not necessary

Οδηγίες Προεπεξεργαστή (Παραμετροποιημένες Μακροεντολές)



- Παραδείγματα **Παραμετροποιημένων μακροεντολών (parameterized macros):**

```
#define MAX(x,y)     ((x)>(y)?(x):(y))
#define IS_EVEN(n)   ((n)%2==0)
```

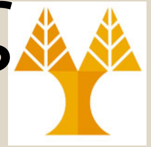
- **Κλήση των Μακροεντολών:**

```
i = MAX(j+k, m-n);
if (IS_EVEN(i)) i++;
```

- **Οι ίδιες γραμμές μετά από την αντικατάσταση της μακροεντολής:**

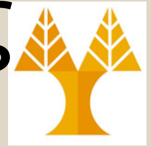
```
i = ((j+k)>(m-n)?(j+k):(m-n));
if (((i)%2==0)) i++;
```

Παραμετροποιημένες Μακροεντολές (Πλεονεκτήματα & Μειονεκτήματα)



- Η Χρήση **παραμετρο-ποιημένων μακροεντολών** αντί **πραγματικών συναρτήσεων** έχει κάποια **πλεονεκτήματα** και **μειονεκτήματα**:
- **Πλεονεκτήματα:**
  A. Ένα πρόγραμμα με **μακροεντολές** μπορεί να είναι **ελαφρώς γρηγορότερο**.
     - Μια **κλήση συνάρτησης** συνοδεύεται από επιπλέον κόστος δέσμευσης μνήμης στην **στοίβα του προγράμματος**.
     - Μια μακροεντολή από την άλλη **αντικαθιστά την κλήση** την κατά την μεταγλώττιση
  B. Τα Macros είναι "γενικά": Μια μακροεντολή μπορεί να δεχθεί ορίσματα **οποιουδήποτε τύπου**
     - Π.χ., στη συνάρτηση **MAX** μπορούσα να χρησιμοποιήσω int, long int, float, double.
     - Βέβαια η έλλειψη **ελέγχου τύπου (type-check)** μπορεί να θεωρηθεί και ως μειονέκτημα όπως θα δούμε στην επόμενη διαφάνεια.

Παραμετροποιημένες Μακροεντολές (Πλεονεκτήματα & Μειονεκτήματα)



- **Μειονεκτήματα (μερικά):**
  A. ***Το μεταγλωττισμένος κώδικας είναι μεγαλύτερος (program text).***

`n = MAX(i, MAX(j, k));` ΙΣΟΔΥΝΑΜΕΙ (μετά την προεπεξεργασία με)

```
n = ((i)>(((j)>(k)?(j):(k)))?(i):(((j)>(k)?(j):(k))));
```

  B. Δεν ελέγχεται ο **τύπος των ορισμάτων** (***type-check***) *ούτε και γίνονται* ***αυτόματες μετατροπές*** *τύπων.*

  C. Δεν υπάρχουν **δείκτες σε Macros**, ενώ υπάρχουν δείκτες σε συναρτήσεις.

  D. Δείτε το βιβλίο για άλλα μειονεκτήματα

- **Επομένως να τα αποφεύγουμε όσο μπορούμε!**