ΕΠΛ232 – Προγραμματιστικές Τεχνικές και Εργαλεία

Οργάνωση Προγραμμάτων σε Πολλαπλά Αρχεία II
(Κεφάλαιο 15.2-15.4 &14.1,14.2,14.4, KNK-2ED)

Τμήμα Πληροφορικής, Πανεπιστήμιο Κύπρου

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

Dr. Andreas Aristidou a.aristidou@ieee.org

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

Dr. Andreas Aristidou a.aristidou@ieee.org

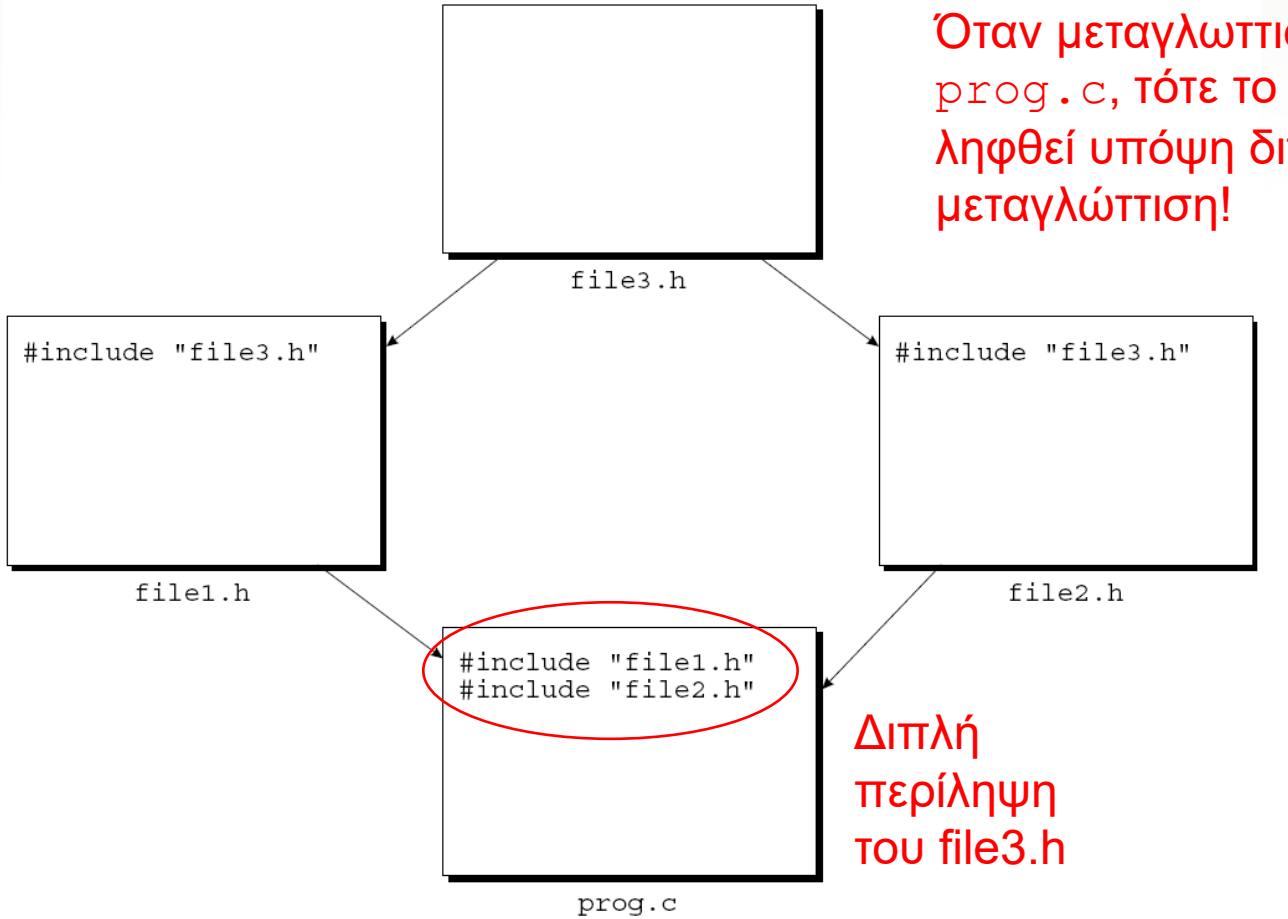
Protecting Header Files

Προστασία Αρχείων Κεφαλίδες

- Εάν ένα αρχείο κώδικα (.c) περιέχει το **ΙΔΙΟ αρχείο κεφαλίδας 2 φορές**, τότε ενδέχεται να προκύψουν **προβλήματα** μεταγλώττισης.
- Αυτό το πρόβλημα είναι σύνηθες όταν τα αρχεία κεφαλίδας περιλαμβάνουν άλλα αρχεία κεφαλίδας.
- Ας υποθέσουμε ότι το `file1.h` περιλαμβάνει τα `file3.h`, το `file2.h` περιλαμβάνει τα `file3.h`, και το `prog.c` περιλαμβάνει τα `file1.h` και `file2.h`.
  - 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

Dr. Andreas Aristidou a.aristidou@ieee.org

Protecting Header Files
Προστασία Αρχείων Κεφαλίδες



Όταν μεταγλωττιστεί το `prog.c`, τότε το `file3.h` θα ληφθεί υπόψη διπλά στη μεταγλώττιση!

Διπλή περίληψη του file3.h

Dr. Andreas Aristidou a.aristidou@ieee.org

Protecting Header Files

Προστασία Αρχείων Κεφαλίδες

- Για να προστατέψουμε ένα αρχείο κεφαλίδας (και να εξοικονόμησετε χρόνο κατά τη διάρκεια της ανάπτυξης του προγράμματος, αποφεύγοντας την περιττή επαναμεταγλώττιση του ίδιου αρχείου 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
Το macro `BOOLEAN.H`, δεν μπορεί να χρησιμοποιηθεί, οπότε η χρήση του ονόματος `BOOLEAN_H` είναι μια καλή εναλλακτική

Dr. Andreas Aristidou a.aristidou@ieee.org

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:** Συμπερίλαβε εδώ όσες βιβλιοθήκες χρειάζεστε για την αυτόνομη μεταγλώττιση / δοκιμή του αντικειμένου `foo.c`

Dr. Andreas Aristidou a.aristidou@ieee.org

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.

Dr. Andreas Aristidou a.aristidou@ieee.org

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)**

Η χρήση της εντολής **#ifdef DEBUG**

Αυτό το πρόγραμμα υπολογίζει το εμβαδό ενός ορθογωνίου και εκτυπώνει "μηνύματα εντοπισμού σφαλμάτων", εάν έχει οριστεί η μακροεντολή DEBUG:

```
void printd(const char *msg) {
#ifdef DEBUG
    printf("%s", msg);
#else
    (void)msg; // ignore msg
#endif
}

int main(void)
{
    printd("main: Starting the program\n");
    printd("main: Calculating the area\n");
    double area = 2.84 * 5.67;
    printd("main: Printing the area\n");
    printf("Area: %0.2f\n", area);
    return EXIT_SUCCESS;
}
```

Οι περισσότεροι μεταγλωττιστές σας επιτρέπουν να μεταγλωττίζετε με ένα συγκεκριμένο σύμβολο που ορίζεται. Έτσι, αν μεταγλωττίσετε με το gcc:

```
gcc -DDEBUG -o printarea printarea.c
```

Το αποτέλεσμα θα περιλαμβάνει τα μηνύματα εντοπισμού σφαλμάτων. Ωστόσο, εάν αποφασίσετε ότι δεν χρειάζεστε τα μηνύματα εντοπισμού σφαλμάτων πλέον, αλλά θέλετε να τα διατηρήσετε στον πηγαίο κώδικα για σκοπούς αναφοράς, τότε απλά μεταγλωττίζετε ξανά το ίδιο αρχείο προέλευσης χωρίς να ορίσετε αυτό το σύμβολο

```
gcc -o printarea printarea.c
```

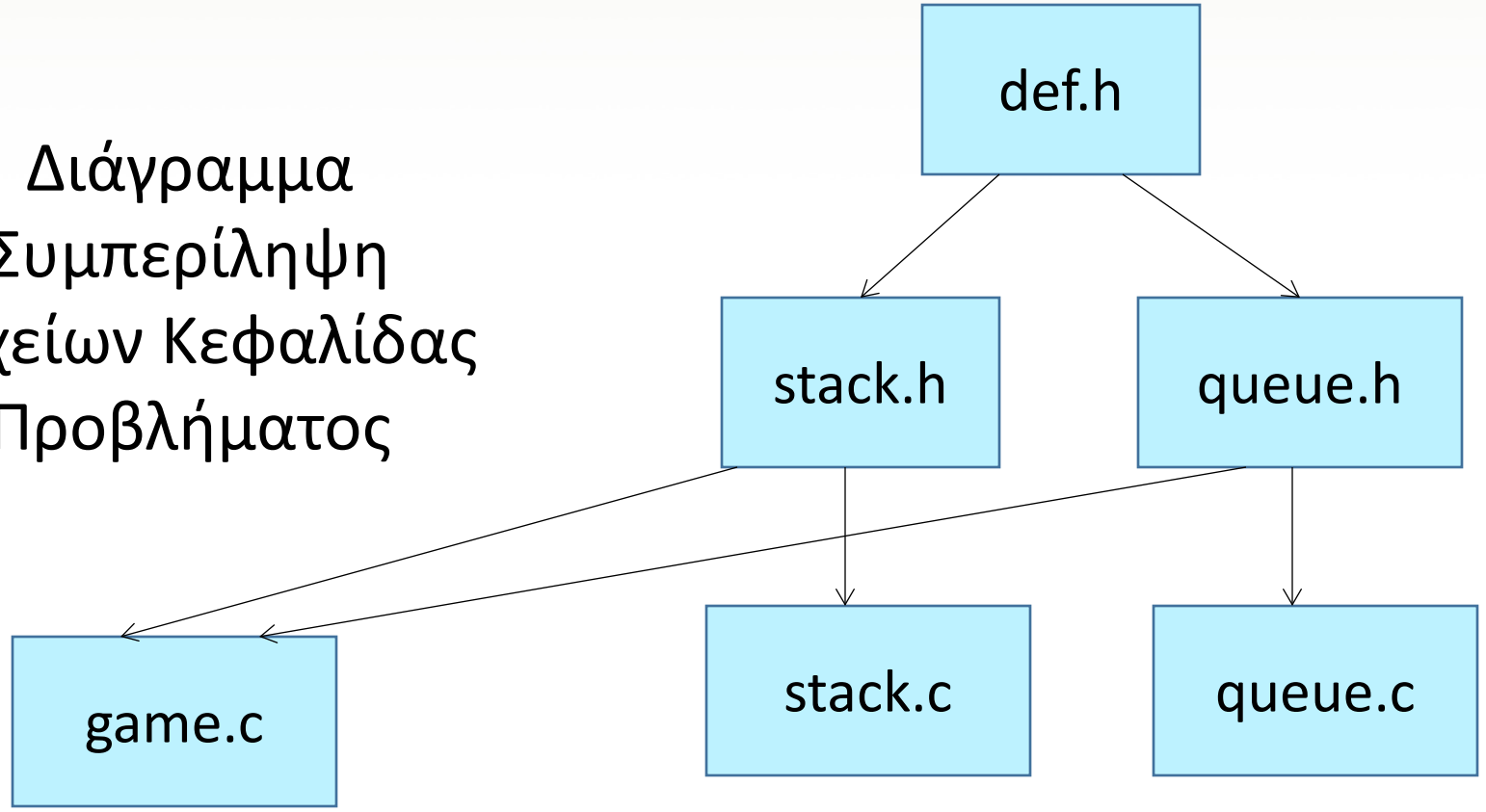
Παράδειγμα Οργάνωσης Προγράμματος

- 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!**

Dr. Andreas Aristidou a.aristidou@ieee.org

Παράδειγμα Οργάνωσης Προγράμματος

Διάγραμμα Συμπερίληψη Αρχείων Κεφαλίδας Προβλήματος



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

Dr. Andreas Aristidou a.aristidou@ieee.org

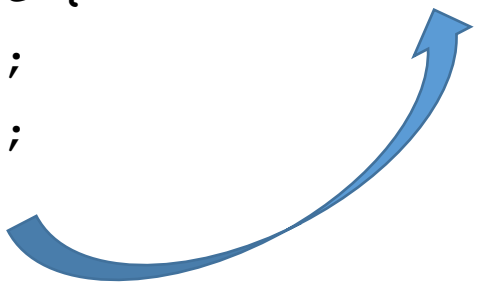
Παράδειγμα Οργάνωσης Προγράμματος

[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
```


Dr. Andreas Aristidou a.aristidou@ieee.org

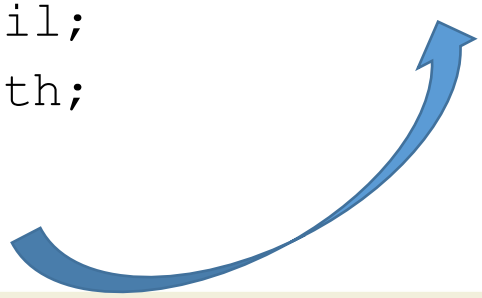
Παράδειγμα Οργάνωσης Προγράμματος

[queue.h]

```
#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
```


Dr. Andreas Aristidou a.aristidou@ieee.org

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 …

Dr. Andreas Aristidou a.aristidou@ieee.org

Δημιουργία προγράμματος πολλαπλών αρχείων

- Η οικοδόμηση ενός μεγάλου προγράμματος απαιτεί τα ίδια βασικά βήματα με τη δημιουργία ενός μικρού:
  - Compiling
  - Linking
- Κάθε πηγαίο αρχείο στο πρόγραμμα πρέπει να μεταγλωττίζεται ξεχωριστά.
- Τα αρχεία κεφαλίδας δεν χρειάζεται να μεταγλωττίζονται.
  - Τα περιεχόμενα ενός αρχείου κεφαλίδας μεταγλωττίζονται αυτόματα κάθε φορά που ένα αρχείο προέλευσης που το περιλαμβάνει μεταγλωττίζεται.
  - Για κάθε πηγαίο αρχείο, το πρόγραμμα μεταγλώττισης δημιουργεί ένα αρχείο που περιέχει το object code.
  - Αυτά τα αρχεία — γνωστά ως ***object files***—έχουν την επέκταση `.o` στα UNIX και `.obj` στα Windows.

Dr. Andreas Aristidou a.aristidou@ieee.org

Δημιουργία προγράμματος πολλαπλών αρχείων

- Το πρόγραμμα σύνδεσης συνδυάζει τα αρχεία αντικειμένων που δημιουργήθηκαν στο προηγούμενο βήμα — μαζί με τον κώδικα για τις συναρτήσεις βιβλιοθήκης — για την παραγωγή ενός εκτελέσιμου αρχείου.
- Μεταξύ άλλων καθηκόντων, το πρόγραμμα σύνδεσης είναι υπεύθυνο για την επίλυση εξωτερικών παραπομπών που έχουν μείνει πίσω από τον μεταγλωττιστή.
- Μια εξωτερική αναφορά παρουσιάζεται όταν μια συνάρτηση σε ένα αρχείο καλεί μια συνάρτηση που ορίζεται σε άλλο αρχείο ή αποκτά πρόσβαση σε μια μεταβλητή που ορίζεται σε άλλο αρχείο.

Makefiles (Αυτοματοποίηση Μεταγλώττισης)

- Οι περισσότεροι μεταγλωττιστές μας επιτρέπουν να οικοδομήσουμε ένα πρόγραμμα σε ένα μόνο βήμα.
- Μια GCC εντολή που χτίζει το `justify`:

```
gcc -o justify justify.c line.c word.c
```

- Τα τρία αρχεία συγκεντρώνονται πρώτα σε κώδικα αντικειμένου.
- Στη συνέχεια, τα αρχεία αντικειμένων μεταβιβάζονται αυτόματα στο πρόγραμμα σύνδεσης, το οποίο τα συνδυάζει σε ένα αρχείο.
- Η επιλογή `-o` καθορίζει ότι θέλουμε να ονομαστεί το εκτελέσιμο αρχείο σε `justify`.

Dr. Andreas Aristidou a.aristidou@ieee.org

Makefiles (Αυτοματοποίηση Μεταγλώττισης)

- Για την ευκολότερη μεταγλώττιση μεγάλων προγραμμάτων, το UNIX παρέχει την έννοια των ***makefile.***
  - **Την οποία θα χρησιμοποιήσουμε για τις εργασίες 3-4**

- 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**.

Dr. Andreas Aristidou a.aristidou@ieee.org

Makefiles (Αυτοματοποίηση Μεταγλώττισης)

Target (Στόχος):

Εξαρτήσεις:

Κανόνες (Rules):
4 Ομάδες Εντολών

- Makefile για το πρόγραμμα `game` :

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

game.o
```

Προτιμήστε το γενικό makefile που ακολουθεί σε λίγες διαφάνειες!

Εντολές:

Compile but don't link

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

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

Η πρώτη γραμμή σε κάθε κανόνα δίνει ένα αρχείο ***target***, που ακολουθείται από τα αρχεία στα οποία εξαρτάται.

ommand να εκτελεστεί εάν το υαστεί λόγω μιας αλλαγής σε ένα υ.

Η πρώτη γραμμή αναφέρει ότι η `game` εξαρτάται από τα αρχεία `game.o`, `stack.o`, και `queue.o`.

Εάν κάποιο από αυτά τα αρχεία έχει αλλάξει από την τελευταία φορά που χτίστηκε το πρόγραμμα, η `game` πρέπει να ξαναχτιστεί

Στην επόμενη γραμμή εμφανίζεται ο τρόπος ενημέρωσης του `game.o` (με την επαναμεταγλώττιση του `game.c`).
Η εντολή `-c` λέει στο μεταγλωττιστή να μεταγλωττίσει το `game.c` αλλά δεν προσπαθεί να το συνδέσει.

Dr. Andreas Aristidou a.aristidou@ieee.org

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.**

Dr. Andreas Aristidou a.aristidou@ieee.org

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)

Dr. Andreas Aristidou a.aristidou@ieee.org

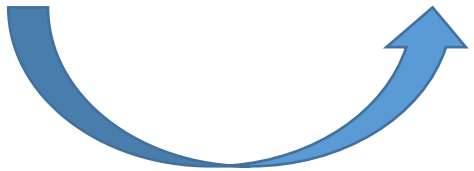
Ένα Γενικό Makefile για Όλα τα Προγράμματα

```
##############################################
# 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
```


Dr. Andreas Aristidou a.aristidou@ieee.org

Επανα-μεταγλώττιση Προγράμματος με Makefile

- Υποθέστε ότι ένα πρόγραμμα έχει τροποποιηθεί στα ακόλουθα αρχεία: `stack.h`, `stack.c`, και `game.c`.
- Όταν το πρόγραμμα `game` επανα-μεταγλωττίζεται, το `make` θα διεκπεραιώσει **ΑΥΤΟΜΑΤΑ** τις ακόλουθες ενέργειες:
  1. Δημιουργία `game.o` από `game.c` (εφόσον `game.c` και `stack.h` έχουν αλλάξει).

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

  2) Δημιουργία `stack.o` απο `stack.c` (εφόσον `stack.c` και `stack.h` έχουν αλλάξει).

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

  3. Δημιουργία `game` συνδέοντας (linking) **`game.o`**, **`stack.o`**, και `queue.o` εφόσον `game.o` και `stack.o` έχουν αλλάξει).

Dr. Andreas Aristidou a.aristidou@ieee.org

Επανα-μεταγλώττιση Προγράμματος με Makefile

- Ένα από τα πλεονεκτήματα της χρήσης makefiles είναι ότι η επανα-μεταγλώττιση γίνεται αυτόματα.
- Εξετάζοντας την ημερομηνία κάθε αρχείου, η `make` μπορεί να προσδιορίσει ποια αρχεία έχουν αλλάξει από την τελευταία φορά που χτίστηκε το πρόγραμμα.
- Στη συνέχεια, μεταγλώττιση αυτά τα αρχεία, μαζί με όλα τα αρχεία που εξαρτώνται από αυτά, είτε άμεσα είτε έμμεσα.

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`

    Σε αυτό το παράδειγμα, το macro της `DEBUG` ορίζεται να έχει την τιμή 1 στο πρόγραμμα `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

Dr. Andreas Aristidou a.aristidou@ieee.org

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

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

all :

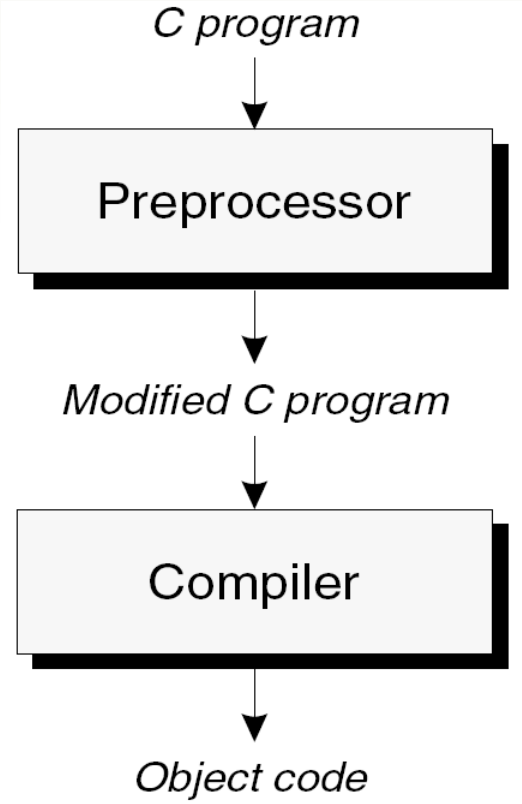
make
make doxy

Dr. Andreas Aristidou a.aristidou@ieee.org

Preprocessor Directives

Οδηγίες Προεπεξεργαστή

- 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



Preprocessor Directives

Οδηγίες Προεπεξεργαστή

- Ο Προεπεξεργαστής αναζητά ***οδηγίες προεπεξεργασίας,*** οι οποίες ξεκινούν με # χαρακτήρα***.***
- Μέχρι τώρα συναντήσαμε την `#define` και την `#include`.
- Η οδηγία `#define` δηλώνει ένα ***macro***—ένα όνομα που αντιπροσωπεύει κάτι άλλο, όπως μια σταθερά.
- Ο Προεπεξεργαστής αποκρίνεται σε μια οδηγία `#define` με την αποθήκευση του ονόματος της μακροεντολής μαζί με τον ορισμό της.
- Όταν η μακροεντολή χρησιμοποιείται αργότερα, ο Προεπεξεργαστής "επεκτείνει" τη μακροεντολή, αντικαθιστώντας την με την καθορισμένη τιμή της.
- Η οδηγία `#include` λέει στον προεπεξεργαστή να ανοίξει ένα συγκεκριμένο αρχείο και να "συμπεριλάβει" τα περιεχόμενά του ως μέρος του αρχείου που μεταγλωττίζεται.
- Για παράδειγμα, η γραμμή

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

  αναθέτει στον προεπεξεργαστή να ανοίξει το αρχείο με το όνομα `stdio.h` και να φέρει το περιεχόμενό της στο πρόγραμμα.

Dr. Andreas Aristidou a.aristidou@ieee.org

Preprocessor Directives

Οδηγίες Προεπεξεργαστή

```
/* Converts a Fahrenheit temperature to Celsius */

#include <stdio.h>

#define FREEZING_PT 32.0f
#define SCALE_FACTOR (5.0f / 9.0f)

int main(void)
{
  float fahrenheit, celsius;

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

  celsius = (fahrenheit - FREEZING_PT) * SCALE_FACTOR;
  printf("Celsius equivalent is: %.1f\n", celsius);

  return 0;
}
```

Οδηγίες όπως οι `#define` και `#include` αντιμετωπίζονται από τον προεπεξεργαστή, ένα κομμάτι του λογισμικού που επεξεργάζεται τα προγράμματα C ακριβώς πριν από τη μεταγλώττιση.

Dr. Andreas Aristidou a.aristidou@ieee.org

Preprocessor Directives

Οδηγίες Προεπεξεργαστή

Δείτε το μόνοι σας!
gcc –E example.c

- 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;
}
```

Ο Προεπεξεργαστής κάνει κάτι παραπάνω από το να εκτελεί οδηγίες.

Συγκεκριμένα, αντικαθιστά κάθε σχόλιο με ένα single space character.

Ορισμένοι προεπεξεργαστές πηγαίνουν περαιτέρω και καταργούν τους περιττούς χαρακτήρες κενού χώρου, συμπεριλαμβανομένων των κενών διαστημάτων και των καρτελών στην αρχή των γραμμών.

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
```

Dr. Andreas Aristidou a.aristidou@ieee.org

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)
…
#else
#error No operating system specified
#endif
```

- **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

Χρήση 5: Απλές Συναρτήσεις

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

Εάν ο προεπεξεργαστής συναντήσει `#error`, εκτυπώνει ένα μήνυμα σφάλματος (Ορισμένοι μεταγλωττιστές τερματίζουν αμέσως τη μεταγλώττιση χωρίς να επιχειρούν να βρουν άλλα σφάλματα).

Dr. Andreas Aristidou a.aristidou@ieee.org

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++;
```

Dr. Andreas Aristidou a.aristidou@ieee.org

Πλεονεκτήματα & Μειονεκτήματα

Παραμετροποιημένες Μακροεντολές

- Η Χρήση **παραμετρο-ποιημένων μακροεντολών** αντί **πραγματικών συναρτήσεων** έχει κάποια **πλεονεκτήματα** και **μειονεκτήματα:**
- **Πλεονεκτήματα:**
  - Α. 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** κατά την μεταγλώττιση
  - **Β. 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.

Dr. Andreas Aristidou a.aristidou@ieee.org

Πλεονεκτήματα & Μειονεκτήματα

Παραμετροποιημένες Μακροεντολές

- **Μειονεκτήματα (μερικά):**
  - ***Α. Ο 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))));
```

  - Β. 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!**

Dr. Andreas Aristidou a.aristidou@ieee.org