



# ΕΠΛ232 – Προγραμματιστικές Τεχνικές και Εργαλεία

## Διάλεξη 11: Οργάνωση Προγραμμάτων σε Πολλαπλά Αρχεία

(Κεφάλαιο 15.1-15.2, ΚΝΚ-2ΕΔ)

**Δημήτρης Ζεϊναλιπούρ**

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

# Περιεχόμενο Διάλεξης 11



- **Αρχεία Πηγαίου Κώδικα (Source Files)**
  - Μελέτη Παραδείγματος: Αριθμομηχανή σε 1 Αρχείο
    - Περιγραφή Προβλήματος: Αντίστροφος Πολωνικός Συμβολισμός
    - Περιγραφή Λύσης και Ψευδοκώδικας.
- **Οργάνωση Προγ. σε Πολλαπλά Αρχεία**
  - Αριθμομηχανή σε 3 Αρχεία (Προσέγγιση A:  $3 * c$ )
  - Μεταγλώττιση Προγρ. Πολλαπλών Αρχείων με GCC
- **Αρχεία Κεφαλίδας (Header Files)**
  - Αριθμομηχανή σε Πολλα. Αρχεία (Προσέγγιση B:  $1 * h + 3 * c$ )
  - Η οδηγία προεπεξεργαστή `#include`
  - Διαμοιρασμός Μεταβλητών μεταξύ Αρχείων και `extern`.
- **Τελική Προσέγγιση: Διάλεξη 12!**

# Αρχεία Πηγαίου Κώδικα (Source Files)



- Στις πλείστες γλώσσες προγραμματισμού, ένα πρόγραμμα μπορεί να χωριστεί σε **πολλαπλά αρχεία πηγαίου κώδικα**
- **Λόγοι:**
  - Καλύτερη **Άρθρωση Κώδικα**
  - **Ταχύτητα Μεταγλώττισης:** Κάθε αρχείο μπορεί να αναπτυχθεί, μεταγλωττιστεί, δοκιμαστεί ξεχωριστά, πράγμα το οποίο γλιτώνει πολύτιμο χρόνο!
  - **Επαναχρησιμοποίηση Κώδικα:** Οι συναρτήσεις γίνονται πιο γενικές (μείωση εμβέλειας μεταβλητών)
  - **Ανάπτυξη σε Ομάδες:** Δυνατότητα ταυτόχρονης ανάπτυξης (σε ομάδα προγραμματιστών όπου καθένας δουλεύει σε ένα υποσύνολο αρχείων).

# Μελέτη Παραδείγματος

## Αριθμομηχανής σε 1 Αρχείο



- Υποθέστε ότι θέλουμε να γράψουμε ένα πρόγραμμα για μια απλή υπολογιστική μηχανή σε **1 Αρχείο** αρχικά.
- Το πρόγραμμα θα αποτιμά **αριθμητικές εκφράσεις** οι οποίες θα δίνονται στον **Αντίστροφο Πολωνικό Συμβολισμό (Reverse Polish notation (RPN))**
  - **Επιθεματική Αναπαράσταση (postfix)**: Ο Τελεστής ακολουθεί του τελεστέους, π.χ.,
    - **30 5 - 7 \*** ισοδυναμεί την ενθεματική αριθμητική έκφραση  $(30-5) * 7 = 25 * 7 = 175$
    - **3 6 + 8 6 - \*** ισοδυναμεί την ενθεματική αριθμητική έκφραση  $(3+6) * (8-6) = 9 * 2 = 18$
  - Υπάρχει αντίστοιχα και η **προθεματική (prefix)** (+ 3 4) αναπαράσταση αλλά δεν μας ενδιαφέρει στα πλαίσια αυτής της συζήτησης.

# Μελέτη Παραδείγματος

## Αριθμομηχανής σε 1 Αρχείο



- Μια RPN έκφραση δεν απαιτείται η χρήση παρενθέσεων, όπως είδαμε νωρίτερα.

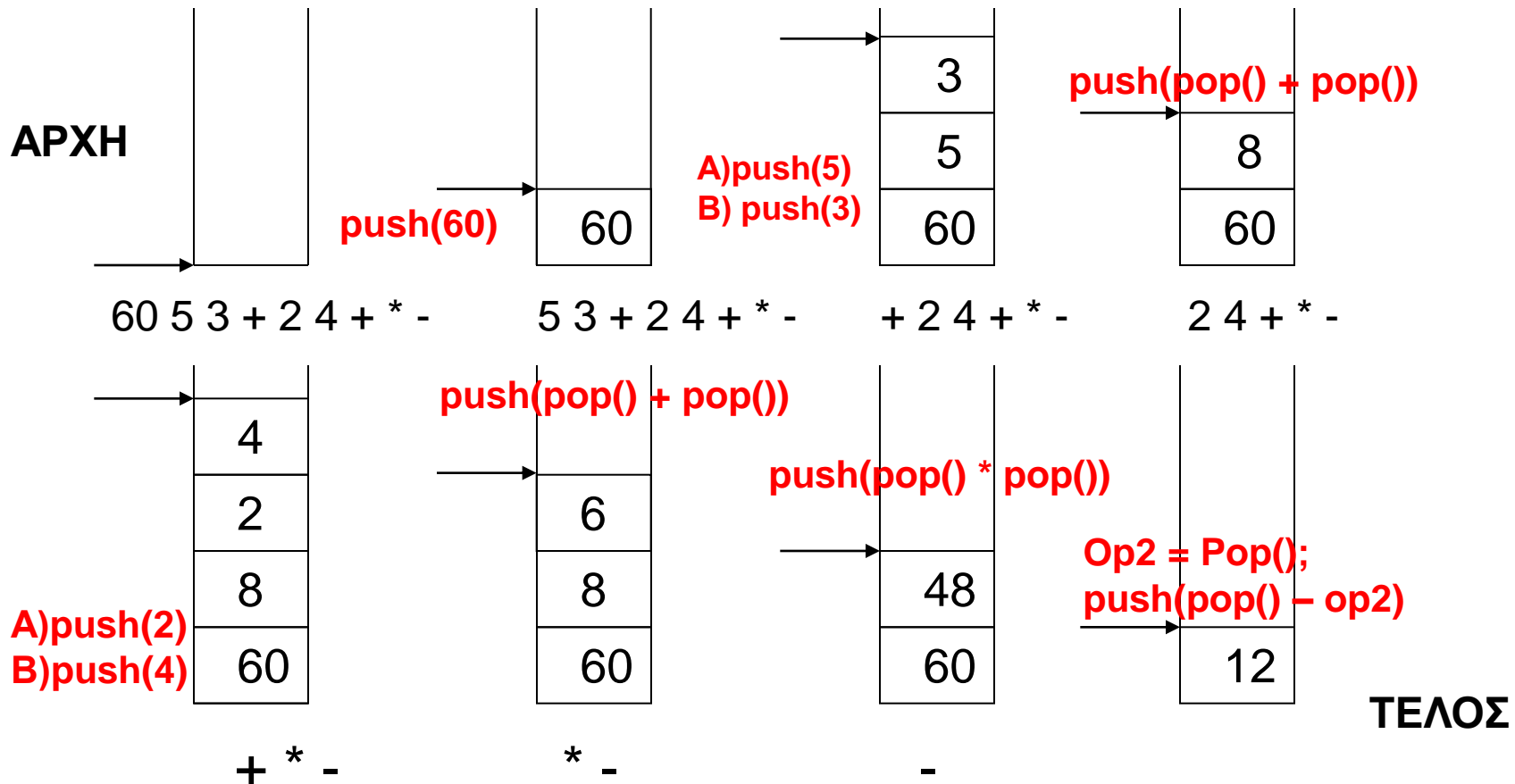
30 5 - 7 \*

- Για τον υπολογισμό τέτοιων εκφράσεων μπορεί να χρησιμοποιηθεί η έννοια της **Στοίβας (stack)**: μιας λίστας που συνοδεύεται από τις διαδικασίες:
  - **push**, για εισαγωγή στοιχείου στο τέλος της λίστας.
  - **pop**, για εξαγωγή του τελευταίου στοιχείου της λίστας.
- Ακολουθεί η **λογική αποτίμησης** εκφράσεων με την **στοίβα** και στη **συνέχεια** θα δοθεί ο **αναλυτικός ψευδο-κώδικας**.

# Μελέτη Παραδείγματος Αριθμομηχανής σε 1 Αρχείο



- Έστω η παράσταση  $60\ 5\ 3\ +\ 2\ 4\ +\ *\ -\ .$   
ή σε ενθεματική μορφή:  $60 - (5 + 3) * (2 + 4) = 12$



# Υλοποίηση Αριθμομηχανής σε 1 Αρχείο ( Ψευδοκώδικας )



```
#include <stdio.h>
#include <stdlib.h>
#define MAXOP      100
#define NUMBER    '0'

int getop(char []);
void push(double);
double pop(void);

int main(void) {
    int type;
    double op2;
    char s[MAXOP];

    while ( (type = getop(s)) != EOF) {
        switch (type) {
            case NUMBER:  push(atof(s));          break;
            case '+':     push( pop() + pop() );  break;
```

Υλοποίηση με ΣΤΟΙΒΑ σταθερού μεγέθους!

Η συνάρτηση αυτή θα διαβάζει τον επόμενο τελεστή ή τελεστέο από τον χρήστη στον πίνακα s[MAXOP] και επιστρέφει σταθερά NUMBER εάν διαβάζεται τελεστέος.

Στοίβα Πραγματικών Αριθμών

# Υλοποίηση Αριθμομηχανής σε 1 Αρχείο



```
case '*': push( pop() * pop() );
          break;
case '-': op2 = pop();
          push( pop() - op2 );
          break;
case '/': op2 = pop();
          if (op2!=0.0)
              push(pop()/op2);
          else printf("error: zero divisor\n");
          break;
case '\n': printf("result=%.2f\n", pop());
           break;
default: printf ("error: Unknown command\n");
         break;
}
}
return 0;
```

## Προβλήματα

- Άσχημη Οργάνωση
- Δύσκολη Συντήρηση
- ...



# Οργάνωση Προγράμματος σε Πολλαπλά Αρχεία



- Ψάχνοντας μια καλύτερη οργάνωση του παραπάνω προγράμματος χρησιμοποιούμε τη δυνατότητα **κατάτμησης ενός προγράμματος σε περισσότερα από ένα αρχεία**.
- **Λογική Κατάτμησης Προγράμματος:**
  1. Η συνάρτηση **main()** να τοποθετηθεί σε ένα αρχείο το οποίο να ονομαστεί **main.c** (ή **calc.c**)
  2. Οι συναρτήσεις που υλοποιούν τη **στοίβα** (**push**, **pop**, **make\_empty**, **is\_empty**, **and is\_full**) και οι μεταβλητές της να τοποθετηθούν στο αρχείο **stack.c**
  3. Άλλες συναρτήσεις που επεξεργάζονται την είσοδο και βρίσκουν τους τελεστές και τους τελεστέους να τοποθετηθούν σε ένα τρίτο αρχείο, το **getop.c** (ή **token.c**)

# Οργάνωση Προγράμματος σε Πολλαπλά Αρχεία



Πρώτη Προσέγγιση: 3 Αρχεία c (με 1 main)  
(η οποία θα βελτιωθεί στις επόμενες διαφάνειες)

main.c

```
#include <stdio.h>
#include <stdlib.h>
#define MAXOP 100
#define NUMBER '0'
/* prototypes */
int getop(char []);
void push(double);
double pop(void);

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

stack.c

```
#include <stdio.h>
#define MAXVAL 100
#define NUMBER '0'

int sp = 0;
double val[MAXVAL];

void push(double f) {
    .....
}

double pop(void) {
    .....
}
```

getop.c

```
#include <stdio.h>
#include <ctype.h>
#define NUMBER '0'

int getop(char s[]) {
    .....
}
```

# Μεταγλώττιση Πολλαπλών Αρχείων με GCC



Στο παραπάνω παράδειγμα μπορούμε να χρησιμοποιήσουμε τις εξής εντολές:

1. **gcc -c main.c** Δημιουργία του **main.o**
2. **gcc -c stack.c** Δημιουργία του **stack.o**
3. **gcc -c getop.c** Δημιουργία του **getop.o**

Compile (Not Link): Το αρχείο δεν χρειάζεται main()

Reject non-standard features

Μην ξεχνάτε τα επιπλέον ορίσματα: `gcc -std=c99 -Wall -Wuninitialized -Wunreachable-code -pedantic file.c`

4. **gcc -o calculator main.o stack.o getop.o**

Compile (And Link): 1 αρχείο χρειάζεται main()

Σύνδεση των object files και η δημιουργία του εκτελέσιμου προγράμματος **calculator**

ή διαφορετικά σε ένα βήμα:

**gcc -o calculator main.c stack.c getop.c**

# Μεταγλώττιση Πολλαπλών Αρχείων με GCC



- Στο προηγούμενο παράδειγμα η **gcc -c main.c** δημιουργεί το αντικειμενικό αρχείο (object file) **main.o**
- Θα έχετε προσέξει ότι το main.c αναφέρεται σε **συναρτήσεις** που **ΔΕΝ βρίσκονται** στο ίδιο αρχείο.

- **Πως ακριβώς το αντιλαμβάνεται ο μεταγλωττιστής;**

- Θυμίζουμε ότι **Μεταγλώττιση = Δημιουργία (Compile) ΚΑΙ Σύνδεση (Linking) Αντικειμενικών Αρχείων**

- **Compile:** Ο μεταγλώττισης βλέπει μια κλήση συνάρτησης αλλά δεν την αναζητεί (θεωρεί ότι θα βρεθεί κάπου στην πορεία)
- **Linking:** Ο μεταγλώττισης ψάχνει (το ορισμό) της συνάρτησης και δίνει σφάλμα σύνδεση εάν δεν την βρει.

```
// main.c
#include <stdio.h>
#include <stdlib.h>
#define MAXOP 100
#define NUMBER '0'
```

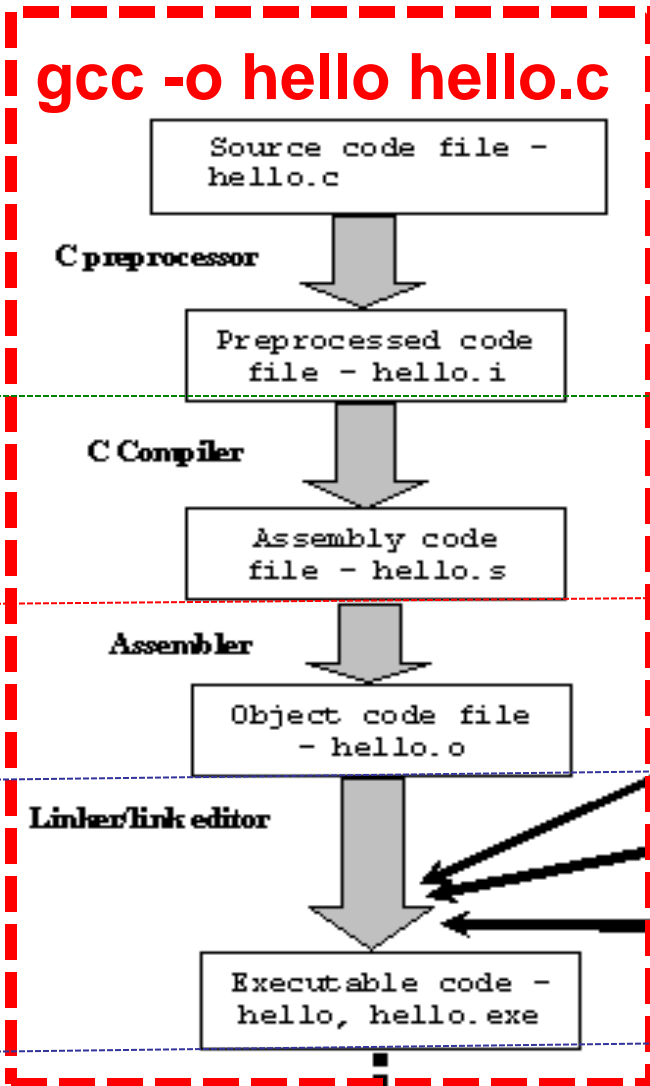
```
int getop(char[]);
void push(double);
double pop(void);
```

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

# Μεταγλώττιση με GCC (Αναλυτικά Βήματα)



**gcc -o hello hello.c**



**gcc -E hello.c > hello.i**

Οι ορισμοί των βιβλιοθηκών που περιλαμβάνονται στο #include, τοποθετούνται μαζί με το .c κώδικα στο .i αρχείο.

**gcc -c hello.c**

**gcc -S hello.c**

Παράγεται αρχείο **hello.s** που περιέχει τον συμβολικό κώδικα του προγράμματος.

**as -arch x86\_64 -o hello.o hello.s**

Ο συμβολομεταφραστής (assembler) AS ή GAS (GNU), μεταφράζει το συμβ. κώδικα σε κώδικα μηχανής x86\_64

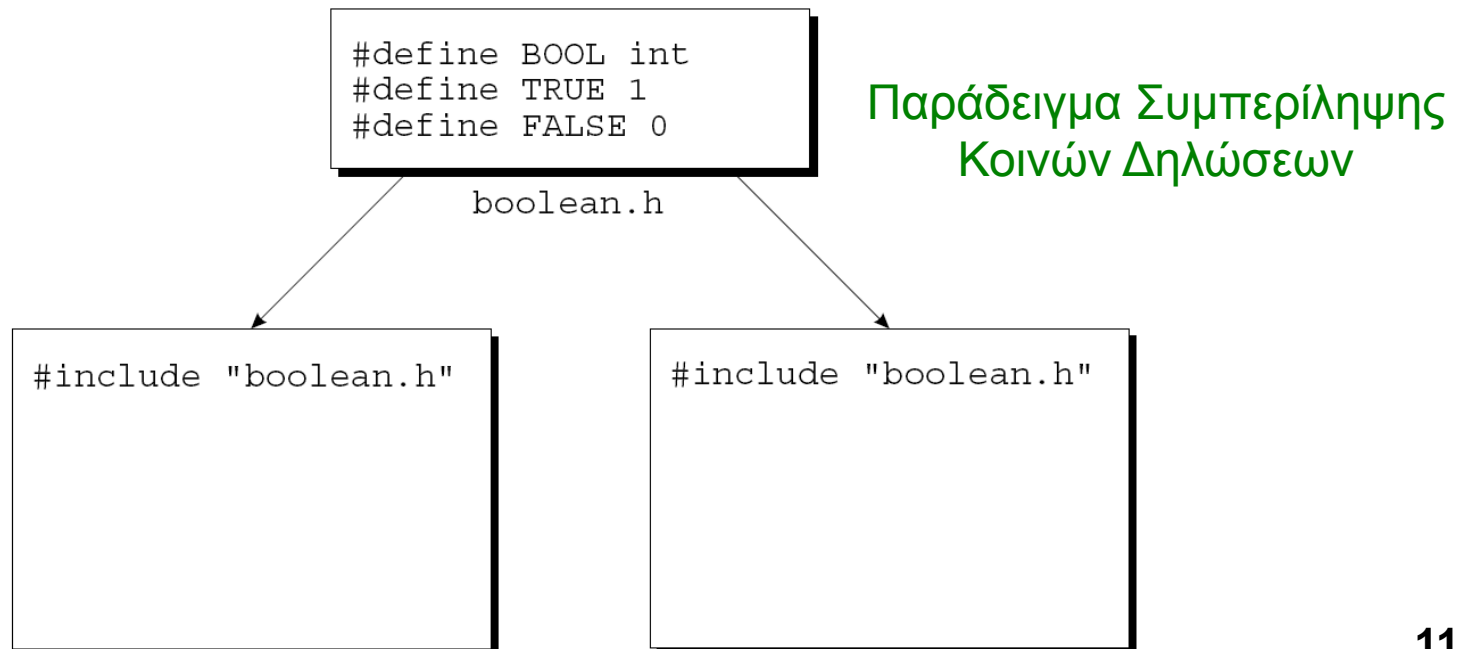
**ld -e \_main -o hello -lc hello.o**

Ο αυνδέτης (linker) παράγει το τελικό εκτελέσιμο.

# Αρχεία Κεφαλίδας (.h) (Header Files)



- Ένα ακόμα θέμα που μας απασχολεί καθώς μοιράζουμε ένα πρόγραμμα σε περισσότερα από ένα αρχεία είναι οι **κοινοί ορισμοί** και οι **δηλώσεις μεταξύ των αρχείων** αυτών.
  - Π.χ., **#define NUMBER '0'** (που βρισκόταν σε όλα τα αρχεία)
- Προσπαθούμε λοιπόν ως τελευταίο βήμα της παραπάνω διαδικασίας οργάνωσης, να μαζέψουμε κεντρικά σε ένα αρχείο όλες τις δηλώσεις το οποίο ονομάζεται **header file**.



# Αρχεία Κεφαλίδας (.h) (Header Files)



- Στην περίπτωση του παραδείγματος της αριθμομηχανής κατασκευάζουμε το αρχείο ***calc.h*** το οποίο έχει το εξής περιεχόμενο:

```
calc.h  
#define NUMBER '0'  
void push(double);  
double pop(void);  
int getop(char []);
```

Κοινές Δηλώσεις

Πρότυπα  
Συναρτήσεων

- Το **αρχείο κεφαλίδα ΔΕΝ** χρειάζεται ξεχωριστή μεταγλώττιση.
- Το προτεινόμενο πρόγραμμα γίνεται συνεπώς αυτό που φαίνεται στην επόμενη διαφάνεια:

# Αρχεία Κεφαλίδας (.h) (Header Files)



Δεύτερη Προσέγγιση: 3 Αρχεία .c και 1 .h  
(η οποία θα βελτιωθεί στις επόμενες διαφάνειες)

Η οδηγία `#include` λέει του προεπεξεργαστή να προσθέσει το περιεχόμενο του `calc.h` στο σημείο της εντολής

## main.c

```
#include <stdio.h>
#include <stdlib.h>
#include "calc.h"
#define MAXOP 100

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

## stack.c

```
#include <stdio.h>
#include "calc.h"
#define MAXVAL 100

int sp = 0;
double val[MAXVAL];
void push(double f) {
    .....
}
double pop(void) {
    .....
}
```

## getop.c

```
#include <stdio.h>
#include <ctype.h>
#include "calc.h"

int getop(char s[]) {
    .....
}
```

- Εάν εμφανιστούν σε άλλο αρχείο δημιουργούνται συγκρούσεις στο όνομα.
- Οπότεν πρέπει να δηλωθεί ως `static`

```
calc.h
#define NUMBER '0'
void push(double);
double pop(void);
int getop(char []);
```



# Αρχεία Κεφαλίδας (.h) (Header Files)



- Τα ονόματα των μεταβλητών `val` και `sp` (στο `stack.c`) είναι για αποκλειστική χρήση των συναρτήσεων `push()` και `pop()`.
  - **Ερώτηση:** Τι γίνεται αν οριστούν ξανά (κατά λάθος) σε ένα άλλο αρχείο;
  - **Απάντηση:** Θα έχουμε δύο εξωτερικές μεταβλητές με το ίδιο όνομα!!!
- **Λύση:** Ορίζοντας τις **εξωτερικές μεταβλητές** `val` και `sp` στο αρχείο `stack.c` ως **static** περιορίζουμε την **εμβέλεια των μεταβλητών αυτών στο αρχείο το οποίο ορίζονται (file scope)**.
  - Έτσι, **αποφεύγεται η σύγκρουση** σε δηλώσεις συνώνυμων μεταβλητών **σε άλλα αρχεία**.
  - Στο αρχείο **`stack.c`** έχουμε συνεπώς τις δηλώσεις:

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

# Οδηγίες Προεπεξεργαστή

## #include



- Χρησιμοποιείται για να προστίθεται περιεχόμενο άλλου αρχείου στο σημείο της κλήσης
  - Προσθήκη Βιβλιοθήκης  
`#include <filename>`
  - Προσθήκη Header File (Δικά μας αρχεία κεφαλίδας)  
`#include "filename"`
- Η διαφορά μεταξύ των δυο **έγκειται στο πως ο μεταγλωττιστής βρίσκει το αρχείο κεφαλίδας.**
  - `#include <filename>`: Ψάξε τους καταλόγους στο οποίοις το σύστημα αποθηκεύει τα header file.
    - Στο UNIX: `/usr/include` ή `gcc -I/add/somepath`
  - `#include "filename"`: Ψάξε στον τρέχων κατάλογο και μετά ψάξε τους καταλόγους των header files

# Οδηγίες Προεπεξεργαστή

## #include



- **Άσχημη Ιδέα: Απόλυτα Μονοπάτια**

- Δυσχεραίνεται η μεταγλώττιση σε άλλα συστήματα

```
#include "c:\cprogs\utils.h" /* Windows*/  
#include "/cprogs/utils.h" /* UNIX */  
#include <myheader.h> /*** WRONG ***/
```

- **Καλή Ιδέα: Σχετικά Μονοπάτια**

```
#include "utils.h"
```

- Προσοχή: slash (/ σε UNIX) και backslash (\ σε Windows)

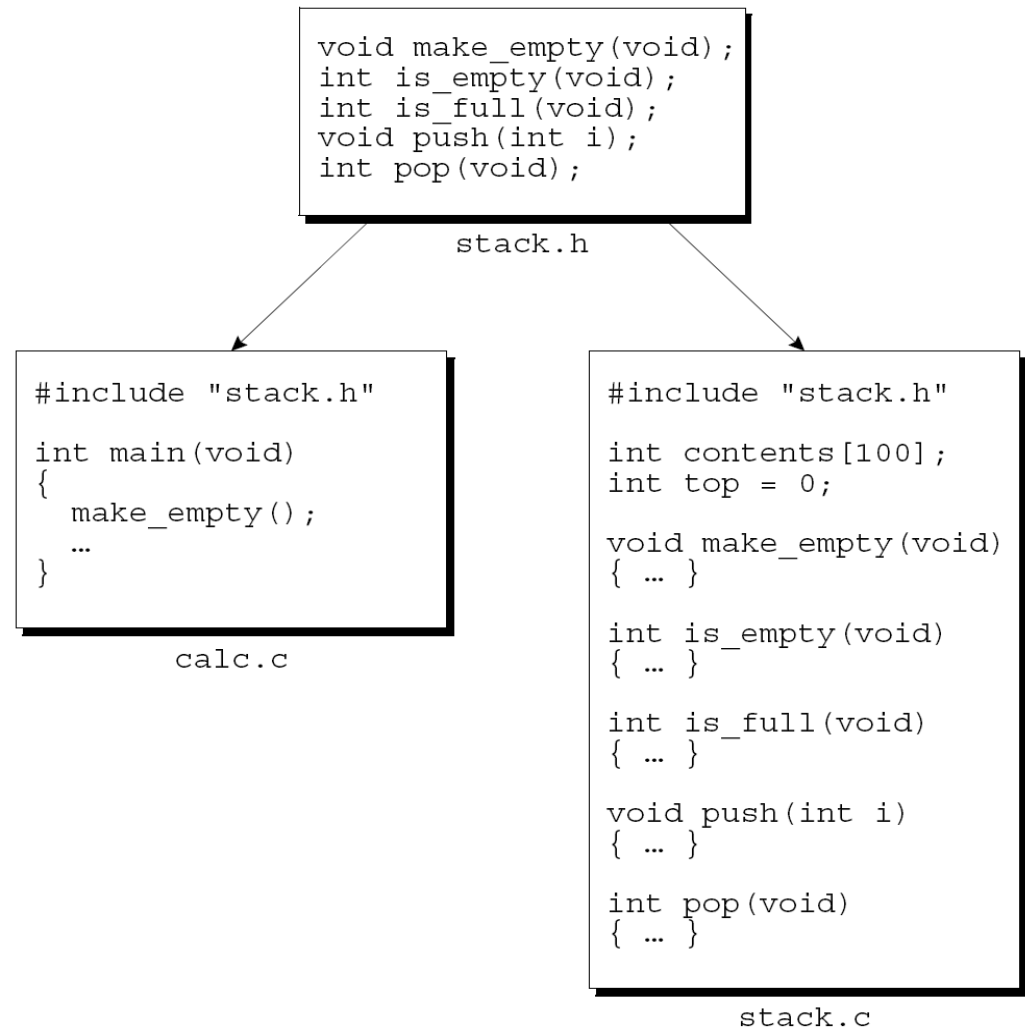
```
#ifdef __unix__  
    #include "..\include\utils.h"  
#elif defined _WIN32  
    #include "../include/utils.h"  
#endif
```

Θα ξαναμιλήσουμε  
στην διάλεξη 12 για  
include και define

# Αρχεία Κεφαλίδας (.h) (Header Files)



Οργάνωση της  
Υπολογιστικής  
Μηχανής βάσει της  
παρόμοιας λογικής  
στο βιβλίο του  
μαθήματος ΚΝΚ/2Ε



# Διαμοιρασμός Μεταβλητών Μεταξύ Αρχείων



- Παράδειγμα δήλωσης και ορισμού (δέσμευσης χώρου) μεταβλητής `i: int i;`
  - ο μεταγλωττιστής δεσμεύει τον απαιτούμενο χώρο,
  - η μεταβλητή έχει **εμβέλεια αρχείου**, συνεπώς **ΔΕΝ** είναι **ορατή** από άλλα αρχεία.
- Η λέξη κλειδί **extern** χρησιμοποιείται για να δηλώσουμε μια μεταβλητή χωρίς να την ορίσουμε (να δεσμευθεί χώρος):  
`extern int i;`
- Το `extern` ενημερώνει τον μεταγλωττιστή ότι το `i` είναι **ορισμένο κάπου αλλού στο πρόγραμμα**, έτσι ώστε να μην δεσμευθεί επιπλέον χώρος για την μεταβλητή.  
`extern int a[]; // είναι OK (χωρίς μέγεθος)!`

# Διαμοιρασμός Μεταβλητών μεταξύ Αρχείων



- Για να **διαμοιράσουμε** λοιπόν μια **μεταβλητή** `i` **μεταξύ διαφορετικών** αρχείων, πρώτα την ορίζουμε (define) σε ένα αρχείο: `int i;`
- Τα άλλα αρχεία θα περιέχουν δηλώσεις: `extern int i;`
- Με αυτό τον τρόπο γίνεται **εφικτό** να **προσπελάνουμε** και να **μεταβάλλουμε** το `i` μέσα στα αρχεία αυτά.
- **Προσοχή**, το `extern` πρέπει να χρησιμοποιείται με **φειδώ**, εφόσον **αναιρεί** την εξ' ορισμού **εμβέλεια αρχείου** που έχουν οι **μεταβλητές**.