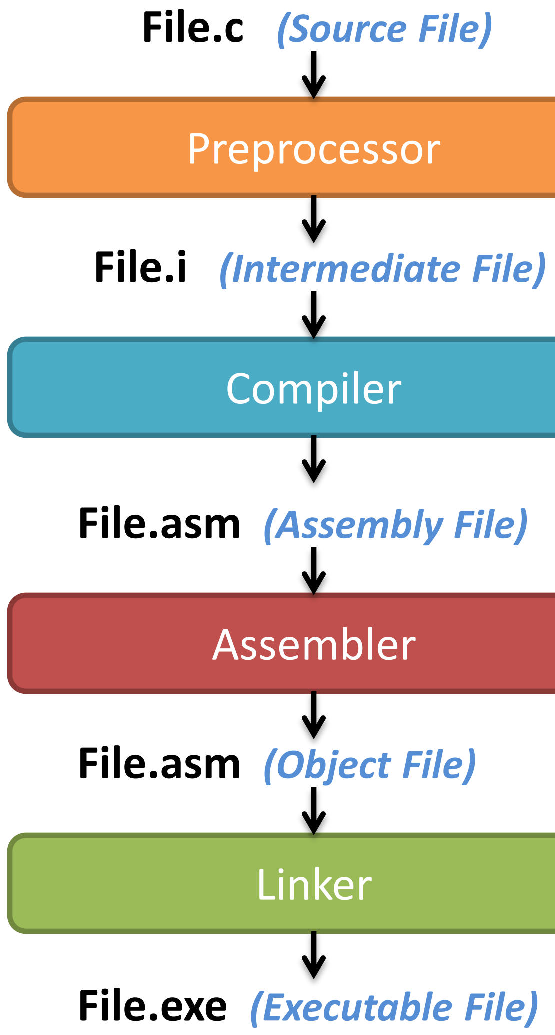
C Programming

Lecture nine

Preprocessor directives

C Building Process



Cmd

```
gcc   -E   file.c   -o  out.i
```

Cmd

```
gcc   -S   file.c   -o  out.asm
```

Cmd

```
gcc   -c   file.c   -o  out.o
```

Cmd

```
gcc        file.c   -o  out.exe
```

LAB 1

Write C code to print your name, generate the following files:

1- The intermediate file
2- The assembly file
3- The object file
4- The executable file

Time To

Code

Preprocessor Directives

The C Preprocessor:

1- First part of the building process

2- It concerns only with its directives which always start with # sign

3- It generates an intermediate file written in C language after executing all directives

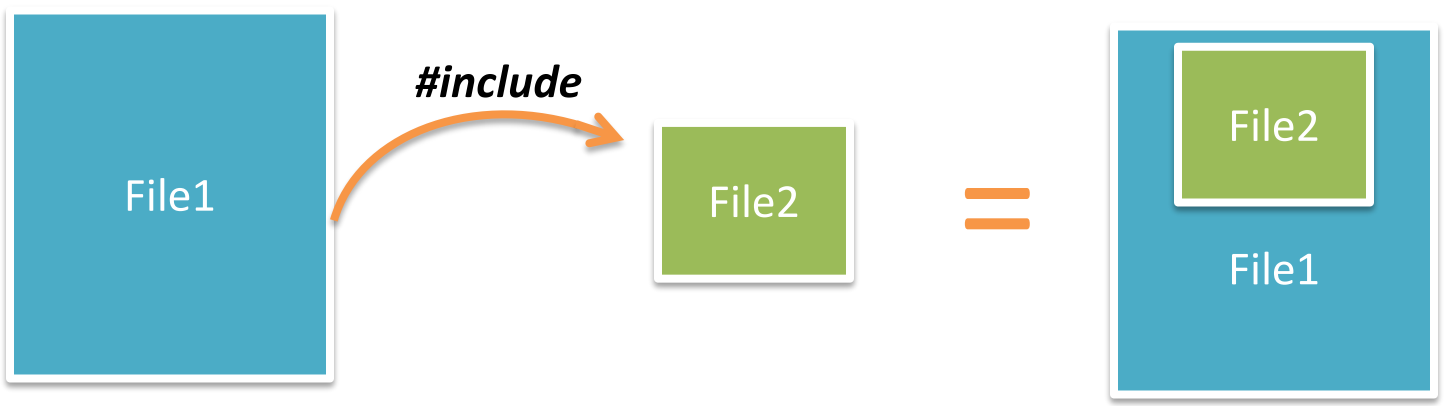
The Preprocessor Directives:

- #include
- #define
- #warning
- #error
- **Conditional Directives** →

#if
#else
#elif
#ifdef
#ifndef
#endif

#include Directive

This directive includes a file content into another file, it just copy the file content without checking it.



File 1 will now get the content of file 2 whatever was this content, even if it is not C code !

Preprocessor is a just text replacement

#include Syntax

#include <   file name   >

In case of standard or default file

#include “ file path “

In case of user file

Syntax 1   *#include <   file name   >*

This is used to include a file from a predetermined directory.

Example:

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

#include directive

Syntax 2 ***#include " file path "***

This is used to include a file from any directory. The path may be absolute or relative.

Absolute Path

```
#include "C:\Users\Ahmed\Desktop\IMT_Labs\Lab106\file.h"
```

The absolute path is not recommended because:

1- It is very long.
2- It is not portable, if you moved the code to another machine it wouldn't work because the new machine doesn't have the same path, So you have to edit the code

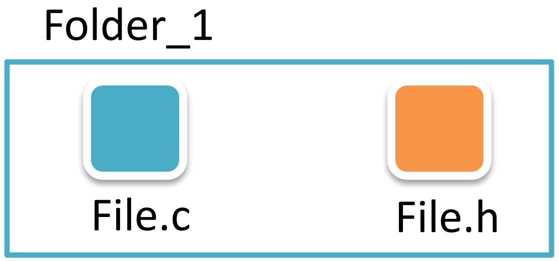
#include directive

Relative Path

The path from the source file till the header file

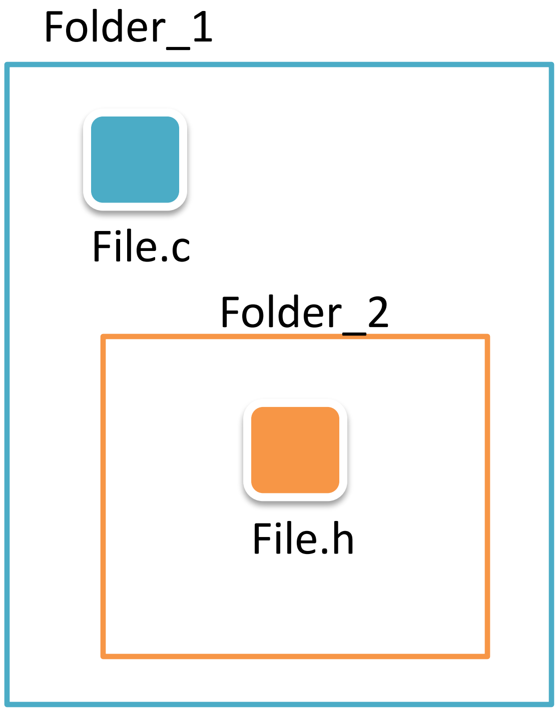
- The header file in the same folder of the source file

```
#include "File.h"
```



- The header file is in inner folder from the source file

```
#include "Folder2/File.h"
```



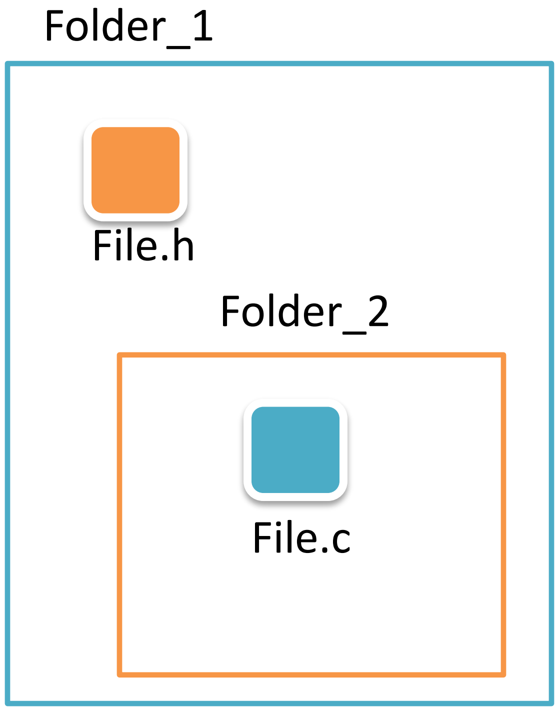
#include directive

Relative Path

The path from the source file till the header file

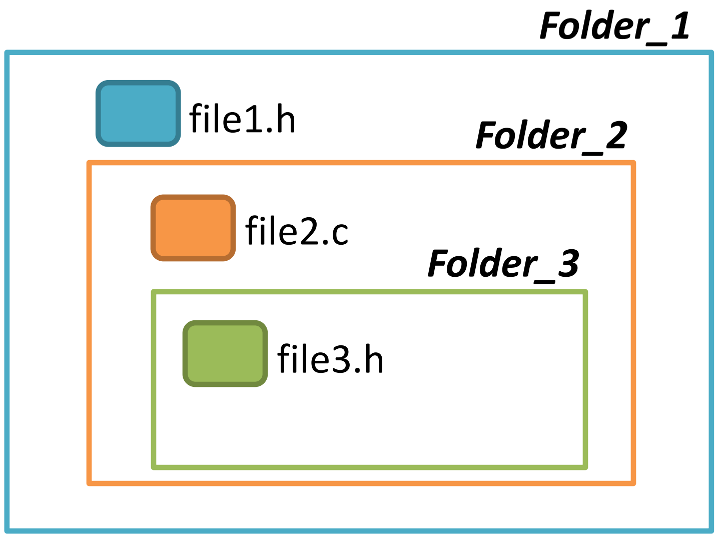
- The header file in outer folder of the source file

```
#include "../File.h"
```



LAB 2

Write C code to print the value of two variables, each one of them is defined in a header file structured as follow.

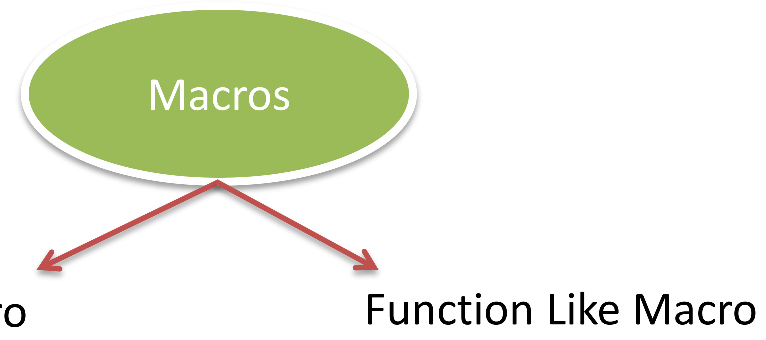


Time To

Code

#define directive

Also called Macro, and it has two types …



- **Object Like Macro**

```
#define    x    100
```

The preprocessor then will replace x in all upcoming code with 100

Object Like Macro

Example 1

```
int y = x;
```

Preprocessor expand it to

```
int y = 100;
```

x will be replaced by 100 in the preprocessor stage and before the compilation stage

Object Like Macro

Example 2

```
printf("The value of x is ", x);
```

Preprocessor expand it to

```
printf("The value of x is ", 100);
```

x here will not be replaced as it is not alone, it is a part from a string

x here is replaced by 100

Object Like Macro

Example 3

```
int yx = 50;
```

Preprocessor expand it to

```
int yx = 50;
```

x here is not replaced as it is note alone, it is a part from a word

Object Like Macro

Example 4

```
int x = 50;
```

Preprocessor expand it to

```
int 100 = 50;
```

x will be replaced and the compiler will give an error

Object Like Macro

What are the benefits of using object like macros … ?

1- Configurability

2- Readability

3- Avoid magic numbers

4- Consumes no memory

Note, the object like macro is not a variable, so you cann’t assign a value to it or change its value in the run time ! it is like a nickname for certain value

function like Macro

Example

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

Opening a bracket after the macro name make it a function like macro

At any line of the code if you wrote for example:

```
int z = Add(3,4);
```

It would be expanded to:

```
int z = 3+4;
```

LAB 3

Write a header file that provides the following functions like macros:

1- ***Set_Bit(Var,BitNo)*** -> This macro will set the bit number (BitNo) in the variable (Var) to 1

2- ***Clr_Bit(Var,BitNo)*** -> This macro will set the bit number (BitNo) in the variable (Var) to 0

3- ***Toggle_Bit(Var,BitNo)*** -> This macro will toggle the bit number (BitNo) in the variable (Var)

Time To

Code

Multiline Macro

If you have a function like macro that consists of many lines, use the backslash \ and the end of each line except for the last line.

```
#define print()      printf("I");      \
                     printf("Like");   \
                     printf("IMT")
```

If the user call the macro in his code like this:

```
print();
```

It will be expanded to:

```
printf("I");
printf("Like");
printf("IMT");
```

Multiline Macro Problem

Consider the following piece of code:

```
if (x==10)
    print();

else
    printf("I'm in the else");
```

The user here relied on calling the function print is one line so no need for brackets !
He doesn't know that the print is a function like macro.

This code will expand to:

```
if (x==10)
    printf("I");
    printf("Like");
    printf("IMT");

else
    printf("I'm in the else");
```

Compilation error: else without if !

Solution

The only valid solution is to use do while (0) …

```
#define    print()  do { printf("I");      \
                         printf("Like");  \
                         printf("IMT");   \
                       }while(0)
```

#warning directive

This directive will generate a warning on the console while building the project, the waning doesn’t stop the building process

```
#warning "This is a warning"
```

#error directive

This directive will generate an error on the console while building the project, the error stops the building process, no output file will be generated

```
#error  "This is an eroor"
```

Conditional directives

Syntax:

```
#if
        /* Code */
#elif
        /*  Code */
#else
        /* Code */
#endif
```

Notes:

1- #elif and #else are optional
2- the #if statement must end with #endif
3- No { } are used

Conditional directives

```
#define x  100

#if x > 10
    printf("Ahmed");

#elif x < 10
    printf("Gemy");
#else
    printf("Mohamed");
#endif
```

Once the preprocessor finds the true condition, it will include the statements under that condition and remove all other code

Will be expanded to →

```
printf("Ahmed");
```

Conditional directives

The header file guard

Used to avoid many inclusion of the same header file, if you by mistake included the same header file many times, the guard will allow you to include it only one !

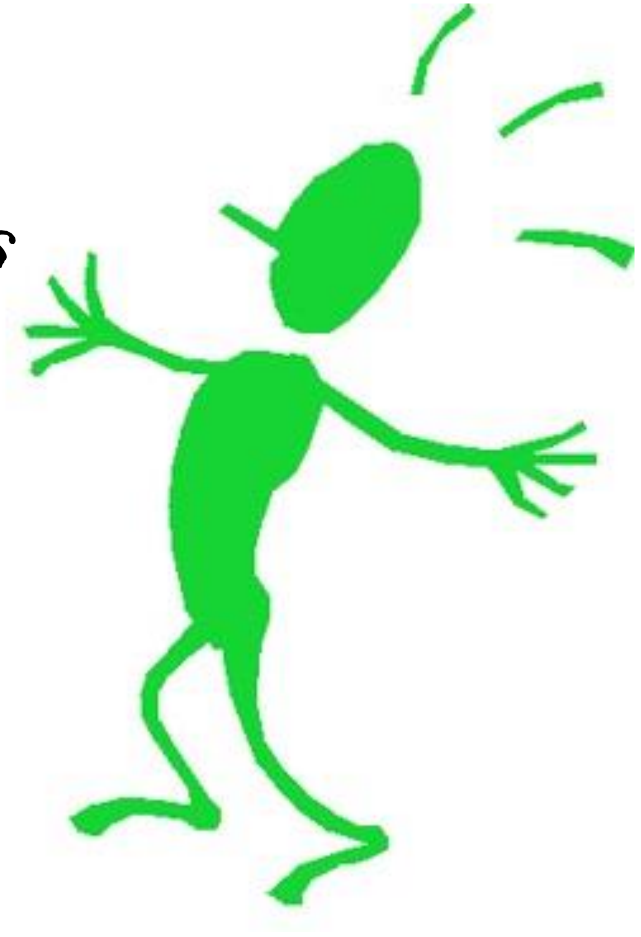
HeaderFile

```
#ifndef HeaderFile
#define HeaderFile

/* Header file Content */

#endif
```

The End ...

Any questions

... ?



IMT
SCHOOL

www.imtschool.com

ww.facebook.com/imaketechnologyschool/