The C preprocessor

By: Yehia M. Abu Eita

Outlines

- Introduction
- Includes
- Object-like macros
- Function-like macros
- Conditional directives
- Header file guard

Introduction

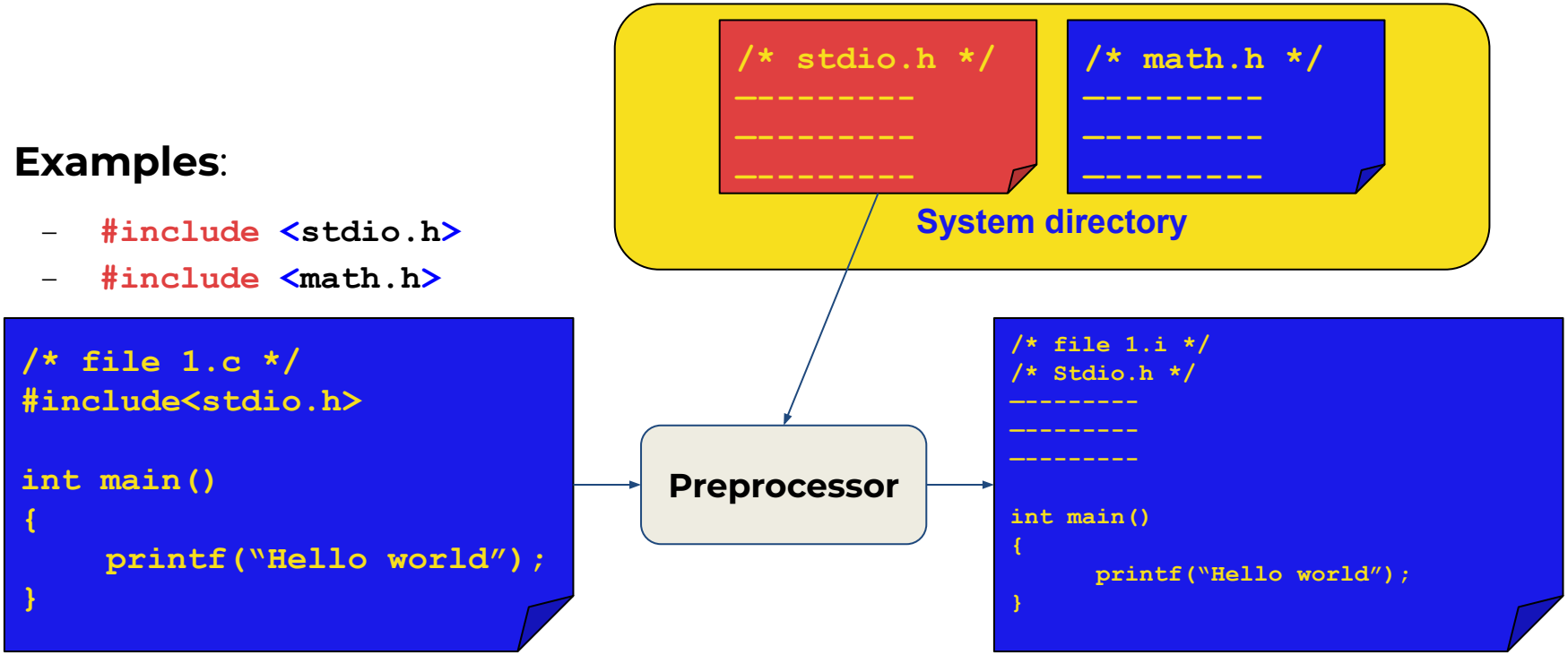
- The Preprocessor is a **text replacement tool** that replaces a **text by a value** corresponding to that text.
- **Input**: **.c** file.
- **Output**: **.i** file, **pure C** file.
- **Operations**:
  - **Deletes** any **comments**
  - **Removes** any **white spaces**
  - **Replaces** the **preprocessor directives**, after the ' **#** ', with their corresponding values **in iterations**
  - **Expand all macros, includes**, **and provide additional information to the compiler using #pragma**
- **No errors** can be **detected** by the preprocessor.

Includes

- **Including built-in C libraries**:
  - A **library in C** is basically a **.h (Header) file** that may be **built-in** or **user-defined**.
  - A **header file** in C may **contain**, **object-like macros**, **function-like macros**, **functions' prototypes**, and **typedefs**.
  - To **include a built-in C** library, `#include <library_name.h>`.
  - The preprocessor will **search for** this library **header file** into **system directory**.
  - If the **library is found**, the preprocessor will **copy its content to the include line**.
  - If the **library is not found**, the preprocessor will **make nothing** and the **compiler generates file not found error**.

Includes

- **Examples**:
  - `#include <stdio.h>`
  - `#include <math.h>`

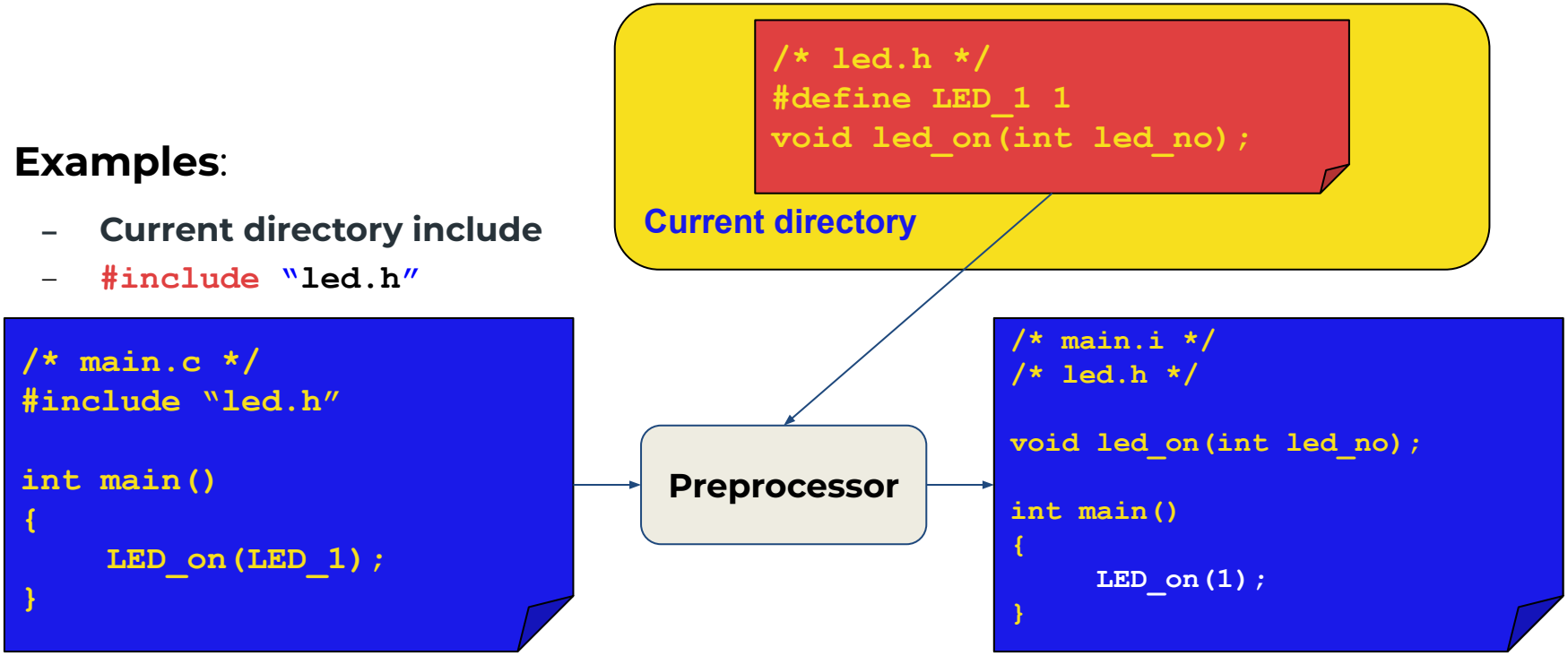


Includes

- **Including user-defined libraries**:
  - To **include a user-defined** library, `#include "library_name.h"`.
  - The preprocessor will **search for** this library **header file** into the **current directory**.
  - If the **library is found**, the preprocessor will **copy its content to the include line**.
  - If the **library is not found in the current directory**, the preprocessor will **search for** this library **header file** into **system directory.**
  - If the **library is not found**, the preprocessor will **make nothing** and the **compiler generates file not found error**.

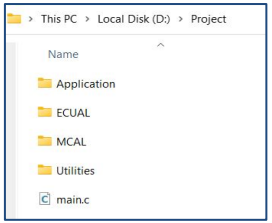
Includes

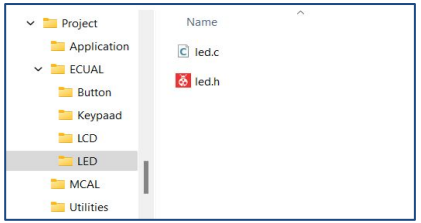
- **Examples**:
  - Current directory include
  - `#include "led.h"`

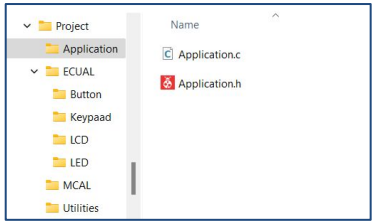


Includes

- **Examples**:
  - **Absolute path include**
  - `#include "D:/Project/ECUAL/LED/led.h"`
  - This **may cause errors** if you **changed** the **project directory** or **took a copy** on **another PC**.
  - A **solution** is to **modify all relative paths**, which is **very exhausting** when you deal with hundreds of files.

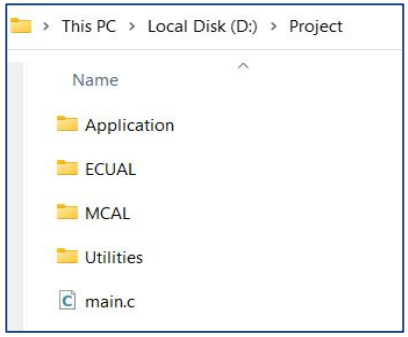


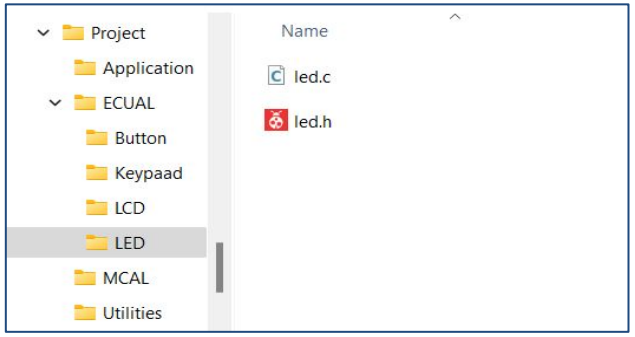


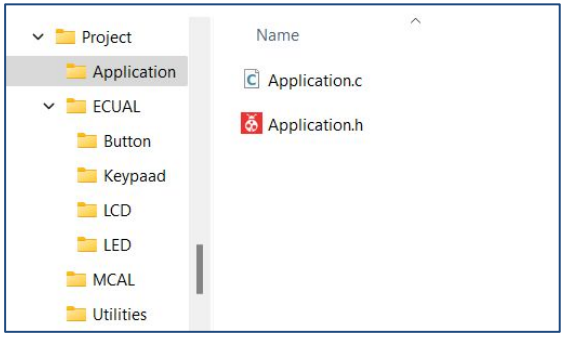


Includes

- **Examples**:
  - **Relative path include**
  - `#include "ECUAL/LED/led.h" // When including led.h in main.c`
  - `#include "../ECUAL/LED/led.h" // When including led.h in Application.c`







Object-like macros

- These macros define text for constants.
- Example:
  - `#define PI 3.14`
  - `#define RADIUS 5`
  - `circleArea = PI * RADIUS * RADIUS; // replaced with circleArea = 3.14 * 5 * 5;`

Function-like macros

- These macros define text for small functions.
- Example:
  - `#define PI 3.14`
  - `#define RADIUS 5`
  - `#define AREA(R) PI * R * R // Function-like macro`
  - `circleArea = AREA(5); // replaced with circleArea = 3.14 * 5 * 5;`
- Function-like macros may cause some wrong results:
  - `#define SQUARE(X) X * X`
  - `x = SQUARE(2+3); // replaced with x = 2+3*2+3;` **x will be 11 not 25**
  - **Solution:** `#define SQUARE(X) (X) * (X)` **- > replaced with** `x = (2+3)*(2+3);`

Conditional directives

- A conditional in the C preprocessor begins with a conditional directive.
- There are three types of conditional directives:
  - `#if`
  - `#ifdef`
  - `#ifndef`
- You may use the following to complete your conditions:
  - `#elif`
  - `#else`
- You must use `#endif` to end the conditional statement.

Conditional directives

- **`#if`** and **`#elif`** must take an expression.
- This expression like any C expression.
- The expression may contain:
  - **Constants** (**integer, character**)
  - **Operations** (**arithmetic, logical, and bitwise**)
  - **Macros**
  - The **defined** operator
- **Identifiers that are not macros**, **all** considered to be the **number zero**.

```
/* main.c */
#include <stdio.h>
#define R 5
int main()
{
      #if R>5
            printf("R > 5");
      #elif defined x
            printf("x is not defined");
      #elif y>10
            printf("y > 10");
      #else
            printf("All are false");
      #endif
}
```

Header file guard

- This guard is used to **prevent including** the **same header file more than once within** the **same file**.
- Including the same header file twice within the same file **will cause** an **error** when the compiler sees the same **structure definition twice**.

```
/* Header_file.h */
#ifndef HEADER_FILE
#define HEADER_FILE

the entire file

#endif /* End of Header_file.h */
```

Header file guard

```c
/* main.c */
#include "led.h"
#include "led.h"

int main()
{
	LED_on(LED_1);
}
```

Without guard

```c
/* led.h */

typedef struct led
{
	uint8_t ledPort;
	uint8_t ledState;
}ST_LED_t;
```

```c
/* main.c */
/* led.h */

typedef struct led
{
	uint8_t ledPort;
	uint8_t ledState;
}ST_LED_t;
/* led.h */
/* This generates
compiler error */
typedef struct led
{
	uint8_t ledPort;
	uint8_t ledState;
}ST_LED_t;

int main()
{
	LED_on(LED_1);
}
```

With guard

```c
/* led.h */
#ifndef LED_H
#define LED_H

typedef struct led
{
	uint8_t ledPort;
	uint8_t ledState;
}ST_LED_t;

#endif
/* End of led.h */
```

```c
/* main.c */
/* led.h */
#ifndef LED_H
#define LED_H

typedef struct led
{
	uint8_t ledPort;
	uint8_t ledState;
}ST_LED_t;

#endif
/* End of led.h */
/* led.h */
#ifndef LED_H
#define LED_H
/* This will be deleted after
preprocessing is completed */
typedef struct led
{
	uint8_t ledPort;
	uint8_t ledState;
}ST_LED_t;

#endif
/* End of led.h */

int main()
{
	LED_on(LED_1);
}
```

Summary

- Now you have good understanding about the C preprocessor.
- It’s clear now how the preprocessor works.
- Remember that absolute path includes may produce errors.
- Now you are familiar with macros, define, use, and types.
- Remember, put function-like macros arguments between (and) to avoid faulty results.
- Remember to use conditional directives to control what code to compile.
- Finally remember to use the header file guard to avoid duplicated structures errors.