Yrkes Akademin

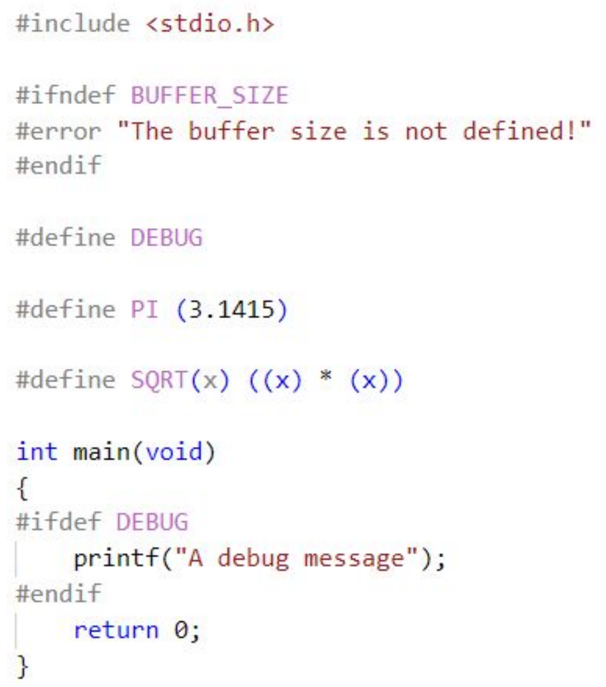
Vi hjälper dig att lyckas!

C Programming

Preprocessing Directives

Preprocessing Directives

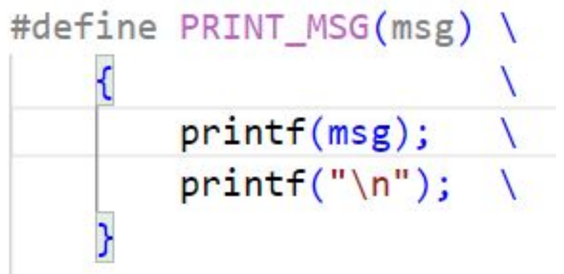
- ❖ Preprocessors are used by the compiler to prepare the actual code for compilation.
- ❖ Using preprocessors we tell the compiler how the program shall be compiled.
- ❖ Preprocessing is done in the first step of build executable programs
- ❖ Preprocessor directives are used to do different things
  - ➢ Inserting the content of header files
  - ➢ Defining and using macros
  - ➢ Defining and using generic macros
  - ➢ Conditional compilation
  - ➢ Generating error messages
- ❖ Preprocessor directives begin with **#** and end with the first **newline** character



```
#include <stdio.h>

#ifndef BUFFER_SIZE
#error "The buffer size is not defined!"
#endif

#define DEBUG

#define PI (3.1415)

#define SQRT(x) ((x) * (x))

int main(void)
{
#ifdef DEBUG
    printf("A debug message");
#endif
    return 0;
}
```

Preprocessing Directives

- To have multiline directives
  - End the line with a backslash (\) and continue the directive on the next line.

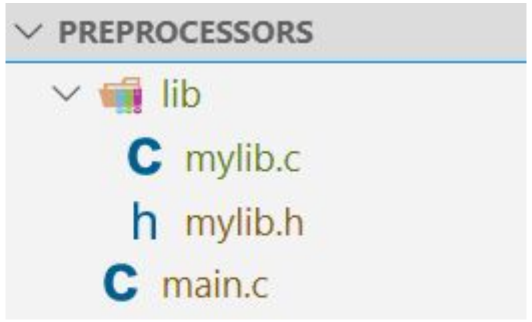


```
#define PRINT_MSG(msg) \
    {                  \
        printf(msg);   \
        printf("\n");  \
    }
```

- **#include** directive is used to insert content of header files. E.g. #include <stdio.h>
  - There are two ways to specify filenames
    - Using angle brackets (#include <filename>). For example; #include <stdio.h>
      - Use this form when you include the standard libraries
    - Using double quotation marks (#include "filename"). For example #include “mylib.h”
      - Use this form when you include non-standard libraries
  - The compiler shall be able to resolve the path of included files
    - It automatically resolves the path of standard libraries
    - For non-standard libraries, you need to help the compiler to resolve paths

Preprocessing Directives

- Inclusion of non-standard libraries
  - Relative paths can used. E.g. **#include “./lib/mylib.h”**
  - During compilation of the code we can specify the paths to the included libraries using **-I**
    - E.g. using **#include “mylib.h”** in the code and using **gcc main.c -I./lib -o main** to compile the code



- The **#ifdef** and **#ifndef** directives are used to check if a macro has been defined or not
- Prevent multiple inclusions of a library
  - We can use a macro (e.g. **MY_LIB**) and check if it has not been defined, we define it using **#define**. **#ifndef** shall be ended with **#endif**. In this way the block between #ifdef and #endif will be included in the actual code only once

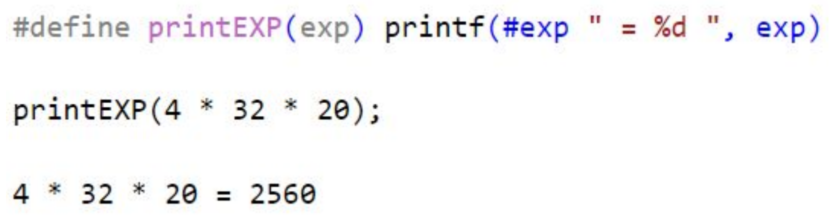
```c
// Include guards
#ifndef MY_LIB
#define MY_LIB

int add_ints(int a, int b);
double add_doubles(double a, double b);

#endif /* MY_LIB */
```

Preprocessing Directives - Macros

- A macro is a fragment of code which has been given a name.
- A macro can be defined using the preprocessor directive **#define**.
- Macros can be defined with or without parameters
  - **#define macro_name replacement_text**. E.g. #define PI 3.1415
  - **#define macro_name([parameter_list]) replacement_text**. E.g. #define ADD(x,y) ((x)+(y))
  - **#define macro_name([parameter_list ,] ...) replacement_text**.
    - Macros with variable number of parameters using spread (…) operator.
      - **...** means one or more parameters. __VA_ARGS__ identifier represents the arguments
      - E.g. #define PRINT(fmt, ...) printf(fmt, __VA_ARGS__)
- #define allows us to give a name to any text like constants or statements.
- A common use of macros is to define names for constants.

Preprocessing Directives - Macros

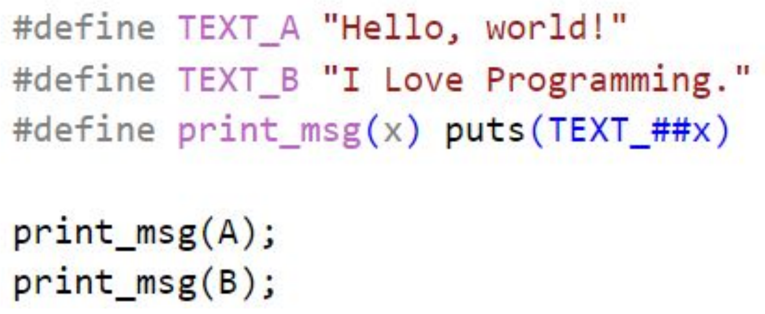
- ❖ Macros are we very useful to improve the readability of code.
  - ➢ Avoid magic numbers and literals in your code
- ❖ Macros are replaced during preprocessing by text replacement
- ❖ A function-like macro can be called like a function. E.g. int value = ADD(2, 3);
- ❖ In function-like macros we should enclose the parameters by parentheses
- ❖ In C there are some predefined macros like **__LINE__**, **__func__**, **__DATE__** and etc.
- ❖ It is possible to use a macro in another macro.
  - ➢ E.g. **#define PI 3.1415** and **#define AREA(r) (PI * (r) *(r))**
- ❖ Using the stringify operator (**#**) in macros
  - ➢ It converts a macro argument into a string.



```
#define printEXP(exp) printf(#exp " = %d ", exp)

printEXP(4 * 32 * 20);

4 * 32 * 20 = 2560
```

Preprocessing Directives - Macros

- The concatenation operator (**##**) joins its left and right operands into a single token
  - The outputs are
    - **Hello World!** and **I Love programming.**

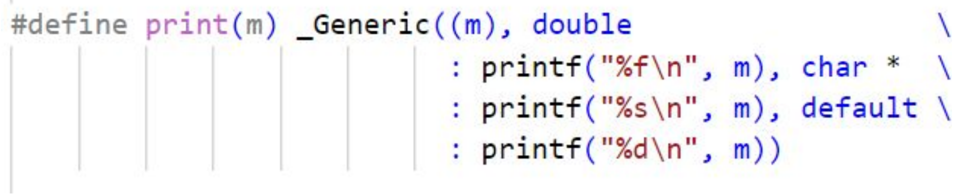


```
#define TEXT_A "Hello, world!"
#define TEXT_B "I Love Programming."
#define print_msg(x) puts(TEXT_##x)

print_msg(A);
print_msg(B);
```

- Macros can be defined during compilation
  - E.g. **gcc main.c -DDEVELOPMENT -o main** or **gcc main.c -DBUFSIZE=10 -o main**
- To redefine a macro first you need to undefine it and define it again using #define.
  - To undefine a macro you can use **#undef** preprocessor
    - E.g. **#define A 10**, **#undef A** and **#define A 20**
  - Undefine a macro which is not defined in your code during compilation using -U
    - E.g. **gcc main.c -U__LINE__ -o main**

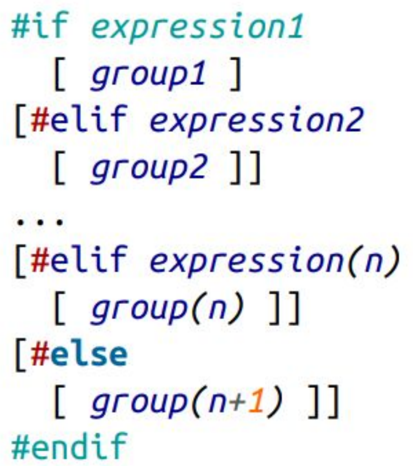
Preprocessing Directives

- Define type-generic macros using **_Generic**

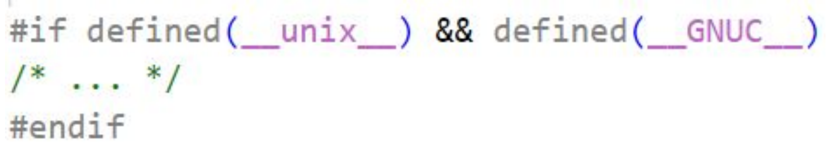


```
#define print(m) _Generic((m), double                \
                         : printf("%f\n", m), char *  \
                         : printf("%s\n", m), default \
                         : printf("%d\n", m))
```

- Conditional Compiling
  - To compile a code conditionally the following preprocessor directives can be used
  - **#if**, any number of **#elif**, **#else** and **#endif**
    - To comment a block of code out we shall use **#if 0** and **#endif**
  - **#ifdef** and **#endif**. #ifdef identifier is equivalent to **#if defined identifier**
  - **#ifndef** and **#endif.** #ifndef identifier is equivalent to **#if !defined identifier**
  - **defined** Operator can be used in the conditions of #if and #elif
  - **#error [message]** can be used to generate an error message
    - In the case of an error, the message is reported and the compilation is stopped.



```
#if expression1
  [ group1 ]
[#elif expression2
  [ group2 ]]
...
[#elif expression(n)
  [ group(n) ]]
[#else
  [ group(n+1) ]]
#endif
```



```
#if defined(__unix__) && defined(__GNUC__)
/* ... */
#endif
```

Yrkes Akademin
Vi hjälper dig att lyckas!