IS1201: Programming & Problem Solving

6. Macros & Preprocesser

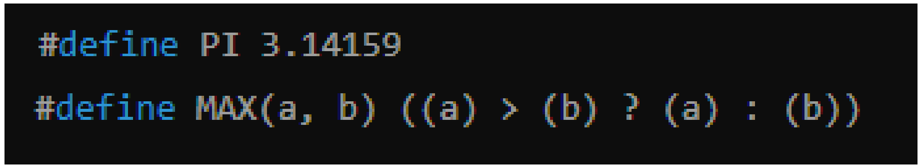
Viraj Welgama

Macro in C Programming

- In C programming, a macro is a fragment of code that is given a name.
- Whenever the name is used, it is replaced by the contents of the macro.
- Macros are defined using the #define directive and are processed by the preprocessor before the actual compilation of the code begins.
- They are often used to define constants, create inline functions, and simplify repetitive code.

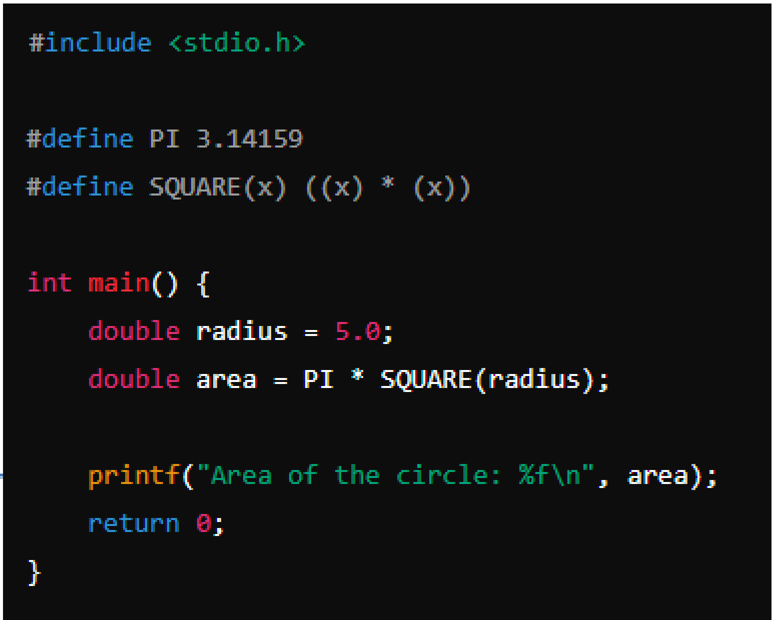
Defining a Macro

- A macro is defined using the #define directive, followed by the name of the macro and the code it should expand to.



```c
#define PI 3.14159
#define MAX(a, b) ((a) > (b) ? (a) : (b))
```

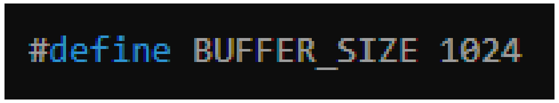
- Once defined, you can use the macro name in your code, and the preprocessor will replace it with the macro's content.



```c
#include <stdio.h>

#define PI 3.14159
#define SQUARE(x) ((x) * (x))

int main() {
    double radius = 5.0;
    double area = PI * SQUARE(radius);

    printf("Area of the circle: %f\n", area);
    return 0;
}
```

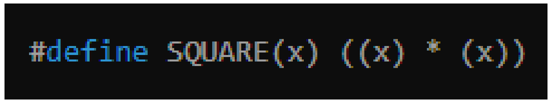
Types of Macros

- **Object-like Macros:**
  - These are simple replacements.



```
#define BUFFER_SIZE 1024
```

- **Function-like Macros**:
  - These take arguments and look like function calls.



```
#define SQUARE(x) ((x) * (x))
```

Macro: Example

```
# define PI 3.14
# define curcleArea(r) (PI*r*r)
```

- Every time the program encounters circleArea(argument),

  it is replaced by (3.14*(argument)*(argument)).

- Suppose, we passed 5 as an argument during the program, then,
  - CurcleArea(5) expands to (3.14 * 5 * 5)

Macro: Example

- A typical exercise in an algebra book asks you to evaluate an expression like

  n/3+2,

  for n=2, n=5 and n=9

- We can formulate such an expression as a program and use the program as many times as necessary.

Macro: Example

- A typical exercise in an algebra book asks you to evaluate an expression like

  n/3+2,

  for n=2, n=5 and n=9

- We can formulate such an expression as a program and use the program as many times as necessary.

```
#define f(n) (n/3.0 + 2)
```

Macro: Example

- A typical exercise in an algebra book asks you to evaluate an expression like

  n/3+2, for n=2, n=5 and n=9

- We can formulate such an expression as a program and use the program as many times as necessary.

```c
#define f(n) (n/3.0 + 2)
```

```c
int main() {
    for (int i=1; i<=10; i++) {
        printf("when n = %d, %.2f\n", i, f(i));
    }
}
```

Exercise:

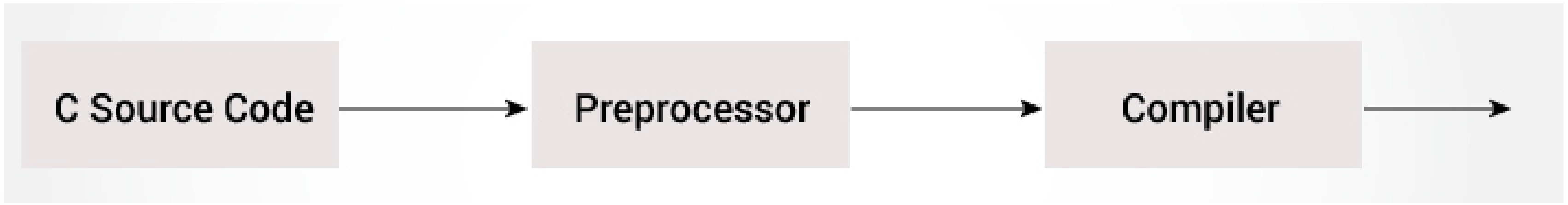
- formulate the following six expressions as programs and determine their results for n=2, n=5 and n=9.

1. $n^2 + 10$
2. $(n^2 + 40)/2$
3. $2 – (1/n)$
4. $n^2 + (1/2) * n^2 + 30$
5. $(2 – (1/n))*(n^2 + 10)$
6. $(1/2) * n^2 + 20 + (2 – (1/n))*(n^2 + 10)$

C Preprocessor

- The C Preprocessor is not part of the complier, but is a separate step in the complication process.
- C Preprocessor is just a text substitution tool and it instructs the complier to do required pre-processing before actual complication.
- It handles directives for macro substitution, file inclusion, conditional compilation, and other preprocessing tasks.
- All preprocessor commands begin with # symbol.

```
# define PI 3.14
```

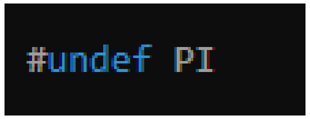


How the Preprocessor Works

1. Source Code Input:
   - The preprocessor takes the original source code as input.
2. Macro Expansion:
   - It expands all the macros defined by #define directives.
3. File Inclusion:
   - It replaces #include directives with the content of the included files.
4. Conditional Compilation:
   - It evaluates conditional directives and includes/excludes parts of the code based on conditions.
5. Output:
   - The preprocessed source code is then passed to the compiler for actual compilation.

Macro Substitution

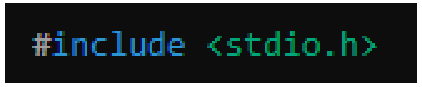
- When the preprocessor encounters a macro in the code, it replaces it with its defined value or code.
- You can undefine a macro using the #undef directive



```
#undef PI
```

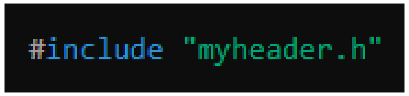
File Inclusion

- Include Directives:
  - The preprocessor can include the contents of other files into your source code using the #include directive.
  - **Standard Library Files:** These are included using angle brackets:



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

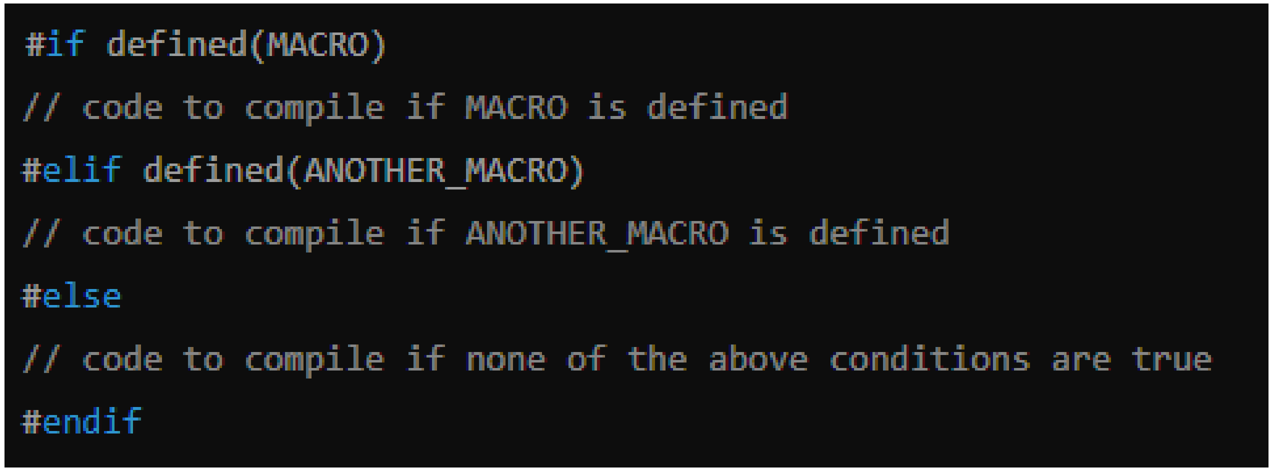
  - **User-defined Files:** These are included using double quotes



```
#include "myheader.h"
```

Conditional Compilation

- Conditional Directives:
  - These directives allow parts of the code to be included or excluded based on certain conditions. The common directives are:
  - #if, #elif, #else, #endif: Used to compile code conditionally

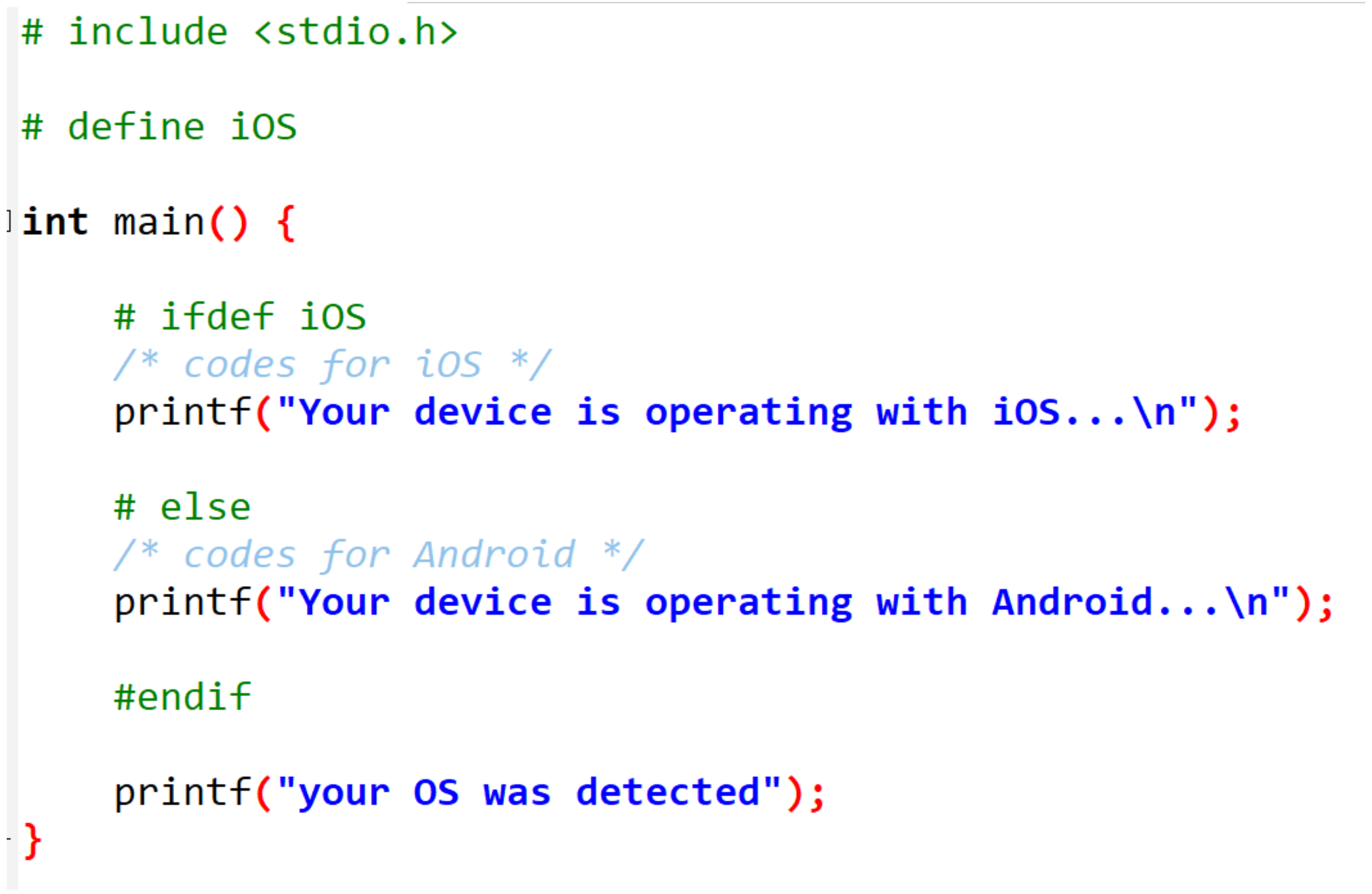


```
#if defined(MACRO)
// code to compile if MACRO is defined
#elif defined(ANOTHER_MACRO)
// code to compile if ANOTHER_MACRO is defined
#else
// code to compile if none of the above conditions are true
#endif
```

Conditional Compilation

- Uses of Conditional compilation can be;
    1. use different code depending on the machine, operating system
    2. compile same source file in two different programs
    3. to exclude certain code from the program but to keep it as reference for future purpose
- To use conditional compilations, #define, #defined, #ifdef, ifndef, #if, #elif, #else and #endif directives are used.
- The special operator #defined is used to test whether certain macro is defined or not.
    - It's often used with #if directive.

Example 1: Conditional Compilation



```
# include <stdio.h>

# define iOS

int main() {

    # ifdef iOS
    /* codes for iOS */
    printf("Your device is operating with iOS...\n");

    # else
    /* codes for Android */
    printf("Your device is operating with Android...\n");

    #endif

    printf("your OS was detected");
}
```

List of Important Preprocessors

Directive	Description
# define	Substitutes a preprocessor macro
# include	Inserts a particular header from another file
# undef	Undefines a preprocessor macro
# ifdef	Returns true if this macro is defined
# ifndef	Returns true if this macro is not defined
# if	Tests if a compile time condition is true
# else	The alternative for #if
# elif	#else and #if in one statement
# endif	Ends preprocessor conditional
# error	Prints error message on stderr
# pragma	Issues special commands to the compiler, using standardized method

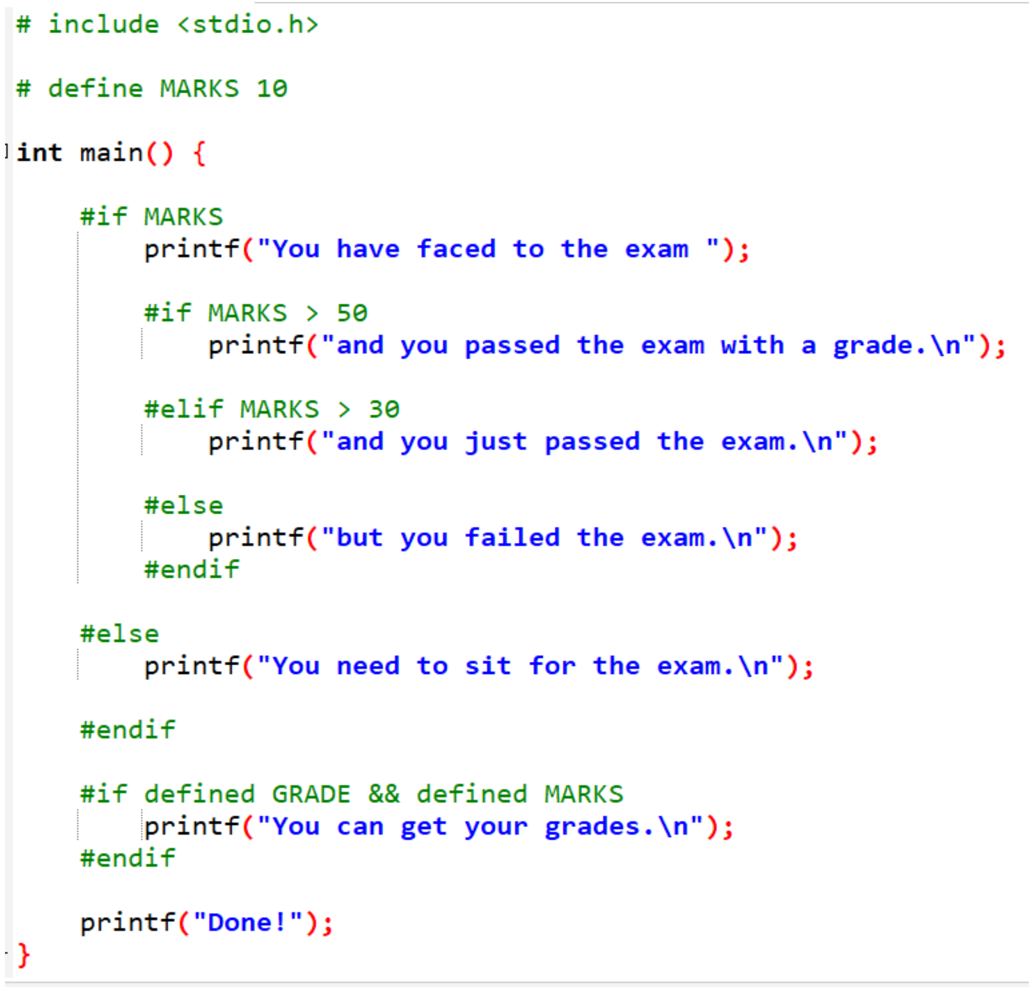
Preprocessor Operators

- The C preprocessor offers the following operators to help in creating macros:
  - Macro Continuation (\)
    - A macro usually must be contained on a single line. The macro continuation operator is used to continue a macro that is tool long for a single line.
  - Stringize (#)
    - The Stringize or number-sign operator ('#'), when used within a macro definition, converts a macro parameter into a string constant.
    - This operator may be used only in a macro that has a specified argument or parameter list.
  - Token Passing (##)
    - The token-pasting operator (##) within a macro definition combines two arguments.
    - It permits two separate tokens in the macro definition to be joined into a single token

Preprocessor Operators

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

# ifndef MESSAGE
# define MESSAGE "\nhi dear... good luck!\n"
# endif

# define message_for(x, y) \
printf (#x " " #y ": love you!\n")

# define tokenpaster(n) printf("token" #n " = %d", token##n)

int main() {

    int token24 = 40;

    printf(MESSAGE);
    message_for(tom, jerry);
    tokenpaster(24);
}
```

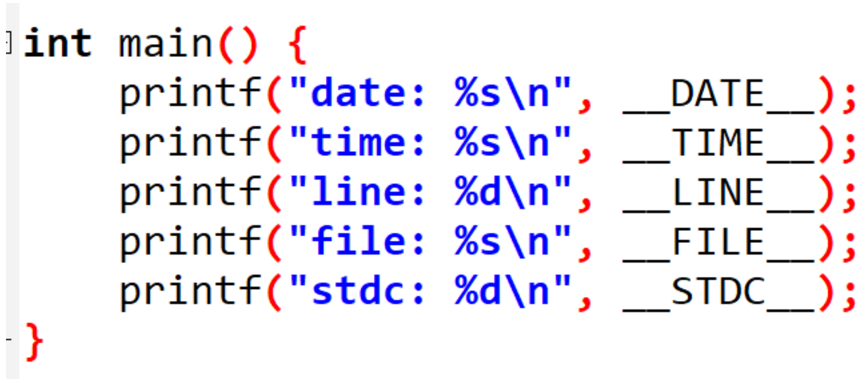
Example 2: Conditional Compilation



```
# include <stdio.h>

# define MARKS 10

int main() {

    #if MARKS
        printf("You have faced to the exam ");

        #if MARKS > 50
            printf("and you passed the exam with a grade.\n");

        #elif MARKS > 30
            printf("and you just passed the exam.\n");

        #else
            printf("but you failed the exam.\n");
        #endif

    #else
        printf("You need to sit for the exam.\n");

    #endif

    #if defined GRADE && defined MARKS
        printf("You can get your grades.\n");
    #endif

    printf("Done!");
}
```

Predefined Macros

Predefined Macro	Value
__DATE__	Staring containing the current date
__TIME__	String containing the current time.
LINE	Integer representing the current line number
__FILE__	String containing the file name.
__STDC__	If follows ANSI standard C, then value is a nonzero integer



```
int main() {
    printf("date: %s\n", __DATE__);
    printf("time: %s\n", __TIME__);
    printf("line: %d\n", __LINE__);
    printf("file: %s\n", __FILE__);
    printf("stdc: %d\n", __STDC__);
}
```