Preprocessing Directives and Conditional Compilation

Vivek kumar
Sharukh Ansari
Vibhuti Roy
Gatadi Ashwitha

University of Hyderabad

September 29, 2021

Introduction

- Lines preceded by hash(#) at the top of the program(C/C++ programming languages)
- Preprocessed by the compiler before actual compilation begins
- End of lines are identified with newline character rather than semicolon(;)
- Used in defining macros, evaluating conditional statements, source file inclusion, pragma directive, line control, error detection etc.

#include

- Includes the contents of a file during the compilation
- Makes easy to handle collections of #define and declarations
- Two forms:
  - #include<filename>
  - #include“filename”

#define

- Syntax: **#define token replacement−text**
- These macros can be defined with or without arguments
- Drawback with order of evaluation

#define macros without arguments

- It is processed like a symbolic constant
- Ex:
```
#define X 10
int main(){
        printf("X*X = %d",X*X);
        return 0;
}
```

Output: X*X = 100

#define macros with arguments

- The arguments are substituted in the replacement text, then the macro is expanded
- Ex: #define SQUARE(X) (X)*(X)

```
    int main(){
            int a = 4;
            printf("Square of X = %d", SQUARE(a));
            return 0;
    }
```

Output: Square of X = 16

Drawback with order of evaluation

- Ex: #define SQUARE(X) X*X

```
int main(){
        int a = 4;
        printf("Square of X = %d", SQUARE(a+1));
        return 0;
}
```

  Output: Square of X = 9
- Evaluation: SQUARE(a+1) = a+1*a+1 = 4+1*4+1 = 9

#undef

- Used to undefine the names
- Syntax: #undef "undefines"
- Undefined names can be redefined with #define
- Ex:

```
#define X 4
    int main(){
            printf("X = %d", X);
            #undef X
            #define X 6
            printf("X = %d", X);
            return 0;
    }
```

Output: $X = 4$

$X = 6$

Conditional Compilation

- Includes the code selectively, depending on the value of conditions evaluated during compilation
- #if, #endif, #elif, #else, #ifdef, #ifndef are the conditional preprocessing constructs
- One of the main use of it is to avoid multiple declarations

Example:
```
#if !defined(VAR)
#define VAR
    struct point{
        int x;
        int y;
    };
#endif
```

Conditional compilation in debugging aid

- Helps in knowing whether the value is as expected
- Ex:#ifdef DEBUG

```
        printf("x is %d \n",x);
  #endif
```

- In above example, the printf statement is executed if DEBUG is defined
- Insert line '#define DEBUG' to enable debugging printouts

#error

- Used to raise an error during compilation and terminate the process.
- Syntax: #error <error message>
- Example:

```
#ifndef macro
    #error macro is not defined
#endif
```

#line

- Sets the value of line number and filename to a given line number and filename
- Syntax: #line digit−sequence ["filename"]
- Line number is incremented after each line is processed

```
Line 1:    int main(){
Line 2:        printf("Hello world\n");
Line 3:        printf("Line: %d\n",__LINE__);
Line 4:        //reset the line number by 36
Line 5:        #line 36
Line 37:       printf("Line: %d \n",__LINE__);
Line 38:       return 0;
Line 39:   }
```

Stringize(#) operator

- Using the operator '#' the tokens in the replacement text converted to a string surrounded by quotes.
- Ex:#include<stdio.h>

```
    #define STR_PRINT(x) #x
    int main(){
            printf(STR_PRINT(This is a string without double quotes));
            return 0;
            }
```

Output: This is a string without double quotes

Token Pasting(##) operator

- '##' operator used to concatenate two tokens
- Ex:#include<stdio.h>

```
    #define STR_CONCAT(x, y) x##y
    int main(){
            printf("%d", STR_CONCAT(20, 50));
            return 0;
    }
```

Output: 2050

Thank You