# Chapter - 10 The C++ Pre-processor

## The Pre-processor

The C++ Pre-processor is nothing more than a glorified text editor.

It has its own syntax and knows nothing about C++ syntax.

## ***#define*** statement

```
#define SIZE 20
```

Tells the C++ pre-processor “global change word ‘SIZE’ to 20”.

Note: The **#define** statement was widely used in the old C language (which didn't have a **const** declaration).
In C++ most **#define** statements can and should be replaced by **const** declarations.

General form of the #define statement:

```
#define Name Substitute-Text
```

## ***#define*** misuse

Anything can be used as the substitute text. For example:

```
    #define FOR_ALL for(i = 0; i < ARRAY_SIZE; ++i)
```

Sample use:

```
/*
 * Clear the array
 */
FOR_ALL {
    data[i] = 0;
}
```

This changes the syntax of C++ and will confuse any programmer who doesn't know what `FOR_ALL` is. (And programmers hate to have to look up such things.)

## ***#define*** super misuse

```
#define BEGIN {
#define END }


. . .
    if (index == 0)
    BEGIN
        std::cout << "Starting\n";
    END
```

This isn’t C++. It’s PASCAL (sort of).

Excerpt from an early version of a program called the Bourne Shell (a UNIX utility).

```
IF (x GREATER_THAN 37) OR (Y LESS_THAN 83) THEN
    CASE value OF
        SELECT 1:
            start();
        SELECT 3:
            backspace();
        OTHERWISE:
            error();
    ESAC
FI
```

**Yuck!!**

# Pre-processor surprises

```
 1 #define BIG_NUMBER 10 ** 10
 2
 3 main()
 4 {
 5     // index for our calculations
 6     int   index;
 7
 8     index = 0;
 9
10     // syntax error on next line
11     while (index < BIG_NUMBER) {
12         index = index * 8;
13     }
14     return (0);
15 }
```

Syntax error on line 11.

Note: That’s no where near the line that caused the error.

# Question:

**The following program generates the answer 47 instead of the expected answer 144. Why? (Hint below.)**

```
#include <iostream>

#define FIRST_PART      7
#define LAST_PART       5
#define ALL_PARTS       FIRST_PART + LAST_PART

main() {
    std::cout << "The square of all the parts is " <<
        ALL_PARTS * ALL_PARTS << '\n';
    return (0);
}
```

To see the output of the Pre-processor on UNIX execute the command:

```
    CC -E prog.cpp
```

On MS-DOS/Windows, use:

```
    cpp prog.cpp
```

# Question:

This program generates a warning that counter is used before it is set. This is a surprise to us because the for loop should set it. We also get a very strange warning, null effect”, for line 11.

```
 1 // warning, spacing is VERY important
 2
 3 #include <iostream>
 4
 5 #define MAX=10
 6
 7 main()
 8 {
 9     int  counter;
10
11     for (counter =MAX; counter > 0;
12       --counter)
13         std::cout << "Hi there\n";
14
15     return (0);
16 }
```

# Question:

**The following program is supposed to print the message "Fatal `Error: Abort`" and exit when it receives bad data. But when it gets good data, it exits. Why?**

```
#include <iostream>
#include <stdlib.h>

#define DIE \
  std::cerr << "Fatal Error:Abort\n";exit(8);

main() {
    // a random value for testing
    int value;

    value = 1;
    if (value < 0)
        DIE;

    cerr << "We did not die\n";
    return (0);
}
```

# ***#define***vs. *const*

***Const***

- Relatively new (before const, #define was the only way to go)
- Part of the C++ syntax
- Follows C++ scope rules
- Compiler detects errors where they occur

***#define***

- Used mostly by older programs
- Can be used to define almost anything (including statements)
- Pre-processor style syntax
- Errors may be detected far from where they occur

You should use **const** whenever possible instead of **#define**.

## Conditional Compilation

Example:

```
#ifdef DEBUG
        std::cout <<
             "In compute_hash " <<
             " value=" << value <<
             " hash=" << hash << "\n",
  #endif /* DEBUG */
```

The code is turned on by putting:

```
#define DEBUG
```

in your program or by putting:

```
-DDEBUG
```

in as part of the compilation line.

# Conditional Compilation Style

Put any statements that control conditional compilation at the top of your code where they're easy to find.

If you use:

```
    #define DEBUG        /* Turn debugging on */
```

to turn on debugging, then use

```
    #undef DEBUG         /* Turn debugging off */
```

to turn it off. (Strictly speaking the #**undef** is not needed, however it does serve to notify someone that changing it to a #**define** will do something.)

## ***#ifndef*** and ***#else***

#**ifndef** compiles the code if the symbol is not defined.
#**else** reverses the sense of the conditional.

```
#ifdef DEBUG
    std::cout << "Test version. Debugging is on\n";
#else /* DEBUG */
    std::cout <<"Production version\n";
#endif /* DEBUG */
```

# Commenting out code

A programmer wanted to get rid of some code temporarily so he commented it out:

```
/***** Comment out this section
    section_report();
    /* Handle the end of section stuff */
    dump_table();
**** end of commented out section */
```

This generates a syntax error for the fifth line. (Why?)
A better method is to use the **#ifdef** construct to remove the code.

```
#ifdef UNDEF
    section_report();
    /* Handle the end of section stuff */
    dump_table();
#endif /* UNDEF */
```

Note: Any programmer defining the symbol `UNDEF` will be shot.

## Include Files

The directive:

```
   #include <iostream>
```

tells the pre-processor: "go to the directory containing the standard include files and copy the file *iostream* in here."

The directive:

```
   #include "defs.h"
```

tells the pre-processor: "Copy the file in from my local directory."

## Protection against double includes

```
#ifndef _CONST_H_INCLUDED_

/* define constants */

#define _CONST_H_INCLUDED_
#endif  /* _CONST_H_INCLUDED_ */
```

## Parameterized Macros

Example:

```
#define SQR(x)  ((x) * (x))     /* Square a number */
    SQR(5) expands to  ((5) * (5))
```

Example of how *not* to use:

```
#include <iostream>
#define SQR(x) (x * x)


main()
{
    int counter;    // counter for loop


    for (counter = 0; counter < 5; ++counter) {
        std::cout << "x " << counter+1 <<
                " x squared " << SQR(counter+1) << '\n';
    }
    return (0);
}
```

## Why ++ and -- should always be on their own line:

```
#include <iostream>
#define SQR(x) ((x) * (x))

int main()
{
    int counter;    /* counter for loop */

    counter = 0;
    while (counter < 5)
        std::cout << "x " << counter+1 <<
                " x squared " <<
                SQR(++counter) << '\n';
    return (0);
}
```

## Question

The following program tells us that we have an undefined variable `number`, but our only variable name is `counter`. Why?

```
#include <iostream>
#define RECIPROCAL (number) (1.0 / (number))

int main()
{
    float   counter;

    for (counter = 0.0; counter < 10.0;
         counter += 1.0) {

        std::cout << "1/" << counter << " = " <<
                  RECIPROCAL(counter) << "\n";
    }
    return (0);
}
```

# The # operator

The # operator turns a parameter into a string. For example:

```
#define STR(data) #data
STR(hello)
```

generates

```
"hello"
```

## Parameterized macros vs. *inline* functions

*Parameterized Macros*

- Are part of the older C style pre-processor syntax
- Can easily get you into trouble with hidden side effects
- The SQR macro we defined works on both **float** and **int**.

*inline functions*

- Are part of the C++ syntax
- Much better error detection
- Do not do mere text replacement. We could not define a SQR **inline** function that would work on both **float** and **int**.

**inline** functions are must less risky than parameterized macros and should be used whenever possible.

# Rule for pre-processor use

1. In particular you should enclose **#define** constants and macro parameters.

```
#define area (20*10) // Correct
#define size 10+22      // Wrong
#define DOUBLE(x)  (x * 2) // Wrong
#define DOUBLE(x)  ((x) * 2) // Right
```

2. When defining a macro with more than one statement, enclose the code in {}.

3. The pre-processor is not C++. Don't use = or ;.

```
#define X = 5 // Illegal
#define X 5;  // Illegal
#define X = 5;// Very Illegal
```

```
#define X 5   // Correct
```