# C Preprocessor Directives

The C preprocessor is a micro processor that is used by compiler to transform your code before compilation. It is called micro preprocessor because it allows us to add macros.

#### *Note: Proprocessor direcives are executed before compilation.*

C preprocessor

All preprocessor directives starts with hash # symbol.

List of preprocessor directives.

* #include
* #define
* #undef
* #ifdef
* #ifndef
* #if
* #else
* #elif
* #endif
* #error
* #pragma

# C Macros

A macro is a segment of code which is replaced by the value of macro. Macro is defined by #define directive. There are two types of macros:

1. Object-like Macros
2. Function-like Macros

## **Object-like Macros**

The object-like macro is an identifier that is replaced by value. It is widely used to represent numeric constants. For example:

1. #define PI 3.14

Here, PI is the macro name which will be replaced by the value 3.14.

## **Function-like Macros**

The function-like macro looks like function call. For example:

1. #define MIN(a,b) ((a)<(b)?(a):(b))

Here, MIN is the macro name.

Visit [#define](http://www.javatpoint.com/c-preprocessor-define) to see the full example of object-like and function-like macros.

## **C Predefined Macros**

ANSI C defines many predefined macros that can be used in c program.

|  |  |  |
| --- | --- | --- |
| **No.** | **Macro** | **Description** |
| 1 | \_DATE\_ | represents current date in "MMM DD YYYY" format. |
| 2 | \_TIME\_ | represents current time in "HH:MM:SS" format. |
| 3 | \_FILE\_ | represents current file name. |
| 4 | \_LINE\_ | represents current line number. |
| 5 | \_STDC\_ | It is defined as 1 when compiler complies with the ANSI standard. |

## **C predefined macros example**

*File: simple.c*

#include <stdio.h>

main() {

   printf("File :%s\n", \_\_FILE\_\_ );

   printf("Date :%s\n", \_\_DATE\_\_ );

   printf("Time :%s\n", \_\_TIME\_\_ );

   printf("Line :%d\n", \_\_LINE\_\_ );

   printf("STDC :%d\n", \_\_STDC\_\_ );

}

Output:

File :simple.c

Date :Dec 6 2015

Time :12:28:46

Line :6

STDC :1

# C #include

The #include preprocessor directive is used to paste code of given file into current file. It is used include system-defined and user-defined header files. If included file is not found, compiler renders error.

By the use of #include directive, we provide information to the preprocessor where to look for the header files. There are two variants to use #include directive.

1. #include <filename>
2. #include "filename"

The **#include <filename>** tells the compiler to look for the directory where system header files are held. In UNIX, it is \usr\include directory.

The **#include "filename"** tells the compiler to look in the current directory from where program is running.

## **#include directive example**

Let's see a simple example of #include directive. In this program, we are including stdio.h file because printf() function is defined in this file.

1. #include <stdio.h>
2. main() {
3. printf("Hello C");
4. }

Output:

Hello C

## **#include notes:**

**Note 1:** In #include directive, comments are not recognized. So in case of #include <a//b>, a//b is treated as filename.

**Note 2:** In #include directive, backslash is considered as normal text not escape sequence. So in case of #include <a\nb>, a\nb is treated as filename.

**Note 3:** You can use only comment after filename otherwise it will give error.

# C #define

The #define preprocessor directive is used to define constant or micro substitution. It can use any basic data type.

Syntax:

1. #define token value

Let's see an example of #define to define a constant.

1. #include <stdio.h>
2. #define PI 3.14
3. main() {
4. printf("%f",PI);
5. }

Output:

3.140000

Let's see an example of #define to create a macro.

1. #include <stdio.h>
2. #define MIN(a,b) ((a)<(b)?(a):(b))
3. **void** main() {
4. printf("Minimum between 10 and 20 is: %d\n", MIN(10,20));
5. }

Output:

Minimum between 10 and 20 is: 10

# C #undef

The #undef preprocessor directive is used to undefine the constant or macro defined by #define.

Syntax:

1. #undef token

Let's see a simple example to define and undefine a constant.

1. #include <stdio.h>
2. #define PI 3.14
3. #undef PI
4. main() {
5. printf("%f",PI);
6. }

Output:

Compile Time Error: 'PI' undeclared

The #undef directive is used to define the preprocessor constant to a limited scope so that you can declare constant again.

Let's see an example where we are defining and undefining number variable. But before being undefined, it was used by square variable.

1. #include <stdio.h>
2. #define number 15
3. **int** square=number\*number;
4. #undef number
5. main() {
6. printf("%d",square);
7. }

Output:

225

# C #ifdef

The #ifdef preprocessor directive checks if macro is defined by #define. If yes, it executes the code otherwise #else code is executed, if present.

Syntax:

1. #ifdef MACRO
2. //code
3. #endif

Syntax with #else:

1. #ifdef MACRO
2. //successful code
3. #else
4. //else code
5. #endif

## **C #ifdef example**

Let's see a simple example to use #ifdef preprocessor directive.

1. #include <stdio.h>
2. #include <conio.h>
3. #define NOINPUT
4. **void** main() {
5. **int** a=0;
6. #ifdef NOINPUT
7. a=2;
8. #else
9. printf("Enter a:");
10. scanf("%d", &a);
11. #endif
12. printf("Value of a: %d\n", a);
13. getch();
14. }

Output:

Value of a: 2

But, if you don't define NOINPUT, it will ask user to enter a number.

1. #include <stdio.h>
2. #include <conio.h>
3. **void** main() {
4. **int** a=0;
5. #ifdef NOINPUT
6. a=2;
7. #else
8. printf("Enter a:");
9. scanf("%d", &a);
10. #endif
12. printf("Value of a: %d\n", a);
13. getch();
14. }

Output:

Enter a:5

Value of a: 5

# C #ifndef

The #ifndef preprocessor directive checks if macro is not defined by #define. If yes, it executes the code otherwise #else code is executed, if present.

Syntax:

1. #ifndef MACRO
2. //code
3. #endif

Syntax with #else:

1. #ifndef MACRO
2. //successful code
3. #else
4. //else code
5. #endif

## **C #ifndef example**

Let's see a simple example to use #ifndef preprocessor directive.

1. #include <stdio.h>
2. #include <conio.h>
3. #define INPUT
4. **void** main() {
5. **int** a=0;
6. #ifndef INPUT
7. a=2;
8. #else
9. printf("Enter a:");
10. scanf("%d", &a);
11. #endif
12. printf("Value of a: %d\n", a);
13. getch();
14. }

Output:

Enter a:5

Value of a: 5

But, if you don't define INPUT, it will execute the code of #ifndef.

1. #include <stdio.h>
2. #include <conio.h>
3. **void** main() {
4. **int** a=0;
5. #ifndef INPUT
6. a=2;
7. #else
8. printf("Enter a:");
9. scanf("%d", &a);
10. #endif
11. printf("Value of a: %d\n", a);
12. getch();
13. }

Output:

Value of a: 2

# C #if

The #if preprocessor directive evaluates the expression or condition. If condition is true, it executes the code otherwise #elseif or #else or #endif code is executed.

Syntax:

1. #if expression
2. //code
3. #endif

Syntax with #else:

1. #if expression
2. //if code
3. #else
4. //else code
5. #endif

Syntax with #elif and #else:

1. #if expression
2. //if code
3. #elif expression
4. //elif code
5. #else
6. //else code
7. #endif

## **C #if example**

Let's see a simple example to use #if preprocessor directive.

1. #include <stdio.h>
2. #include <conio.h>
3. #define NUMBER 0
4. **void** main() {
5. #if NUMBER==0
6. printf("Value of Number is: %d",NUMBER);
7. #endif
8. getch();
9. }

Output:

Value of Number is: 0

# C #else

The #else preprocessor directive evaluates the expression or condition if condition of #if is false. It can be used with #if, #elif, #ifdef and #ifndef directives.

Syntax:

1. #if expression
2. //if code
3. #else
4. //else code
5. #endif

Syntax with #elif:

1. #if expression
2. //if code
3. #elif expression
4. //elif code
5. #else
6. //else code
7. #endif

## **C #else example**

Let's see a simple example to use #else preprocessor directive.

1. #include <stdio.h>
2. #include <conio.h>
3. #define NUMBER 1
4. **void** main() {
5. #if NUMBER==0
6. printf("Value of Number is: %d",NUMBER);
7. #else
8. print("Value of Number is non-zero");
9. #endif
10. getch();
11. }

Output:

Value of Number is non-zero

# C #error

The #error preprocessor directive indicates error. The compiler gives fatal error if #error directive is found and skips further compilation process.

## **C #error example**

Let's see a simple example to use #error preprocessor directive.

1. #include<stdio.h>
2. #ifndef \_\_MATH\_H
3. #error First include then compile
4. #else
5. **void** main(){
6. **float** a;
7. a=sqrt(7);
8. printf("%f",a);
9. }
10. #endif

Output:

Compile Time Error: First include then compile

But, if you include math.h, it does not gives error.

1. #include<stdio.h>
2. #include<math.h>
3. #ifndef \_\_MATH\_H
4. #error First include then compile
5. #else
6. **void** main(){
7. **float** a;
8. a=sqrt(7);
9. printf("%f",a);
10. }
11. #endif

Output:

2.645751

# C #pragma

The #pragma preprocessor directive is used to provide additional information to the compiler. The #pragma directive is used by the compiler to offer machine or operating-system feature.

Syntax:

1. #pragma token

Different compilers can provide different usage of #pragma directive.

The turbo C++ compiler supports following #pragma directives.

1. #pragma argsused
2. #pragma exit
3. #pragma hdrfile
4. #pragma hdrstop
5. #pragma inline
6. #pragma option
7. #pragma saveregs
8. #pragma startup
9. #pragma warn

Let's see a simple example to use #pragma preprocessor directive.

1. #include<stdio.h>
2. #include<conio.h>
4. **void** func() ;
6. #pragma startup func
7. #pragma exit func
8. **void** main(){
9. printf("\nI am in main");
10. getch();
11. }
13. **void** func(){
14. printf("\nI am in func");
15. getch();
16. }

Output:

I am in func

I am in main

I am in func