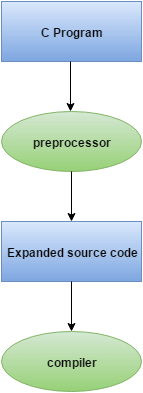
C Preprocessor Directives

* 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.
* All preprocessor directives starts with hash # symbol. 
* Let's see a list of preprocessor directives.
* #include
* #define
* #undef
* #ifdef
* #ifndef
* #if
* #else
* #elif
* #endif
* #error
* #pragma

# Macro Substitution Directives in C

In C, a **macro** is a piece of code in a program that is replaced by the value of the macro. Macro is defined by **#define** directive. Whenever a macro name is encountered by the compiler, it replaces the name with the definition of the macro. Macro definitions need not be terminated by a semi-colon(**;**).

example

// C program to illustrate macros

#include <stdio.h>

// Macro definition

#define LIMIT 5

// Driver Code

int main()

{

// Print the value of macro defined

printf("The value of LIMIT"" is %d",LIMIT);

return 0;

}

Output: The value of LIMIT is 5

The below example demonstrates the use of macros to find the area of a rectangle.

// C program to illustrate macros

#include <stdio.h>

// Macro definition

#define AREA(l, b) (l \* b)

// Driver Code

int main()

{

// Given lengths l1 and l2

int l1 = 10, l2 = 5, area;

// Find the area using macros

area = AREA(l1, l2);

// Print the area

printf("Area of rectangle"" is: %d",area);

return 0;

}

**Output**

Area of rectangle is: 50

# Nested Macros in C

# A macro may be used in the definition of another macro

#include<stdio.h>

#define SQUARE(X) (X+X)

#define CUBE(X) SQUARE(X)\*X

void main()

{

int a=10;

printf("\nResult of square is %d", SQUARE(a));

printf("\nResult of cube is %d",CUBE(a));

return 0;

}

**Output**

Result of square is 20

Result of cube is 200

# Undefining a macro C #undef

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

1. Syntax: #undef token

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

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

Parameterized Macro:

define a macro that uses arguments, you insert parameters between the pair of parentheses in the macro definition that make the macro function-like. The parameters must be valid C identifiers, separated by commas and optionally whitespace.

#include <stdio.h>

// Function-like Macro definition

#define min(a, b) (((a) < (b)) ? (a) : (b))

// Driver Code

int main()

{

    // Given two number a and b

    int a = 18;

    int b = 76;

    printf("Minimum value between"

           " %d and %d is %d\n",

           a, b, min(a, b));

    return 0;

}

**Output**

Minimum value between 18 and 76 is 18

# Difference between Macro and Function

|  |  |
| --- | --- |
| **MACRO** | **FUNCTION** |
| Macro is Preprocessed | Function is Compiled |
| No Type Checking is done in Macro | Type Checking is Done in Function |
| Using Macro increases the code length | Using Function keeps the code length unaffected |
| Use of macro can lead to side effects at later stages | Functions do not lead to any side effects in any case |
| Speed of Execution using Macro is Faster | Speed of Execution using Function is Slower |
| Before Compilation, the macro name is replaced by macro value | During function call, transfer of control takes place |
| Macros are useful when small code is repeated many times | Functions are useful when large code is to be written |
| Macro does not check any Compile-Time Errors | Function checks Compile-Time Errors |

# #pragma Directive in C

In C, the #pragma directive is a special purpose directive that is used to turn on or off some features. #pragma also allows us to provide some additional information or instructions to the compiler. It is compiler-specific i.e., the behavior of pragma directive varies from compiler to compiler.

## **Syntax of #pragma**

#pragmadirective\_name

## Commonly Used #pragma Directives in C

Some of the commonly used #pragma directives are discussed below:

### **1. #pragma startup and #pragma exit**

These directives help us to specify the functions that are needed to run before the program starts ( before the control passes to the main()) and just before the program exits (just before the control returns from the main()).

**Syntax**

#pragma startup function\_name

#pragma exit function\_name

#include <stdio.h>

void func1();

void func2();

#pragma startup func1

#pragma exit func2

void func1() { printf("Inside func1()\n"); }

void func2() { printf("Inside func2()\n"); }

int main()

{

    printf("Inside main()\n");

    return 0;

}

**Output:**

Inside func1()

Inside main()

Inside func2()

**3. Error directive ( #error )**: This directive aborts the compilation process when it is found in the program during compilation and produces an error which is optional and can be specified as a parameter.

**Syntax:**

#error error\_message

Here, **optional\_error** is any error specified by the user which will be shown when this directive is found in the program. Example:

#define BUFFER\_SIZE 255

#if BUFFER\_SIZE < 256

#error "BUFFER\_SIZE is too small."

#endif

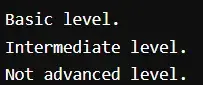
**Output:**

BUFFER\_SIZE is too small.

##### **#if Conditional Compilation Directive Example in C**

* The #if directive in C is used for conditional compilation.
* The compiler evaluates the condition; if the condition is true (non-zero), the code following the #if up to the next #endif, #else, or #elif is compiled. Otherwise, it is skipped.
* This is particularly useful for including code only in certain compile-time conditions, like debugging or targeting specific platforms. Here’s an example to demonstrate the usage of #if, #else, and #endif:
* #include <stdio.h>
* #define LEVEL 2
* int main() {
* #if LEVEL > 0
* printf("Basic level.\n");
* #endif
* #if LEVEL > 1
* printf("Intermediate level.\n");
* #endif
* #if LEVEL > 2
* printf("Advanced level.\n");
* #else
* printf("Not advanced level.\n");
* #endif
* return 0;
* }

Output :



##### **#ifdef Conditional Compilation Directive Example in C**

* The #ifdef directive in C checks if a certain macro is defined before compiling a portion of code.
* This is particularly useful for making your code adaptable to different environments or conditions without changing the source code. Here’s an example to illustrate how #ifdef is used:
* C program **to demonstrate #ifdef, #ifndef , #else and #endif** –
* # include <stdio.h>
* # define a 10
* void main(){
* #ifdef a
* printf("  
   Hello I am here..");
* #endif
* #ifndef a
* printf("  
   Not defined ");
* #else
* printf("  
   R u There ");
* #endif
* }

**Output :**

Hello I am here..

R u There

## **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*

1. #include<stdio.h>
2. **int** main(){
3. printf("File :%s\n", \_\_FILE\_\_ );
4. printf("Date :%s\n", \_\_DATE\_\_ );
5. printf("Time :%s\n", \_\_TIME\_\_ );
6. printf("Line :%d\n", \_\_LINE\_\_ );
7. printf("STDC :%d\n", \_\_STDC\_\_ );
8. **return** 0;
9. }

**Output:**

**File :simple.c**

**Date :Dec 6 2015**

**Time :12:28:46**

**Line :6**

**STDC :1**