C Math

C Programming allows us to perform mathematical operations through the functions defined in <math.h> header file. The <math.h> header file contains various methods for performing mathematical operations such as sqrt(), pow(), ceil(), floor() etc.

C Math Functions

There are various methods in math.h header file. The commonly used functions of math.h header file are given below.

No. Function Description

1) ceil(number) rounds up the given number. It returns the integer value which is greater than or equal to the given number.

2) floor(number) rounds down the given number. It returns the integer value which is less than or equal to the given number.

3) sqrt(number) returns the square root of the given number.

4) pow(base, exponent) returns the power of the given number.

5) abs(number) returns the absolute value of given number.

C Math Example

Let's see a simple example of math functions found in math.h header file.

1. #include<stdio.h>

2. #include<conio.h>

3. #include<math.h>

4. void main(){

5. clrscr();

6.

7. printf("\n%f",ceil(3.6));

8. printf("\n%f",ceil(3.3));

9. printf("\n%f",floor(3.6));

10. printf("\n%f",floor(3.2));

11. printf("\n%f",sqrt(16));

12. printf("\n%f",sqrt(7));

13. printf("\n%f",pow(2,4));

14. printf("\n%f",pow(3,3));

15. printf("\n%d",abs(-12));

16. getch();

17. }

Output:

4.000000

4.000000

3.000000

3.000000

4.000000

2.645751

16.000000

27.000000

12

File Handling in C

File Handling in c language is used to open, read, write, search or close file. It is used for permanent storage.

Advantage of File

It will contain the data even after program exit. Normally we use variable or array to store data, but data is lost after program exit. Variables and arrays are non-permanent storage medium whereas the file is a permanent storage medium.

Functions for file handling

There are many functions in C library to open, read, write, search and close file. A list of file functions are given below:

No. Function Description

1 fopen() opens new or existing file

2 fprintf() write data into file

3 fscanf() reads data from file

4 fputc() writes a character into file

5 fgetc() reads a character from file

6 fclose() closes the file

7 fseek() sets the file pointer to given position

8 fputw() writes an integer to file

9 fgetw() reads an integer from file

10 ftell() returns current position

11 rewind() sets the file pointer to the beginning of the file

Opening File: fopen()

The fopen() function is used to open a file. The syntax of fopen() function is given below:

1. FILE \*fopen( const char \* filename, const char \* mode );

You can use one of the following modes in the fopen() function.

Mode Description

r opens a text file in read mode

w opens a text file in write mode

a opens a text file in append mode

r+ opens a text file in read and write mode

w+ opens a text file in read and write mode

a+ opens a text file in read and write mode

rb opens a binary file in read mode

wb opens a binary file in write mode

ab opens a binary file in append mode

rb+ opens a binary file in read and write mode

wb+ opens a binary file in read and write mode

ab+ opens a binary file in read and write mode

Closing File: fclose()

The fclose() function is used to close a file. The syntax of fclose() function is given below:

1. int fclose( FILE \*fp );

C Preprocessor Directives

The C preprocessor is a microprocessor that is used by the 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.

All preprocessor directives start with the hash # symbol.

Let's see a list of preprocessor directives.

o #include

o #define

o #undef

o #ifdef

o #ifndef

o #if

o #else

o #elif

o #endif

o #error

o #pragma

C Macros

A macro is a segment of code which is replaced by the value of the macro. Macro is defined by the #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 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 the current date in "MMM DD YYYY" format.

2 \_TIME\_ represents current time in "HH:MM: SS" format.

3 \_FILE\_ represents the current file name.

4 \_LINE\_ represents the 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. 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. }

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, the 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 the only comment after filename otherwise it will give an 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

11.

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

Let's see another example to understand the #if directive clearly.

1. #include <stdio.h>

2. #include <conio.h>

3. #define NUMBER 1

4. void main() {

5. clrscr();

6. #if (NUMBER==0)

7. printf("1 Value of Number is: %d",NUMBER);

8. #endif

9.

10. #if (NUMBER==1)

11. printf("2 Value of Number is: %d",NUMBER);

12. #endif

13. getch();

14. }

Output:

2 Value of Number is: 1

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

#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>

3.

4. void func() ;

5.

6. #pragma startup func

7. #pragma exit func

8.

9. void main(){

10. printf("\nI am in main");

11. getch();

12. }

13.

14. void func(){

15. printf("\nI am in func");

16. getch();

17. }

Output:

I am in func

I am in main

I am in func