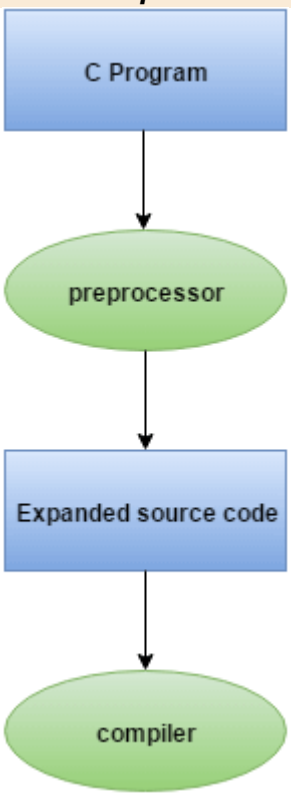
C Preprocessor Directives

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.



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

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.

For example:-

```
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));
}
```

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
```

C #include

The #include pre-processor 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.  int main(){
3.    printf("Hello C");
4.    return 0;
5.  }
```

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 pre-processor directive is used to define constant or micro substitution. It can use any basic data type.

Syntax:

```
#define token value
```

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

```
#include <stdio.h>
#define PI 3.14
main() {
   printf("%f",PI);
}
```

Output:

```
3.140000
```

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

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

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:

```
#undef token
```

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

```
#include <stdio.h>
#define PI 3.14
#undef PI
main() {
   printf("%f",PI);
}
```

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.

```
#include <stdio.h>
#define number 15
int square=number*number;
#undef number
main() {
   printf("%d",square);
}
```

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:

```
#ifdef MACRO
//code
#endif
```

Syntax with #else:

```
#ifdef MACRO
//successful code
#else
//else code
#endif
```

C #ifdef example

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

```
#include <stdio.h>
#include <conio.h>
#define NOINPUT
void main() {
int a=0;
#ifdef NOINPUT
a=2;
#else
printf("Enter a:");
scanf("%d", &a);
#endif
printf("Value of a: %d\n", a);
getch();
}
```

Output:

```
Value of a: 2
```

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

```
#include <stdio.h>
#include <conio.h>
void main() {
int a=0;
#ifdef NOINPUT
a=2;
```

```
#else
printf("Enter a:");
scanf("%d", &a);
#endif

printf("Value of a: %d\n", a);
getch();
}
```

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:

```
#ifndef MACRO
//code
#endif
```

Syntax with #else:

```
#ifndef MACRO
//successful code
#else
//else code
#endif
```

C #ifndef example

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

```
#include <stdio.h>
#include <conio.h>
#define INPUT
void main() {
int a=0;
#ifndef INPUT
a=2;
#else
printf("Enter a:");
scanf("%d", &a);
#endif
printf("Value of a: %d\n", a);
getch();
}
```

Output:

```
Enter a:5
Value of a: 5
```

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

```
#include <stdio.h>
#include <conio.h>
void main() {
int a=0;
#ifndef INPUT
a=2;
#else
printf("Enter a:");
scanf("%d", &a);
#endif
printf("Value of a: %d\n", a);
getch();
}
```

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:

```
#if expression
//code
#endif
```

Syntax with #else:

```
#if expression
//if code
#else
//else code
#endif
```

Syntax with #elif and #else:

```
#if expression
//if code
#elif expression
//elif code
#else
//else code
#endif
```

C #if example

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

```
#include <stdio.h>
#include <conio.h>
#define NUMBER 0
void main() {
#if (NUMBER==0)
printf("Value of Number is: %d",NUMBER);
#endif
getch();
}
```

Output:

```
Value of Number is: 0
```

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

```
#include <stdio.h>
#include <conio.h>
#define NUMBER 1
void main() {
clrscr();
#if (NUMBER==0)
printf("1 Value of Number is: %d",NUMBER);
#endif

#if (NUMBER==1)
printf("2 Value of Number is: %d",NUMBER);
#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:

```
#if expression
//if code
#else
//else code
#endif
```

Syntax with #elif:

```
#if expression
//if code
#elif expression
//elif code
#else
//else code
#endif
```

C #else example

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

```
#include <stdio.h>
#include <conio.h>
#define NUMBER 1
void main() {
#if NUMBER==0
printf("Value of Number is: %d",NUMBER);
#else
print("Value of Number is non-zero");
#endif
getch();
}
```

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.

```
#include<stdio.h>
#ifndef __MATH_H
#error First include then compile
#else
void main(){
    float a;
    a=sqrt(7);
    printf("%f",a);
}
#endif
```

Output:

```
Compile Time Error: First include then compile
```

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

```
#include<stdio.h>
#include<math.h>
#ifndef __MATH_H
#error First include then compile
#else
void main(){
    float a;
    a=sqrt(7);
    printf("%f",a);
}
#endif
```

Output:

```
2.645751
```