

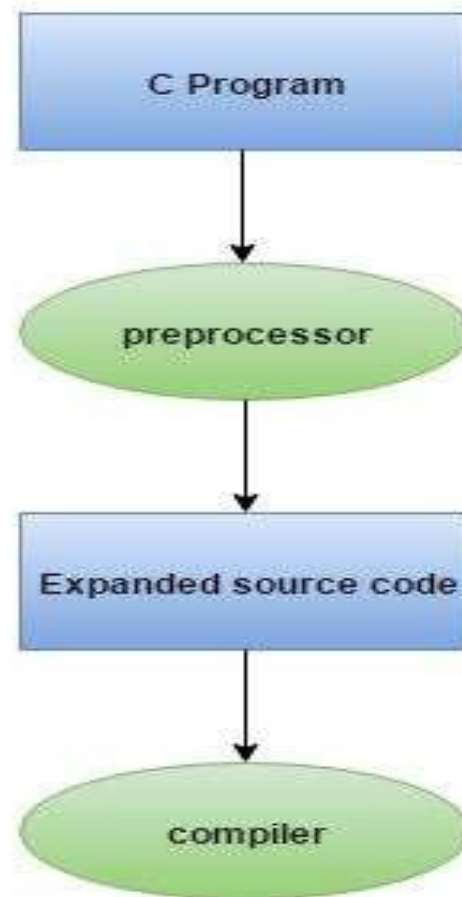


Preprocessor Directives

iC

Definition

- ❑ A preprocessor is a program that processes our source program before it is passed to the compiler.
- ❑ The preprocessor works on the source code and creates “expanded source code”.
- ❑ The preprocessor offers several features called **preprocessor directives**.
- ❑ Each of these preprocessor directives begin with a # symbol.
- ❑ The directives can be placed anywhere in a program but most often placed at the beginning of the program, before the function definition.



Preprocessor directives are executed before compilation.



□ The various preprocessor directives are as follows:

1) Macro Expansion (#define)

II) File Inclusion(#include)

III) Conditional Compilation(#ifdef -#endif)

IV) Miscellaneous Directives

(#undef,#pragma-startup,exit,warn)

Macro Expansion

- A “macro” is a fragment of code which has been given a name.
- Wherever the name is used, it is replaced by the contents of the macro.
- There are two kinds of macros which are as follows:
 - I) **Object-like macros** resemble data objects
 - II) **Function-like macros** resemble function calls.

Object –like Macros

- The object-like macro is an identifier that is replaced by value.
- It is widely used to represent numeric constants.
- It is called object-like because it looks like a data object in code that uses it.
- **Syntax:**

```
#define identifier replacement-text
```

Examples

```
#define x 4
void main()
{
int a; a=x;
}
```

x=identifier 4=replacement-text

```
#define pi 3.14
3.14=substitute
void main()
{
float r=6.25,area;
area=pi*r*r;
printf("Area of circle=%f",area);
}
```

pi=identifier

Note:

- To create macros with the #define directive, the syntax for macros is:

```
#define    macro          macro  
          expansion    replacement list
```

- #define directive can be used as follows:
 - I) It can be used to define operators
like: #define AND &&
#define OR ||



II)it could be used even to replace a condition:

```
# define AND &&
```

```
#define ARRANGE (a>25 AND a<50)
```

III)It could be used to replace even an entire C statement like:

```
# define FOUND printf("Hello World")
```

Function-like Macro

- It looks like a function call & can be defined with the `# define` directive but with a pair of braces or parenthesis immediately after the macro name.
- **For Example:**

```
#define Area(r) 3.14* r*r
```

Macro with arguments

- Function –like macros can take arguments just like true functions.

```
#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
```




Difference between Macro & Function

- 1) In a macro call, the preprocessor replaces the macro template with its macro expansion only. whereas, in a functional call the control is passed to a function along with certain arguments, some calculations are performed in the function & a useful value is returned back from the function.

- 
- II) if we use a macro hundred times in a program, the macro expansion goes into our source code at hundred different places, thus increasing the program size. On the other hand, if a function is used, then even if it is called from hundred different places in the program, it would take the same amount of space in the program, thus making the program smaller and compact.

File Inclusion


- Used to include one file into another file or it simply causes the entire content of one file to be inserted into the source code of another file.
- We use file inclusions for the following reasons:
 - 1) if we have a very large program, the code is best divided into several different files, each containing a set of related functions. These files are then included at the beginning of main program file.



II) There are some commonly needed functions & macro definitions that can be stored in a file & that file can be included in every program we write which would add all the statements in this file to our program as if we have typed in.

- The two ways to write include statement:

`#include "filename"` `#include`
`<filename>`

- 
- `#include "goto.h"`: It searches for a file named "goto.h" in the directory containing the current file.
 - `#include <goto.h>`: it searches for a file named "goto.h" in a standard list of system directories.

Undefining a Macro

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

```
#undef token
```

- Example:

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

Output:

```
Compile Time Error: 'PI' undeclared
```

Preprocessor Operators

- The C preprocessor offers the following operators to help create macros –

I)The Macro Continuation (\) Operator

- A macro is normally confined to a single line. The macro continuation operator (\) is used to continue a macro that is too long for a single line.
- For example –

```
//multi line macro function
#define DISPLAYMATH(a, b) {\
    printf("%d\n", a+b);\
    printf("%d\n", a*b);\
}
```

II)The Stringize (#) Operator

- The stringize or number-sign operator ('#'), when used within a macro definition, converts a macro parameter into a string constant.

- This operator may be used only in a macro having a specified argument or parameter list.

- For example –
define say(m) printf(#m)
void main()
{
say(Hello);

}

Output:
Hello

Conditional Compilation

- These allow us to include certain portion of the code depending upon the output of constant expression.
- 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
```

Example:

```
#include<stdio.h>
#define NUM 10

void main()
{
    // Define another macro if MACRO NUM is defined

    #ifdef NUM
        #define MAX 20
    #endif

    printf("MAX number is : %d",MAX);
}
```

Output :

```
MAX Number is 20
```

#ifndef directive

- 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
```

Example:

```
#include "stdio.h"

void main()
{
    // Define another macro if MACRO NUM is defined

    #ifndef NUM
        #define MAX 20
    #endif

    printf("MAX number is : %d",MAX);
}
```

Output :

```
MAX Number is 20
```

error directive

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

Example:

```
#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
```