PREPROCESSOR DIRECTIVES

The C Preprocessor is not part of the compiler but it extends the power of C programming language. . The preprocessor provides the ability for the inclusion of header files, macro expansions, conditional compilation, and line control.The preprocessor's functionality comes before compilation of source code and it instruct the compiler to do required pre-processing before actual compilation. Working procedure of C program is shown in Fig. 2.8. In general, preprocessor directives

- begin with a # symbol
- do not end with semicolon
- are processed before compilation of source code

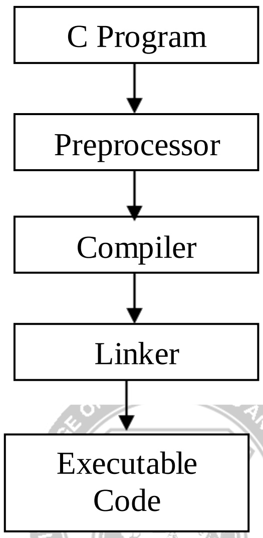

Fig. 2.8 Working Procedure of C Program

There are four types of Preprocessor Directives supported by C language. They are:

- File Inclusion directive
- Macro Substitution directive
- Conditional directive
- Miscellaneous directive

List of all possible directives belong to each of the above is listed in Fig 2.9.

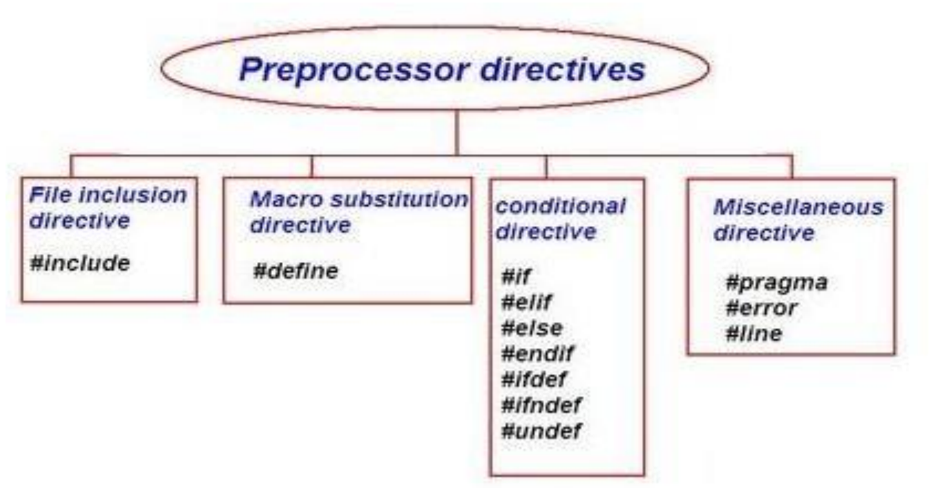

Fig Preprocessor Directives

The details of above listed preprocessor directives are narrated in Table.

Table Preprocessor directives and their description

Directive	Description
#include	It includes header file inside a C Program.
#define	It is substitution macro. It substitutes a constant with an expression.
#if	It includes a block of code depending upon the result of conditional expression.
#else	It is a complement of #if
#elif	#else and #if in one statement. It is similar to if else ladder.
#endif	It flags the end of conditional directives like #if, #elif etc.
#undef	Undefines a preprocessor macro.
#ifdef	Returns true If constant is defined earlier using #define.
#ifndef	Returns true If constant is not defined earlier using #define.
#pragma	Issues special commands to the compiler.
#error	Prints error message on stderr.

File Inclusion directive

#include

It is used to include header file inside C Program. It checks for header file in current directory, if path is not mentioned. To include user defined header file double quote is used ("") instead of using triangular bracket (< >).

Example:

```
#include <stdio.h>                // Standard Header File
#include "big.h"          // User Defined Header File
```

Preprocessor replaces #include <stdio.h> with the content of stdio.h header file. #include "Sample.h" instructs the preprocessor to get Sample.h from the current directory and add the content of Sample.h file.

Macro Substitution directive

#define

It is a simple substitution macro. It substitutes all occurrences of the constant and replace them with an expression.There are two types of macro supported by C. They are:

1. Simple macro
2. macro with arguments

Simple macro

Syntax:

```
#define identifier value
```

Where

#define	- is apreprocessor directive used for text substitution.
identifier	- is an identifier used in program which will be replaced by value.(In general the identifiers are represented in captital letters in order to differentiate them from variable)
value	-It is the value to be substituted for identifier.

Example:

```
#define PI 3.14
#define NULL 0
```

Example:

```
//Program to find the area of a circle using simple macro
#include <stdio.h>
#define PI 3.14
int main()
{
        int radius;
        float area;
        printf("Enter the radius of circle \n");
        scanf("%d", &radius);
        area= PI * radius * radius;
        printf("Area of Circle=%f", radius);
}
```

Output

```
Enter the radius of circle
10
Area of Circle = 314.000000
```

macro with arguments

#define Preprocessing directive can be used to write macro definitions with parameters. Whenever a macro identifier is encountered, the arguments are substituted by the actual arguments from the C program.

Data type definition is not necessary for macro arguments. Any numeric values like int, float etc can be passed as a macro argument . Specifically, argument macro is not case sensitive.

Example:

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

Example:

```
//Program to find the area of a circle using macro with arguments
#include <stdio.h>
#define area(r) (3.14*r*r)
int main()
{
        int radius;
        float a;
        printf("Enter the radius of circle \n");
        scanf("%d", &radius);
        a= area(radius);
        printf("Area of Circle=%f", a);
}
```

Output

```
Enter the radius of circle
10
Area of Circle = 314.000000
```

Predefined Macros in C Language

C Programming language defines a number of macros. Table 2.8 is the list of some commonly used macros in C

Table 2.8 Predefined macros in C

Macro	Description
NULL	Value of a null pointer constant.
EXIT_SUCCESS	Value for the exit function to return in case of successful completion of program.
EXIT_FAILURE	Value for the exit function to return in case of program termination due to failure.
RAND_MAX	Maximum value returned by the rand function.
__FILE__	Contains the current filename as a string.
__LINE__	Contains the current line number as a integer constant.
__DATE__	Contains current date in "MMM DD YYYY" format.
__TIME__	Contains current time in "HH:MM:SS" format.

Example:

// Program to print the values of Predefined macros

```
#include  <stdio.h>
#include <stdlib.h>
int main()
{
        printf("NULL : %d\n", NULL );
        printf("EXIT_SUCCESS : %d\n", EXIT_SUCCESS );
        printf("EXIT_FAILURE : %d\n", EXIT_FAILURE );
        printf("RAND_MAX : %d\n", RAND_MAX );
        printf("File Name : %s\n", __FILE_);
        printf("DATE : %s\n", __DATE__);
        printf("Line : %d\n", __LINE__);
        return 0;
}
```

Output

```
NULL : 0
EXIT_SUCCESS : 0
EXIT_FAILURE : 1
RAND_MAX : 32767
File Name : BuiltinMacro.c
DATE : Aug 16 2017
Line : 12
```

Conditional directive

#if, #elif, #else and #endif

The Conditional directives permit to include a block of code based on the result of conditional expression.

Syntax:

```
        #if  <expression>
                statements;
        #elif   <expression>
                statements;
        #else
                 statements;
        #endif
```

Where

Expression represents a condition which produces a boolean value as a result.

Conditional directive is similar to if else condition but it is executed before compilation. Condition_Expression must be only constant expression.

Example:

```
//Program to illustrate the conditional directives
#include <stdio.h>
#define A 10
int main()
{
        #if  (A>5)
                printf("A=%d", X);
        #elif  (A<5)
                printf("A=%d", 4);
        #else
                printf("A=%d", 0);
        #endif
        return 0;
}
```

Output

```
X=10
```

#undef

The #undef directive undefines a constant or preprocessor macro defined previously using #define.

Syntax:

```
        #undef <Constant>
```

Example:

```
#include<stdio.h>
#define P 100
#ifdef P
        #undef P
       #define P 30
#else
         #define P 100
#endif
int main()
{
        printf("%d",P);
       return 0;
}
```

Output

```
30
```

<u>#ifdef #ifdef, #ifndef</u>

#ifdef

#ifdef directive is used to check whether the identifier is currently defined. Identifiers can be defined by a #define directive or on the command line.

#ifndef

#ifndef directive is used to check whether the identifier is not currently defined.

Example:

```
#ifdef  PI
       printf( "Defined \n" );
#endif
#ifndef  PI
       printf( "First define PI\n" );
#endif
```

Output:

```
First define PI
```

<u>Miscellaneous directive</u>

The pragma directive is used to access compiler-specific preprocessor extensions. Each pragma directive has different implementation rule and use . There are many type of pragma directive and varies from one compiler to another compiler .Ifcompiler does not recognize particular pragma then it ignores the pragma statement without showing any error or warning message.

Example:

```
#pragma sample
int main()
{
      printf("Pragma verification ");
      return 0;
}
```

Output

```
 Pragma verification
```

Since #pragma sample is unknown for Turbo c compiler, it ignores sample directive without showing error or warning message and execute the whole program

assuming #pragma sample statement is not present. The following are the list of possible #pragma directives supported by C.

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

#error

The #error directive causes the preprocessor to emit an error message. #error directive is used to prevent compilation if a known condition that would cause the program not to function properly.

Syntax:

```
#error “message”
```

Example:

```
int main()
{
        #ifndef PI
          #error "Include PI”
        #endif
        return 0;
}
```

Output

```
compiler error --> Error directive : Include PI
```

#line

It tells the compiler that next line of source code is at the line number which has been specified by constant in #line directive

Syntax:

```
#line <line number> [File Name]
```

Where

File Name is optional

Example:

```
int main()
{
        #line 700
        printf(Line  Number  %d”,_LINE_);
        printf(Line  Number  %d”,_LINE_);
        printf(Line  Number  %d”,_LINE_);
        return 0;
}
```

Output

```
700
701
702
```