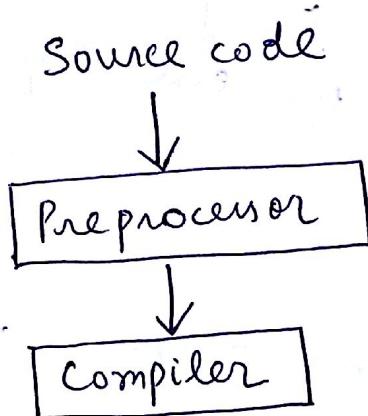


PREPROCESSOR DIRECTIVES

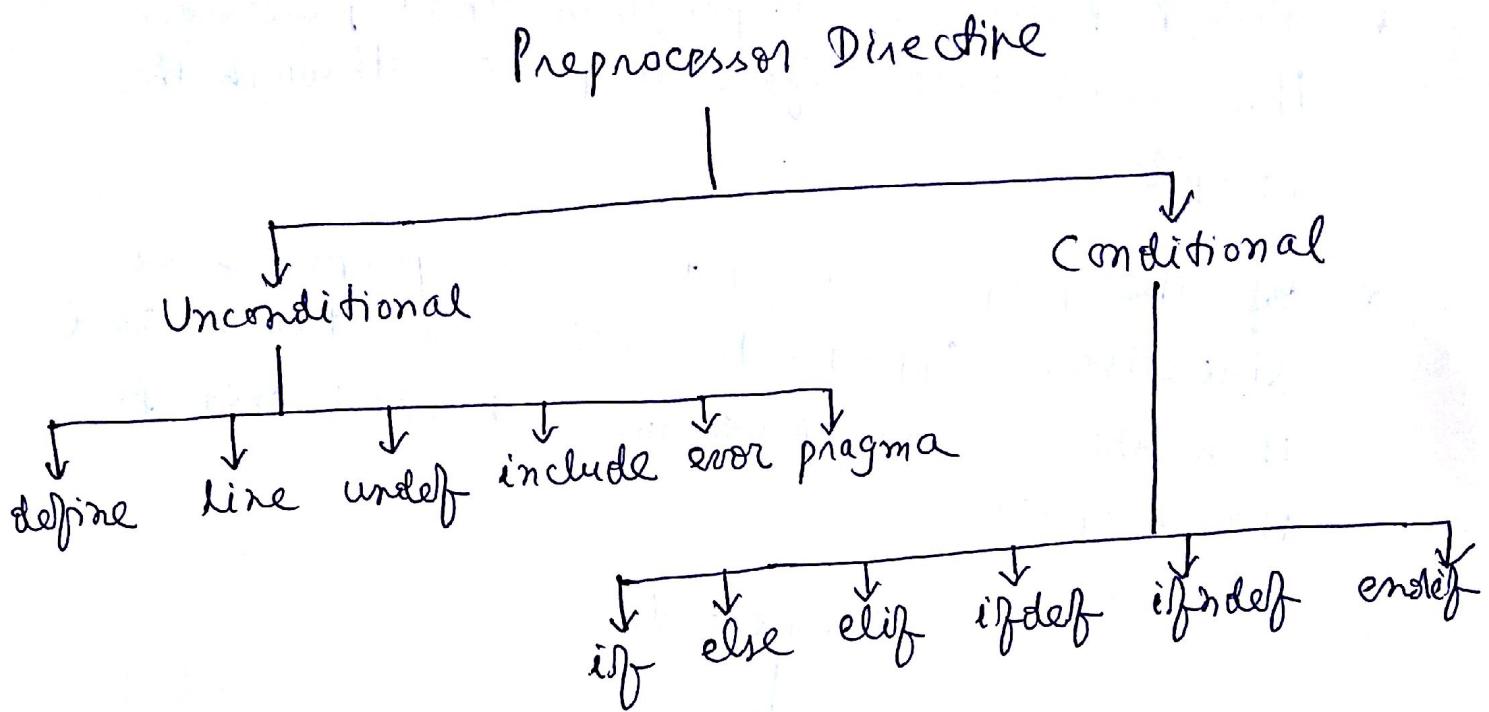
- * The preprocessor is a program that processes the source code before it passes through the compiler
- * If the preprocessor finds some preprocessor directives, appropriate actions are taken and then the source program is handed over to the compiler



Features of Preprocessor Directives

- 1) Each preprocessor directive starts with a # symbol.
- 2) There can be only one directive on a line
- 3) There is no semicolon at the end of a directive
- 4) They can be placed anywhere in a program but they are normally written at the beginning of a program
- 5) A preprocessor directive is active from the point of its appearance till the end of the program.

Types of Preprocessor Directives



Unconditional Directives

- * These directives perform well-defined tasks.

#define

- This is used for defining symbolic constants.
- General syntax:

`#define name constant value`

Eg: `#define TRUE 1`

`#define FALSE 0`

`#define PI 3.14159265`

`#define MAX 100`

line

This directive is used for ~~debugging~~ purposes. It assigns dec-const and string-const to the predefined macros -LINE- and -FILE- respectively.

Syntax:

```
#line dec-const string-const
```

Here dec-const : decimal constant

string-const : string constant

-LINE- : decimal constant that represents the line no. being compiled

-FILE- : string constant that represents the name of the file being compiled.

macro : name given to a block of C statements as a pre-processor directive. A macro is defined with the preprocessor directive.

error

- This preprocessor directive is used for debugging purpose.

- #error directive stops compilation and displays an error message attached with it.

Pragma Directives

- * `#pragma` directive is used to control the actions of the compiler in a particular portion of a program without affecting the program as a whole.
- * The effect of `pragma` is applied from the point where it is included till the end.
- * `#pragma` is an instruction to the compiler and is normally ignored during preprocessing.

Syntax :

`#pragma string`

Types of Pragma Directives

P-342 (Reema Thareja book)

undef

- The definition of a directive exist from the #define directive till the end of the program
- To undefine this directive, the #undef directive is used

Syntax:

`# undef directive-name`

include

- #include is used to include a file into the source code
- This directive is used to include header files in our programs.

Syntax:

`#include <filename>`

Conditional Directives

- * There may be situations when we want to compile some parts of the code based on some conditions.
- * Compilation of a part of code based on some condition
- * The conditions are checked during the preprocessing phase.
- * The conditional directives are:

#if and #endif

An expression which is followed by the `#if` is evaluated, if the result is non-zero then the statements between `#if` and `#endif` are compiled; otherwise they are skipped.

`#if condition`

—
—

`#endif`

#else and #elif

- It is analogous to if...else control statement.
- #else is used with the #if preprocessor directive.

#if condition1
statements1

#elif condition2
statements2

#else
statements3

#endif

- If the condition evaluates to non-zero, then the corresponding statements between #if and #else are compiled.
- Otherwise the statements between #else and #endif are compiled.

#ifdef and #ifndef

These directives provide a way of combining #if with #define directive.

Syntax of #ifdef

#ifdef symbolic-name

====

#endif

If the symbolic name has been defined with #define, then the statements between #ifdef and #endif will be compiled. Otherwise, the statements in between are not compiled.

Syntax of #ifndef

#ifndef symbolic-name

====

#endif

If the symbolic-name has not been defined with #defined or was undefined using #undef, then the statements between #ifndef and #endif are compiled. Otherwise, not compiled.

Command Line Arguments

- * C language provides a method to pass parameters to the main() function. This is accomplished by specifying arguments on the main() function.
- * The main() function can accept two arguments:

Full declaration of main():

```
int main ( int argc, char * argv[] )
```

where - the first argument (argc) specifies the number of command line arguments

- the complete list of all the command line arguments (*argv[]) is an array of character pointers
- Each ~~pointer~~ element of the array argv is a pointer pointing to a string

Normally, argv[0] is the name of the program or empty string if name is not available

- argv[1] points to the next parameter, and so on.

Example

```
#include <stdio.h>
int main ( int argc, char *argv[] )
{
    int i;
    printf (" No. of arguments passed = %d ",  
           argc);
    for ( i = 0; i < argc; i++ )
        printf ("%s\n", argv[i]);
    return 0;
}
```

DOS Shell: Command Prompt (Execute the program from DOS in command prompt)
O/P: C:\> tc test.c Reema Thareja ← by writing
argc = 3 argv[0] = test.c
 argv[1] = Reema
 argv[2] = Thareja

/* C program to add two nos. using command line arguments */

```
#include <stdio.h>
void main ( int argc, char *argv[] )
{
    int i, sum = 0;
    if ( argc != 3 )
    {
        printf (" You have forgot to type nos. ");
        exit(1);
    }
```

```
printf("Sum is: %d");
for (i = 1; i < argc; i++)
    sum = sum + atoi(argv[i]);
printf("%d", sum);
```

}

Steps to be followed to execute program using Command Line Arguments in C

Step 1: Write a Program

Step 2: Open command prompt inside C

Step 3: Click on file → DOS Shell

Step 4: Inside command prompt type this command :

C: TCBIN > add.c 10 20
The sum is : 30 Step 5: Hit enter, you will get the o/p!

C: TCBIN >

Step 6: Type exit command to return to Turbo C Screen.