SNS COLLEGE OF TECHNOLOGY

Coimbatore – 35

An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A+' Grade

Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

DEPARTMENT OF AI & ML

PROGRAMMING FOR PROBLEM SOLVING

I YEAR - I SEM

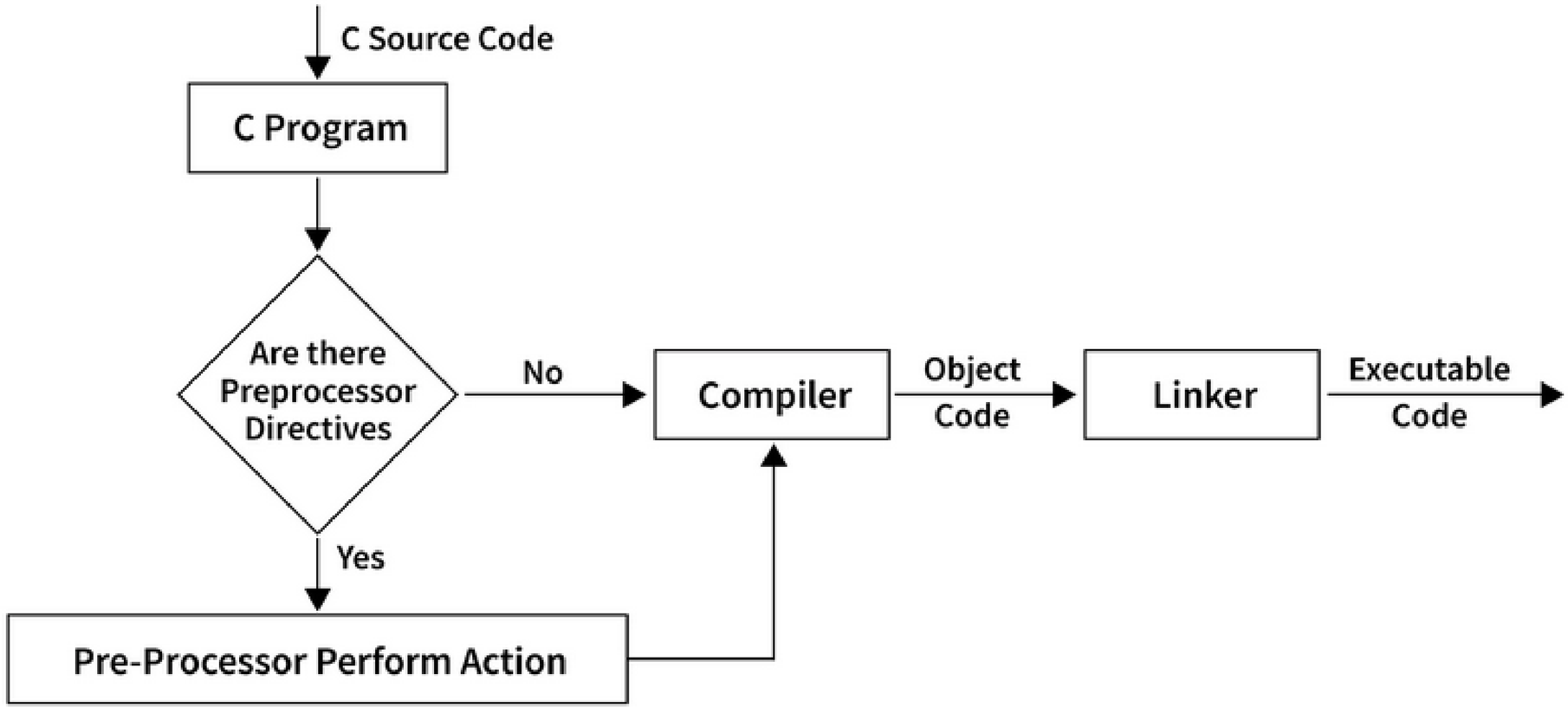
UNIT V – STRUCTURES AND UNIONS

TOPIC 4 – PREPROCESSOR DIRECTIVES

Introduction

- As the name suggests, the preprocessor directives in C are actually a text processing tool or we can call it a text substitution tool. All the preprocessor directives begin with a hash symbol #. It pre-processes the program before the compilation of the program.
- The C preprocessor is used automatically by the C compiler to transform the program before compiling. A series of textual transformations are done on its input and after that, it will return your C code to the compiler. And after that, the compiler compiles the code.
- All the preprocessor directive names start with a hash # which means all the statements starting with a # will first go to the preprocessor and then get executed.
- The **C preprocessor** (CPP) is not a part of the compiler but it's a step that executes before the source code is compiled.
- Preprocessor directives in C can appear anywhere in the program but they apply only to the remainder of the source file.

Flow Diagram



Types of Preprocessor Directives

There are Mainly four Types of Preprocessor Directives:-

1.Macros: The Macros directives replace every occurrence of the identifier with a predefined string which means we are substituting the identifier with a string. Its syntax is as follows:

```
#define identifier string
```

Example

```
#include <stdio.h>
#define PI 3.1415
main()
{
   printf("THE VALUE OF PI IS: %f",PI);
}
```

Explanation: The value of **PI** that is (3.1415 approx) is assigned to the string **PI**.

```
THE VALUE OF PI IS: 3.141500
```

2. File Inclusion: A file inclusion directive is an external file containing function macro definitions, that can be included using **#include** directive. The name suggests you need to include any file at the start of the program. The syntax of this directive is as follows:

```
#include <file name>
```

Example

```
#include <stdio.h>  // file inclusion directive is here
#define PI 3.1415
main()
{
   printf("THE VALUE OF PI IS: %f",PI);
}
```

Explanation: The header file **<stdio.h>** is used to include the functions like standard input and output functions.

3. Conditional Compilation: This C preprocessor offers a feature that is known as **conditional compilation** which means it can be used to switch the ON or OFF of a particular line or group of lines in a program. The "ON" or "OFF" will work for only that line of code. Using it, we can skip any part of the code as per our needs. Some examples:

```
#if, #else, #endif, etc..
```

Example

```c
#include <stdio.h>

#define Age 18

int main()
{
   #ifdef Age    // conditional compilation directives
   printf("This person is over %d years old.\n", Age);
   #endif  // also a conditional compilation directive

   printf("So, he is eligible.\n");

   return 0;
}
```

Explanation: Conditional directive **#ifdef** is used to apply the condition in the code, and **#endif** is used to terminate the code if the condition is satisfied.

```
This person is over 18 years old.
So, he is eligible.
```

4. Line Control: This preprocessor directive in C provided information about the error or mistakes in a program. It gives the line number where there is an error in syntax, indentation, etc. Syntax as follows:

```
Syntax 1
# line-number "file_name"

Syntax 2
# line line-number "file_name"
```

Explanation: Line control preprocessor directives **#pragma** is used to switch the control of line execution of code.

```c
#include <stdio.h>

void function_1();
void function_2();

#pragma startup function_1
#pragma exit function_2

void function_1()
{
        printf("Inside the function_1()\n");
}

void function_2()
{
        printf("Inside the function_2()\n");
}

int main()
{
        void function_1();
        void function_2();
        printf("Inside the main()\n");

        return 0;
}
```

```
Inside the main()
```

List of Preprocessor Directives

1. #include:

This directive is used to tell the system to include the current file specified in the input, in the rest of the original file. If the file that has to be included is not found, the compiler throws an error. This directive is of three types:

#include : This directive is used to include a header file. It searched the entire directory and finds that particular header file as specified by the user.

#include "file": This directive searched for the header file in the current directory. The directory of the current input file is called the current directory.

#include anything else: If any of the above two cases failed, then this directive is used.

2. #define:

This **preprocessor** directive in C comes under Macros. A macro is one when some particular section of code is substituted by the value of the macro. There are two types of Macros:

- Object-like Macros: When the macro is replaced by a value, generally for numerical value. Example:- Here, "PI" is used with the preprocessor dictie. So the value of **'PI' = 3.1415** will be assigned in the code.

  #define PI 3.1415

- Function-like Macros: When the macro id is replaced by a function call. Example:

  #define MIN(a,b) ((a)<(b)?(a):(b))

3. #undef:

This directive is used to remove the definitions related to a particular macro. The syntax of this directive is as follows:

```
#undef token
```

4. #ifdef:

This **preprocessor** directive is used to check if a macro is defined earlier or not. If it is defined, then the code executes normally. The syntax of this directive is as follows:

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

5. #ifndef:

This preprocessor directive is used to check whether a **"#define"** is defined or not defined by the user. If it is so, then the code will execute. The syntax of this directive is as follows:

```
#Ifndef MACRO
//code
#endif
#If
```

6. #if:

This preprocessor directive is used for evaluating the conditions or expressions. If the condition satisfies, the code will execute. The syntax of this derivative is as follows:

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

7. #else:

This preprocessor directive is also used for the evaluation of conditions or expressions. But here it is used to check if the condition is false, then what will further happen to the flow of code like where the compiler will shift for the execution of the program. We can use it with **#if, #ifdef, #elif, and #ifndef** directives. The syntax of this directive is as follows:

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

8. #elif:

This directive can be used with the **#if, #ifdef, or #ifndef** directives and the #elif. If the condition is true then it provides more conditions for the program. The syntax of this directive is as follows:

```
#elif condition
```

9. #endif:

This preprocessor directive is used to indicate the end of the directives like #if, #ifdef, or #ifndef condition.

10. #error:

This preprocessor directive is used to show an error. The compiler shows an **error directive** and it immediately stops the further compilation of the program. The syntax of this directive is as follows:

```
#error error_message
```

11. #pragma:

This preprocessor directive is used to present some extra information to the compiler. This directive can provide information such as machine details or the details of the operating system to the compiler. The syntax of this directive is as follows:

```
#pragma token
```

Conclusion

- Preprocessor directives in C are the simple text processing tool that starts with a hash #.
- All the codes before compilation, goes to the preprocessor for preprocessing.
- There are four types of derivatives in C that is Macros, File Inclusion, Conditional Compilation, and Line Control derivatives.
- Some part of the code is replaced by a name, known as **Macros**.