SNS COLLEGE OF TECHNOLOGY

Coimbatore-36.

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

COURSE NAME : 23CST101 PROBLEM SOLVING AND C PROGRAMMING

I YEAR/ V SEMESTER

UNIT – V STRUCTURES UNIONS AND FILES

PREPROCESSOR DIRECTIVES

Dr.B.Vinodhini

Associate Professor

Department of Computer Science and Engineering

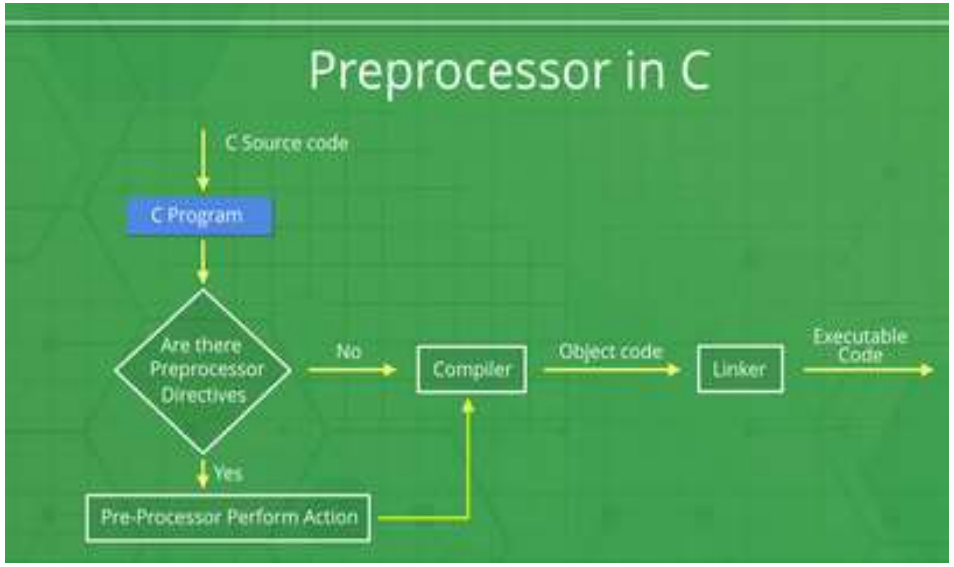
UNIT V

Defining Structures and Unions– Structure declaration – Need forStructure data type-Structure within a structure -Union -Programs using structures and Unions- **Pre-processor directives** –Files: Opening and Closing a Data File – Reading and writing a data file – Processing a data file - Illustrative programs

PREPROCESSOR DIRECTIVES

Preprocessors are *programs that process the source code before compilation*.

This process is called preprocessing.
Commands used in preprocessor are called preprocessor directives and they begin with "#" symbol.



Program process flow	File name in each steps	Description
Source code	sample.c	
Preprocessor		→ Preprocessor replaces #define (macro), #include (files), conditional compilation codes like #ifdef, #ifndef by their Respective values & source codes in source file
Expanded source code	sample.i	
Compiler		→ Compiler compiles expanded source code to assembly source code
Assembly source code	sample.s	
Assembler		→ It is a program that converts assembly source code to object code.
Object code	sample.o	
Linker		→This is a program that converts object code to executable code and also combines all object codes together.
Executable code	sample.exe	
Loader		→ Executable code is loaded in CPU and executed by loader program.
Execution		

PREPROCESSOR DIRECTIVES

Preprocessor Directives	Description
#define	Used to define a macro
#undef	Used to undefine a macro
#include	Used to include a file in the source code program
#ifdef	Used to include a section of code if a certain macro is defined by #define
#ifndef	Used to include a section of code if a certain macro is not defined by #define
#if	Check for the specified condition
#else	Alternate code that executes when #if fails
#endif	Used to mark the end of #if, #ifdef, and #ifndef

Types of C Preprocessors

There are 4 Main Types of Preprocessor Directives:

1. Macros
2. File Inclusion
3. Conditional Compilation
4. Other directives

   #undef Directive

   #pragma Directive

#define *token value*

PREPROCESSOR DIRECTIVES
1.Macros

Macros are pieces of code in a program that is given some name. Whenever this name is encountered by the compiler, the compiler replaces the name with the actual piece of code.

Syntax of Macro Definition

#define *token value*

The **'#define'** directive is used to define a macro.

```c
#include <stdio.h>

// macro definition
#define LIMIT 5

int main()
{
    for (int i = 0; i < LIMIT; i++) {
        printf("%d \n", i);
    }

    return 0;
}
```

In this program, when the compiler executes the word LIMIT, it replaces it with 5. The word **'LIMIT'** in the macro definition **is called a macro template** and **'5' is macro expansion.**

Note:

There is no semi-colon (;) at the end of the macro definition.

PREPROCESSOR DIRECTIVES
1.Macros With Arguments

Macros With Arguments

Pass arguments to macros. Macros defined with arguments work similarly to functions.

```
#define foo(a, b) a + b
#define func(r) r * r
```

```c
#include <stdio.h>

// macro with parameter
#define AREA(l, b) (l * b)

int main()
{
    int l1 = 10, l2 = 5, area;

    area = AREA(l1, l2);

    printf("Area of rectangle is: %d", area);

    return 0;
}
```

2.File inclusion

This type of preprocessor directive tells the compiler to include a file in the source code program. The **#include preprocessor directive** is used to include the header files in the C program.

There are two types of files that can be included by the user in the program:

1.Standard Header Files

#include<file_name>

where *file_name* is the name of the header file to be included. The **'<' and '>' brackets** tell the compiler to look for the file in the s**tandard directory.**

2. User-defined Header Files

When a program becomes very large, it is a good practice to divide it into smaller files and include them whenever needed. These types of files are user-defined header files.

Syntax

```
#include "filename"
```

The **double quotes (" ")** tell the compiler to search for the header file in the **source file's directory.**

3.Conditional Compilation

Conditional Compilation in C directives is a type of directive that helps to compile a specific portion of the program or to skip the compilation of some specific part of the program based on some conditions.

There are the following preprocessor directives that are used to insert conditional code:

1. **#if Directive**
2. **#ifdef Directive**
3. **#ifndef Directive**
4. **#else Directive**
5. **#elif Directive**
6. **#endif Directive**

#endif directive is used to close off the #if, #ifdef, and #ifndef opening directives which means the preprocessing of these directives is completed.

3.Conditional Compilation

```
#ifdef macro_name
    // Code to be executed if macro_name is defined
#ifndef macro_name
   // Code to be executed if macro_name is not defined
#if constant_expr
    // Code to be executed if constant_expression is true
#elif another_constant_expr
    // Code to be excuted if another_constant_expression is true
#else
    // Code to be excuted if none of the above conditions are true
#endif
```

If the macro with the name '*macro_name*' is defined, then the block of statements will execute normally, but if it is not defined, the compiler will simply skip this block of statements.

```
#include <stdio.h>

// defining PI
#define PI 3.14159

int main()
{

#ifdef PI
    printf("PI is defined\n");

#elif defined(SQUARE)
    printf("Square is defined\n");
#else
    #error "Neither PI nor SQUARE is defined"
#endif

#ifndef SQUARE
    printf("Square is not defined");
#else
    cout << "Square is defined" << endl;
#endif

    return 0;
}
```

Output

```
PI is defined
Square is not defined
```

4.Other Directives
#undef Directive

1. #undef Directive

The #undef directive is used to undefine an existing macro. This directive works as:

```
#undef LIMIT
```

Using this statement will undefine the existing macro LIMIT. After this statement, every "#ifdef LIMIT" statement will evaluate as false.

```c
// defining MIN_VALUE

#define MIN_VALUE 10

int main() {
    // Undefining and redefining MIN_VALUE
printf("Min value is: %d\n",MIN_VALUE);

//undefining max value
#undef MIN_VALUE

// again redefining MIN_VALUE
#define MIN_VALUE 20

    printf("Min value after undef and again redefining it: %d\n", MIN_VALUE);

    return 0;
}
```

4.Other Directives
#pragma Directive

This directive is a special purpose directive and is used to turn on or off some features. These types of directives are compiler-specific, i.e., they vary from compiler to compiler.

```
#pragma directive
```

#pragma startup: These directives help us to specify the functions that are needed to run before program startup (before the control passes to main()).
#pragma exit: These directives help us to specify the functions that are needed to run just before the program exit (just before the control returns from main()).

```c
#include <stdio.h>

void func1();
void func2();

// specifying funct1 to execute at start
#pragma startup func1
// specifying funct2 to execute before end
#pragma exit func2

void func1() { printf("Inside func1()\n"); }

void func2() { printf("Inside func2()\n"); }

// driver code
int main()
{
    void func1();
    void func2();
    printf("Inside main()\n");

    return 0;
}
```

Expected Output

```
Inside func1()
Inside main()
Inside func2()
```


Thank
you