PREPROCESSOR

PREPROCESSOR

- The C preprocessor is a tool that processes the source code before it is compiled.
- The commands of a preprocessor are called **preprocessor directives**.
- Each preprocessor directive begins with **#** symbol.
- They are **not** terminated with semi-colon.
- Can appeared anywhere in source file but are generally placed at the beginning of the program before the **main()**.

- Preprocessor Directives can be categorized as :

- ❖ FILE INCLUSION
- ❖ MACRO
- ❖ CONDITIONAL COMPILATION
- ❖ MISCELLANEOUS DIRECTIVES

FILE INCLUSION

- Given as **#include** command to the preprocessor to include header files.
- The entire contents of included file are added to the source code file.
- Library header file names are enclosed in **angle brackets < >.**
- In multi-file programs, user defined declarations or macro definitions are stored in header file and included in program using **#include ” “**

  - **#include “myFunc.h”** //searches in current working directory

  - **#include “user/common/Employee.h”** //searches in specified path

MACRO

- Macros are defined using **#define** directive.

- There are two types of macros :

1. SIMPLE MACRO
2. MACRO WITH ARGUMENTS

SIMPLE MACRO

- Macro with no arguments is called as **SIMPLE MACRO**.
- Syntax :

  #define MACRO_Template MACRO_Expansion
- During preprocessing, every macro template in the program gets replaced by macro expansion.
- Usually macro templates are written in capital letters. This makes easy for programmers to identify them.
- If macro expansion is longer than one line, it may be continued by placing backslash \ at the end of previous line. The backslash or newline gets deleted before preprocessing begins.

- The constant 3.1428 appears many times in your program. Some day you wish to change its value to more accurate 3.14287.
- For this you need to go through program and manually change each occurrence of the constant. But if you have define **PI** in #define directive, then you need to change it to

  ***#define* PI 3.14287**
- A macro can be a part of another macro
- Examples : ***#define* AND &&**

  ***#define* RANGE (a>25 AND a<50)**

  ***#define* WELCOME printf("WELCOME");**

MACRO WITH ARGUMENTS

- Macros can have arguments just like functions.

- Example : ***#define*** **AREA(x) (3.14*x*x)**
- No space between macro template and its argument.
- The entire macro expansion can be enclosed in parenthesis().

FUNCTION VS MACRO

	FUNCTION	**MACRO**
MEMORY REQUIRED	Less as only one copy exists.	More as expansion gets replace everywhere in program.
TIME REQUIRED	More. Since control shifts to called function	Less. Due to inline expansion
DATA TYPE	Considered	Not considered

- Use macros for small code.
- Use function to implement complex logic.

CONDITIONAL COMPILATION

- Make compiler skip over part of source code by using command **#ifdef** and **#endif**

```
int main()
{
        # ifdef NOTNOW
        statements1;
        statements2;
        #endif
}
```

- Statements 1 and 2 gets compiled only if macro NOTNOW is defined.
- If you want to compile statements 1 and 2 then either delete the #ifdef and #endif statements or define NOTNOW at the top.

- ***#ifdef*** can make programs portable

```
int main()
{
        # ifdef INTELI7
                code suitable for i7 machine
        # else
                code suitable for other machines
        # endif
                code common to both computers
}
```

- Sometimes, instead of **#ifdef,** the **#ifndef** directive is used
- The **#ifndef** (which means ‘if not defined’) works exactly opposite to **#ifdef**.

#if and #elif Directives

- The **#if** directive can be used to test whether an expression evaluates to a non-zero value or not.

```
int main()
{
        # if TEST <=5
        statement1;
        # else
        statement2;
        # endif
}
```

MISCELLANEOUS DIRECTIVES

- There are two more preprocessor directives available, though they are not commonly used.
- They are :

1. **#undef**
2. **#pragma**
3. **#error**

#undef

- It makes defined macro template to undefined

  # *undef* PENTIUM

- This would cause the definition of PENTIUM to be removed.
- All the subsequent **#ifdef** statements would evaluate to false.

#pragma

- **#pragma startup** directive allow us to specify function that can be called before **main().**

- **#pragma exit** directive allow us to specify function that can be called just before program terminates**.**

- **#pragma warn** directive tells compiler whether or not we want to suppress a specific warning.

#error

- **#error** forces the compiler to stop compilation.
- It is primarily used for debugging.
- Syntax:

  #error **error_message**

- When this directive is encountered, the error message is displayed,possibly alongother information depending on the compiler.