Course Name: O Level (2nd Sem B1, B2 and B3 Batch) **Subject:** C Language
Topic: Pre-processor in C **Date:** 09-June-2020

What is Preprocessor?

- Preprocessor processes source program before it is passed to compiler.
- Produce a source code file with the preprocessing commands properly sorted out.
- Preprocessor commands are known as ***directives.***
- Preprocessor provides certain ***features.***
- These features are also known as ***preprocessor directives.***
- Preprocessor directives start with #sign.
- Preprocessor directives can be placed anywhere in the source program.

Note: Place it at start of the program.

- Each preprocessor directive must be on its own line.

❑ Preprocessor Directives

- Macro Expansion
- File Inclusion
- Conditional compilation
- Miscellaneous directives

❑ Macro Expansion

- *#define directive is known as macro expansion.*

Definition:

#define PI 3.1415

General Form:

#define macro_template macro_expansion
#define macro_name char_sequence

- Preprocessor search for macro definition.
- After finding ***#define directive*** it search entire program for ***macro_template.***
- Replace each ***macro_template*** with ***macro_expansion.***
- **Best Practice:** Use capital letters for macro template.
- Do not use semicolon ***';'***

❑ Benefits of Macro

- To write efficient programs.
- To increase readability of programs.
- Variable Vs Macro_template
- Compiler can generate faster and compact code for constant than it can for variables.
- When you are dealing with a constant, why use variable.
- A variable may change in the program.

Replace operator:

#define AND &&
#define OR | |

Replace condition:

#define EXCELLENT (a>=75)

Replace statement:

#define ALERT printf("Security Alert");

- Defined macro name can be used as a part of definition of other macro name.
  #define MIN 1
  #define MAX 9
  #define MIDDLE (MAX-MIN)/2

- No text substitution occurs if the identifier is within a quoted string.

❑ **Macro With Arguments**

- Macros can have arguments, same as functions
  #define ISEXCELLENT(x) (x >= 75)
  #define ISLOWER(x) (x>=97 && x<=122)

- Macros expansions should be enclosed within parenthesis.
  #define ISLOWER(x) (x>=97 && x<=122)
  if(!ISLOWER('a'));

- Use '\' to split macro in multiple line.
  #define HLINE for(i=0; i < 40; i++)
  printf("_");

Differences between Structure and Union

Macro	Function
Macro is Preprocessed	Function is Compiled
No Type Checking	Type Checking is Done
Code Length Increases	Code Length remains Same
Use of macro can lead to side effect	No side Effect
Just the replacement of the code.	Passing arguments, doing calculation, returning results. (More serious work).
Macros make the program run faster.	Function calls and return make the program slow.
Before Compilation macro name is replaced by macro value	During function call , Transfer of Control takes place
Useful where small code appears many time	Useful where large code appears many time
Generally Macros do not extend beyond one line	Function can be of any number of lines
Macro does not Check Compile Errors	Function Checks Compile Errors

❑ **File Inclusion**

- Causes **one file to be included in another.**
  #include <filename> //OR
  #include "filename"

- ***<filename>:*** search the directory on current directory only.
- ***"filename":*** search the directory on current directory and specified directories as specified in the include search path.
- Divide a program in multiple files.
  - Each file contains related functions.

- Some functions or macros are required in each program.
  - Put them in a file (Library).
  - Include them in program that need them.

- **Nested includes:** Included file may have more included files in it.

- **Condtional Compilation**

Write single program to run on different environments.

- **#ifdef – if defined**
- **#endif – end if**
- **#else – else**
- **#ifndef – in not defined**
- **#if – if**
- **#elif – else if**

#ifdef & #endif

General form:

#ifdef macroname
statement sequence
#endif

- if macro name has been defined using ***#define*** the code between ***#ifdef*** & ***#endif*** will execute.

#else

- Use #else with #ifdef same as else with if.

General-form:

#ifdef macroname
statement sequence
#else
statement sequence
#endif

#ifndef

- ***#ifndef*** is just opposite to ***#ifdef***

#ifndef __file_h
#define __file_h

- ***#if*** directive test whether an expression evaluates to nonzero value or not.
- ***#elif*** used same as else if.

Where conditional compilation?

- Instead of comments.
  - Nested comments not allowed in C.

- Run the same code on different environment.
- To avoid multiple declaration error.

<u>Advantages of Macro</u>

- The advantage of using macro is the execution speed of the program fragment.
- When the actual code snippet is to be used, it can be substituted by the name of the macro.

The same block of statements, on the other hand, need to be repeatedly hard coded as and when required.

<u>Disadvantages of Macro</u>

- The disadvantage of the macro is the size of the program.
- The reason is, the pre-processor will replace all the macros in the program by its real definition prior to the compilation process of the program.