 **Exercise**

**Theory Question**

1. What is pre-processor directive?

 A pre-processor directive is a command in C/C++ that is processed by the compiler before the actual compilation of the code begins. These directives are used to give instructions to the compiler, like including files, defining constants, or conditional compilation. They begin with the # symbol.

1. What is difference between the macro and function?

* **Macro**: A macro is a fragment of code which is replaced by the preprocessor before the program is compiled. It is typically defined using #define. Macros are simple text replacements and can be constants or expressions.
* **Function**: A function is a block of code that performs a specific task and is executed during runtime. It is part of the compiled code and allows for code reuse and structured programming.

**Key Differences**:

* **Execution Time**: Functions are executed at runtime, while macros are expanded at compile-time.
* **Parameters**: Macros can accept parameters but are expanded literally, leading to possible issues with side effects, whereas functions handle parameters properly, ensuring safety.
* **Type Checking**: Functions undergo type checking, while macros do not.

1. Advantages and disadvantages of macro?

**Advantages**:

* **Efficiency**: Macros can provide faster execution since they are expanded during compilation.
* **Code Reusability**: Like functions, macros allow you to reuse code without repeating it.
* **No Overhead**: As they don't incur the overhead of function calls, macros can sometimes be faster, especially for small operations.

**Disadvantages**:

* **No Type Checking**: Macros don’t check the type of the parameters, which can lead to errors or unexpected results.
* **Debugging Difficulty**: Debugging macros can be challenging since they are expanded before compilation, making it harder to trace errors.
* **Side Effects**: Macros can lead to unintended side effects if their parameters are expressions with side effects (e.g., x++).
* **Readability**: Excessive use of macros can make code harder to read and maintain.

1. Write five list of pre-processor directive.

 **#include**: Includes a file (either standard or user-defined) into the program.

 **#define**: Defines a macro or constant.

 **#undef**: Undefines a macro, making it undefined.

 **#if / #elif / #else / # end if**: Conditional compilation, allowing different code to be included or excluded based on conditions.

 **#pragma**: Provides additional instructions to the compiler, often used for optimizations or specific compiler behaviors.

1. How many type of macro.

**Object-like Macros**: These macros define a value or constant.

**Function-like Macros**: These macros act like functions but do not include the overhead of function calls

**Practical Question**

1. Write a program which takes three integers ***a, b, c*** as input and prints the largest one using define directive macro.

#include <stdio.h>

#define MAX(a, b) ((a) > (b) ? (a) : (b)) // Macro to find the maximum of two numbers

#define LARGEST(a, b, c) MAX(MAX(a, b), c) // Macro to find the largest of three numbers

int main() {

int a, b, c;

// Take three numbers as input

printf("Enter three numbers: ");

scanf("%d %d %d", &a, &b, &c);

// Print the largest number

printf("The largest number is: %d\n", LARGEST(a, b, c));

return 0;

}

1. Which of the following are correctly formed ***#define*** statements and justify your reasons.
   1. **#**define **INCH PER FEET** 12

Let's break down each one:

**#define INCH PER FEET 12**

**Incorrect**: This is wrong because the macro name has spaces. A macro name can’t have spaces.

**Correct Version**: #define INCH\_PER\_FEET 12

* 1. **#**define **SQR** (X) ( X \* X )

**#define SQR (X) ( X \* X )**

**Correct**: This one is valid! It defines a macro SQR that squares a number. It correctly uses parentheses to ensure the proper order of operations.

Example: int result = SQR(5); // result will be 25

* 1. **#**define **SQR**(X) X \* X

**#define SQR(X) X \* X**

**Incorrect**: This is also valid syntax, but there’s a problem with how it works. Without parentheses around the whole expression, it can give wrong results when used with complex expressions.

Example: int result = SQR(3 + 2); // It will give 11 instead of 25

**Correct Version**: #define SQR(X) ((X) \* (X))

This will ensure that SQR(3 + 2) gives the correct result 25.

**MCQ Questions**

1. Pre-processor directive must be start \_\_\_\_\_ symbol
   1. $
   2. **#**
   3. \_
   4. &
2. The directives define anywhere in program but are most often used \_\_\_.

a) **At the beginning of a file**

* 1. Within the main() function
  2. Within the any function
  3. Nothing all

1. The directives is use to import other file in the current file.
   1. **#**define
   2. **#**undef
   3. **#**if
   4. #include

1. Macro is widely used to represent \_\_\_\_\_\_\_\_.
   1. Numeric
   2. String
   3. Both a and b
   4. Nothing all

1. Functions executes relatively slower than macros
   1. **True**
   2. false