Macros and User defined Library Functions

Macros

- A macro is a fragment of code that is given a name.
- You can define a macro in C using the #define preprocessor directive.
- Whenever a macro name is encountered by the compiler, it replaces the name with the definition of the macro.
- Macro definitions need not be terminated by a semi-colon(;).
- Macros are faster in execution.

Types of macros

1) **Object-like Macros:** An object-like macro is a simple identifier that will be replaced by a code fragment. It is popularly used to replace a symbolic name with numerical/variable represented as constant.

   Example: #define PI 3.142

2) **Function-like Macro:** These macros are the same as a function call. It replaces the entire code instead of a function name. Pair of parentheses immediately after the macro name is necessary.

   Example: #define circleArea(r) (3.1415*(r)*(r))

Simple code using #define preprocessor

```
#include <stdio.h>
#define PI 3.1415
 #define circleArea(r) (PI*r*r)
int main()
 {
    float radius, area;
    printf("Enter the radius: ");
    scanf("%f", &radius);
    area = circleArea(radius);
    printf("Area = %f", area);
    return 0;
}
```

Some Predefined macros

Macro	Value
__DATE__	A string containing the current date.
__FILE__	A string containing the file name
__LINE__	An integer representing the current line number.
__TIME__	A string containing the current date.

User Defined Header Files

- A header file is a file with extension **.h** which contains C function declarations and macro definitions to be shared between several source files.

There are two types of header files defined in a program:

System defined header file: The header file which is predefined i.e. the files that comes with your compiler is known as a system defined header file.

User-defined header file: The header file which is defined by the user is known as a user-defined header file.

- Both the user-defined and system-defined header file can be included in a program with the help of using preprocessing directive (#). These preprocessor directives are used to instruct the compiler to process these files before compilation. There are two forms of including header file:

```
#include<file>

#include "file"
```

User defined header file multiply123.h

```
// function to multiply two numbers and return the result.
int multiply(int a,  int b)
{
   return (a*b);
}
```

```
// C program to calculate the multiplication of two numbers
 #include<stdio.h>
 #include "multiply123.h"  // included user defined header file in source code
int main()
{
     int a =1, b = 2;
     printf(“ Result of multiplication: “, multply(a,b));
     return 0;

}
```

Library

A Library in Linux

- A library is a collection of pre-compiled pieces of code called functions.
- **Functions** are blocks of code that are reusable throughout a program.

	Static Library	**Dynamic Library**
Linking	happens as the last step of the compilation process. After the program is placed in the memory.	shared libraries are added during the linking process when executable files and libraries are added to the memory.
Means	Performed by Linker.	Performed by the operating system.
Size	much bigger, because external programs are built in the executable file.	much smaller, because there is only one copy of the dynamic library that is kept in memory.
Time	takes longer to execute	faster, because shared library code is already in memory.
External file changes	the executable file will have to be recompiled if any changes were applied to external files.	no need to recompile the executable.