2022-09-13 LECTURE 08 – MODULAR PROGRAMMING – I

- contents
    - function and module
    - modular programming
    - standard library functions (SLF)
    - user defined functions (UDF)
    - modular programming in LARP
    - modular programming in C
        - function
            - prototype
            - definition
            - call
    - concepts discussed in the lecture
    - exercises

- function
    - is a named block of code that performs a specific task
    - it only runs when it is called from another block of code
    - can receive parameters (data) and perform some specific task over it
    - can also return a value to calling block
    - there are two types of functions
        - standard library function
        - user defined function
- module and modular programming
    - module
        - a collection of related functions and variables that together provide some kind of service to a program that uses that module
        - one has to write several functions to write a module
        - C language does not support explicit syntactic support for modules writing
        - in C language it is done by using existing language features together like organizing physical code structure which means placing modules in different files etc.
    - modular programming
        - is to divide a bigger problem into smaller problems and to design and implement their solutions
        - programmer can write code for each smaller problem called module
        - these modules together provide solution of the bigger problem
        - it technique is also called **divide and conquer**
        - it helps in clarity and easy of design of a solution
        - it also helps in **reusability** of design and code to solve future problems which means a module can be used in other programs as well
        - modules were initially introduced in the form of subsystems (particularly for I/O) and software libraries

        - modular programming which has a goal of modularity was initially developed in the late 1960s and early 1970s, as a larger-scale extension of the concept of structured programming started in early 1960s.

- standard library functions (SLF)
    - C standard library functions or simply C library functions are inbuilt functions in C programming language
    - these built-in functions that are grouped together and placed at common locations called **header and library files**
    - the **prototype** of functions and related data definitions are placed in their respective header files
    - **functions definitions** are precompiled and stored in their respective library files
    - to use a standard library function, one needs to include respective header file in the program
    - standard library functions allow the programmer to use the pre-existing codes available in the C compiler without the need for the programmer to define his own code
    - it reduces overall programming efforts
    - there is a rich collection of standard library functions with their respective header files in C language that helps in performing common tasks like
        - mathematical calculations
        - string manipulations
        - character manipulations
        - input/output and
        - many other useful operations
    - examples of header files available in standard library are
        - <assert.h> contains information for adding diagnostics that aid program debugging
        - <ctype.h> contains function prototypes for functions that test characters for certain properties, and function prototypes for functions that can be used to convert lowercase letters to uppercase letters and vice versa
        - <float.h> contains the floating-point size limits of the system
        - <limits.h> contains the integral size limits of the system
        - <math.h> contains function prototypes for math library functions
        - <signal.h> contains function prototypes and macros to handle various conditions that may arise during program execution
        - <stdarg.h> defines macros for dealing with a list of arguments to a function whose number and types are unknown
        - <stdio.h> contains function prototypes for the standard input/output library functions and information used by them
        - <stdlib.h> contains function prototypes for conversions of numbers to text and text to numbers, memory allocation, random numbers and other utility functions
        - <string.h> contains function prototypes for string-processing functions
        - <time.h> contains function prototypes and types for manipulating the time and date
        - <errno.h> defines macros that are useful for reporting error conditions

- <locale.h> contains function prototypes and other information that enables a program to be modified for the current locale on which it's running. The locale notion enables the computer system to handle different conventions for expressing data such as dates, times, currency amounts and large numbers throughout the world
- <setjmp.h> contains function prototypes for functions that allow bypassing of the usual function call and return sequence
- <stddef.h> contains common type definitions used by C

- examples of library functions used in mathematics available in a header file named <math.h> are
  - sqrt(x) square root of x ➔ sqrt(900.0)=30.0, sqrt(9.0)=3.0
  - cbrt(x) cube root of x (C99 and C11 only) ➔ cbrt(27.0)=3.0, cbrt(-8.0)=-2.0
  - exp(x) exponential function e^x ➔ exp(1.0)=2.718, exp(2.0)=7.389
  - log(x) natural logarithm of x (base e) ➔ log(2.718)=1.0, log(7.389)=2.0
  - log10(x) logarithm of x (base 10) ➔ log10(1.0)=0.0, log10(10.0)=1.0
  - fabs(x) absolute value of x as a float ➔ fabs(13.5)=13.5, fabs(-13.5)=13.5
  - ceil(x) ceiling value of x (smallest integer >= x) ➔ ceil(9.2)=10.0, ceil(-9.8)=-9.0
  - floor(x) floor value of x (largest integer <=x) ➔ floor(9.2)=9.0, floor(-9.8)=-10.0
  - pow(x, y) x raised to power y (x^y) ➔ pow(2, 7)=128.0, pow(9, .5)=3.0
  - fmod(x, y) remainder of x/y as a floating ➔ fmod(13.657, 2.333)=1.992
  - sin(x) sine of x (x in radians) ➔ sin(0.0)=0.0
  - cos(x) cosine of x (x in radians) ➔ cos(0.0)=1.0
  - tan(x) tangent of x (x in radians) ➔ tan(0.0)=0.0

  - **example (demonstration of math standard library functions):** a program that call various match standard library functions prints their output at the console.

// L08-C01

```
#include <stdio.h>
#include <math.h>

int main() {
   double x = 10.10;
   printf("sqrt(x) square root of %.3lf is                    = %.3lf \n", x, sqrt(x));
   printf("cbrt(x) cube root of %.3lf is                      = %.3lf \n", x, cbrt(x));
   printf("exp(x) exponential function %.3lf is               = %.3lf \n", x, exp(x));
   printf("log(x) natural logarithm of %.3lf (base e) is      = %.3lf \n", x, log(x));
   printf("log10(x) logarithm of %.3lf (base 10) is           = %.3lf \n", x, log10(x));
   printf("fabs(x) absolute value of %.3lf is                 = %.3lf \n", x, fabs(x));
   printf("ceil(x) ceiling value of  %.3lf is                 = %.3lf \n", x, ceil(x));
   printf("floor(x) floor value of  %.3lf is                  = %.3lf \n", x, floor(x));
   printf("sin(x) ceiling value of  %.3lf is                  = %.3lf \n", x, sin(x));
   printf("cos(x) ceiling value of  %.3lf is                  = %.3lf \n", x, cos(x));
   printf("tan(x) ceiling value of  %.3lf is                  = %.3lf \n", x, tan(x));
   printf("pow(x,y) %.3lf raised to the power of %.3lf is     = %.3lf \n", x, 3.0, pow(x,3.0));
```

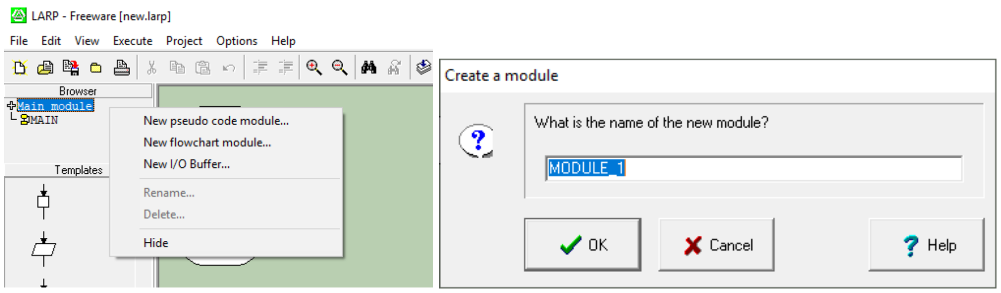
```
    printf("fmod(x,y) %.3lf raised to the power of %.3lf is  = %.3lf \n", x, 3.0,
fmod(x,3.0));
    return 0;
}
```

- **Output:** sqrt(x) square root of 10.100 is = 3.178
- **Output:** cbrt(x) cube root of 10.100 is = 2.162
- **Output:** exp(x) exponential function 10.100 is = 24343.009
- **Output:** log(x) natural logarithm of 10.100 (base e) is = 2.313
- **Output:** log10(x) logarithm of 10.100 (base 10) is = 1.004
- **Output:** fabs(x) absolute value of 10.100 is = 10.100
- **Output:** ceil(x) ceiling value of 10.100 is = 11.000
- **Output:** floor(x) floor value of 10.100 is = 10.000
- **Output:** sin(x) ceiling value of 10.100 is = -0.625
- **Output:** cos(x) ceiling value of 10.100 is = -0.781
- **Output:** tan(x) ceiling value of 10.100 is = 0.801
- **Output:** pow(x,y) 10.100 raised to the power of 3.000 is = 1030.301
- **Output:** fmod(x,y) 10.100 raised to the power of 3.000 is= 1.100

- user defined functions (UDF)
    - C language allows programmers to write their own functions such functions are called user defined functions
    - a programmer can create new functions, header files, and library files
    - can distribute their own written functions for mass consumption
- modular programming in LARP
    - LARP provide mechanism to design module and call these modules from main flowchart or pseudocode
    - design a module in LARP
        - **example (demonstration of a user defined module in LARP):** a program that call a module named MODULE_1, which prints "Pakistan Zindabad" at console, then return program control back to the MAIN flowchart and the MAIN terminates.

// L08-C02

- at Browser pane right click on Main module and select second option from popup list "New flowchart module" ➔ a "Create a module" dialogue will appear
- write name of the module "MODULE_1", and press OK

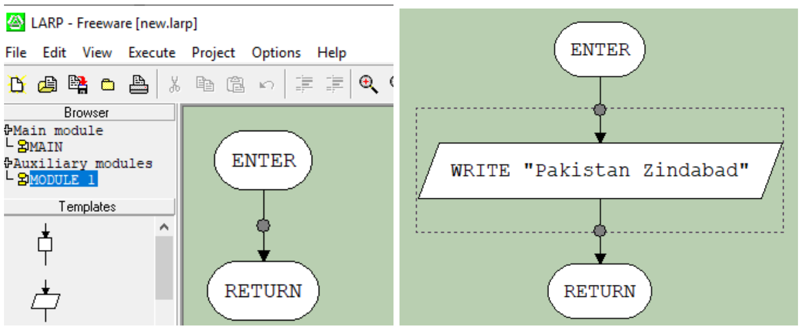


What is the name of the new module?

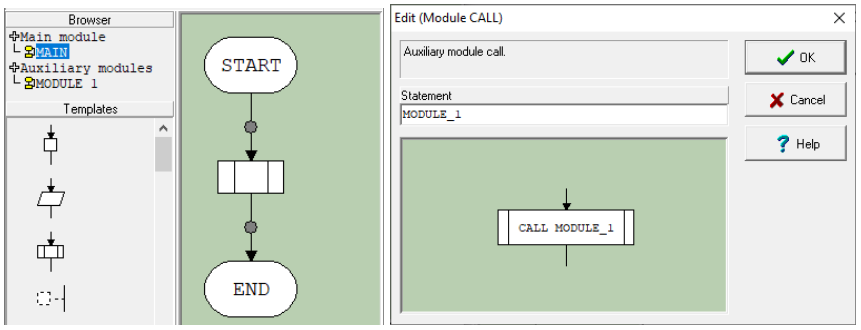
MODULE_1

- you will see a new module appears in the Brower pane under Auxiliary modules tree, and a new flowchart window will appear with the ovels having titles "ENTER" and "RETURN"

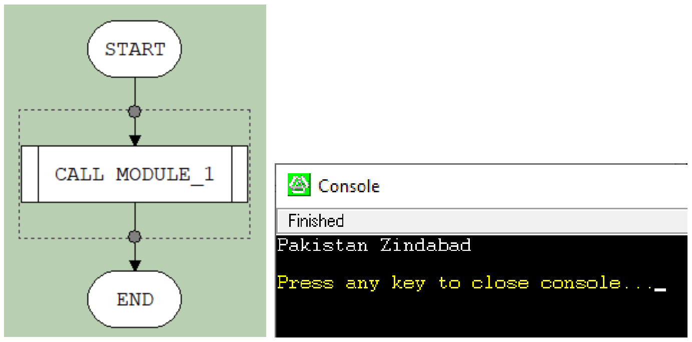
- embed an Input/Output flowchart symbol (parallelogram) with an output string "Pakistan Zindabad"


LARP - Freeware [new.larp]
File Edit View Execute Project Options Help
Browser
Main module
MAIN
Auxiliary modules
MODULE_1
Templates
ENTER
RETURN
ENTER
WRITE "Pakistan Zindabad"
RETURN

- Click on MAIN in the Brower pane under Main module
- This will show the main flowchart
- add Module CALL symbol from Templates pane and add it between START and END ovels of the MAIN flowchart
- double click Module CALL symbol in the MAIN flowchart ➔ an "Edit (Module CALL)" dialogue will appear
- write name of the module to be called from the main flowchart in the "Statement" textbox, in this case we will write "MODULE_1" in the "Statement" textbox and press OK



- a module call to "MODULE_1" will be inserted into the MAIN flowchart
- now if you press F7 or "Execute" from the "Execute" menu, it will display "Pakistan Zindabad" at console



```
Pakistan Zindabad

Press any key to close console..._
```

- in fact when we have pressed F7 and execute the main flowchart, it has CALLED the module named "MODULE_1"
- the program control is shifted from the MAIN module to the MODULE_1
- the MODULE_1 started execution and displayed "Pakistan Zindabad" at the console

        - on completing execution of all statements in the module MODULE_1 the program control returns back to the MAIN module
        - and after executing all statements after the **module CALL** in the MAIN module the program is terminated
    - in LARP one may design multiple modules and can call any module from any other module as and when required
    - the module in LARP behaves as functions in the C programming language
- modular programming in C
    - functions in the C programming language behaves as module in the LARP
    - writing and calling a user defined function in C language requires three different pieces of as follows
        - prototype
            - it is a single statement through which a function declared
            - it includes
                - **function name**
                - its **parameters**
                - **return type**
            - it is primarily used by compiler to check if **function call** confirms the correct syntax
            - for standard library functions, function prototypes are placed in their respective header files which are included before the code of main function
        - definition
            - it is a **header** and **a block of code** which is contained in curly braces { }
            - the **header of a function definition** is used to receive parameters
            - the **block of a function definition** contains the actual set of instruction which should be executed when the function is called
            - for standard library functions, function definitions are placed in their respective library files which are included in the object file of the program at link time with the help of linker
        - call
            - function call is a single statement which is used in any function to call any other function
            - when a function printf is called from main, the program control is shifted to the definition of function printf
            - then the function printf start executing instructions written inside it
            - once all instruction writing in the definition of function printf complete their execution
            - the **program control** is returned back to the next instruction in main function (the **caller**)
        - **example (demonstration of a user defined function):** a program that call a function named Square, which takes an integer as argument, compute its square and returns the value of the square to the calling function (the main), and then the program prints the square at the console.

```
// L08-C03
1.  #include <stdio.h>
2.
3.  int Square (int );
4.
5.  int main (void) {
6.     int n;
7.     n = Square (10);
8.     printf("%d", n);
9.  }
10.
11. int Square (int x) {
12.    int a;
13.    a = x * x;
14.    return a;
15. }
```

- **output/input:** 100

- **Line 3**: is the function prototype of function named Square which is also called function declaration,
  - this function takes an integer as argument and
  - also returns an integer value to its calling function
- **Line 7**: is a function call to function Square,
  - here an integer value 10 is passed to the function Square
  - and when the function Square will complete its execution, it will return a value which would be stored in the variable named n
- **Line 11-15**: it is a function definition of the function Square
  - **Line 11**: is the header of function Square which is similar to the function prototype,
    - only difference is that here the argument is received in a variable named x
    - the value (10) passed by the main function at Line 7 to the Square function is copied in the variable x
  - **Line 12&13:** an integer variable a is defined and square of x is stored in it
  - **Line 14&15:** the value which is stored in variable a is returned to the calling function (main), and the program control is retuned back to line 7 of the main function, in main function the value 100 returned by the Square function is stored in variable n
- **Line 8**: the value stored in the variable n would be displayed at the console,
- **Line 9:** and the program terminates
- **example (demonstration of a user defined function):** a program that call a function named Sum, which takes two integer as arguments, compute their sum and returns the value of the sum to the calling function (the main), and then the program prints the sum at the console.

```
// L08-C04
1.  #include <stdio.h>
2.
3.  int Sum (int , int);
4.
5.  int main (void) {
6.     int n;
7.     n = Sum (10, 20);
8.     printf("%d", n);
9.  }
10.
11. int Sum (int x, int y) {
12.    int a;
13.    a = x + y;
14.    return a;
15. }
```

- **output/input:** 30

- **Line 3**: is the function prototype of function named Sum which is also called function declaration,
    - this function takes two integers as argument and
    - also returns an integer value to its calling function
- **Line 7**: is a function call to function Sum,
    - here two integer value 10 and 20 are passed to the function Sum
    - and when the function Sum will complete its execution, it will return a value which would be stored in the variable named n
- **Line 11-15**: it is a function definition of the function Sum
    - **Line 11**: is the header of function Sum which is similar to the function prototype,
        - only difference is that here the arguments are received in two variables named x and y
        - the values (10, 20) are passed by the main function at Line 7 to the Sum function which are copied in the variable x and y respectively
    - **Line 12&13:** an integer variable a is defined and sum of x and y are stored in it
    - **Line 14&15:** the value which is stored in variable a is returned to the calling function (main), and the program control is retuned back to line 7 of the main function, in main function the value 30 is returned by the function Sum is stored in variable n
- **Line 8**: the value stored in the variable n would be displayed at the console,
- **Line 9:** and the program terminates

- concepts discussed in the lecture

- function, named block of code, calling block, standard library function, user defined function, parameters, module, modular programming, **reusability, divide and conquer,** software libraries, inbuilt functions, header files, library files, function **prototype, functions definition,** Auxiliary modules, Browser pane, “Create a module” dialogue, “ENTER” and “RETURN”, MAIN flowchart, **module CALL, function name, function parameters, function return type, function call,** header of a function definition, block of function definition, **caller**