

SLIIT ACADEMY

Higher Diploma in Information Technology
Year 1, Semester 1



Introduction to Programming(C++)

Lecture 06 : Modularization & Communication between modules

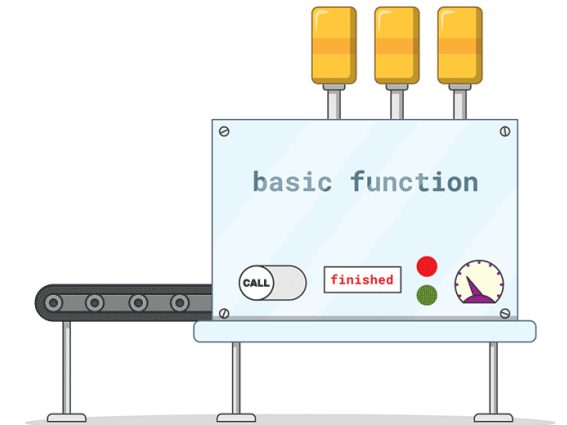
Intended Learning Outcomes

On the Completion of this lecture student will be able to learn ,

- L01: Understand the concept of the modularization.**
- L02: Understand the use of functions in a program.**
- L03: Define new functions in C++.**
- L04: Explain the function calls and argument passing.**
- L05: Apply the knowledge and write C++ programs with functions and arguments.**

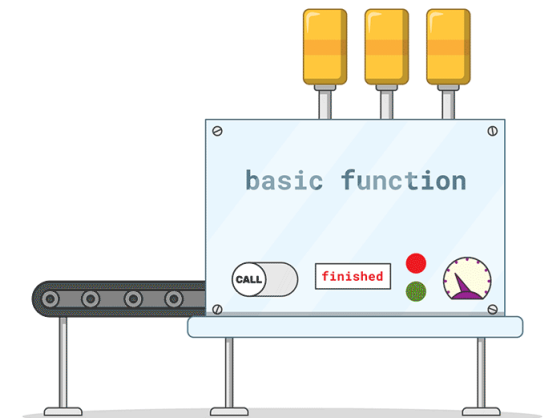
Modularization

- When the programming problem is complex, it is difficult to consider the solution as a whole. Therefore, we need to divide the problem into smaller parts.
- To do this, first identify the major tasks to be performed, and then divide the problem into sections that represent those tasks. These sections can be considered **subtasks or functions**.



Modularization

- Each of the subtasks or functions will eventually become a module within a solution program.
- A module, then, can be defined as a section of a program that is dedicated to a single function.
- **Modularization is the process of dividing a problem into separate tasks, each with a single purpose.**



Top-Down Design

- This process of identifying first the major tasks, then further subtasks within them, is known as '**top-down design**' (also known as functional decomposition or stepwise refinement).
- By using this top-down design methodology, we are **adopting a modular approach to program design**.

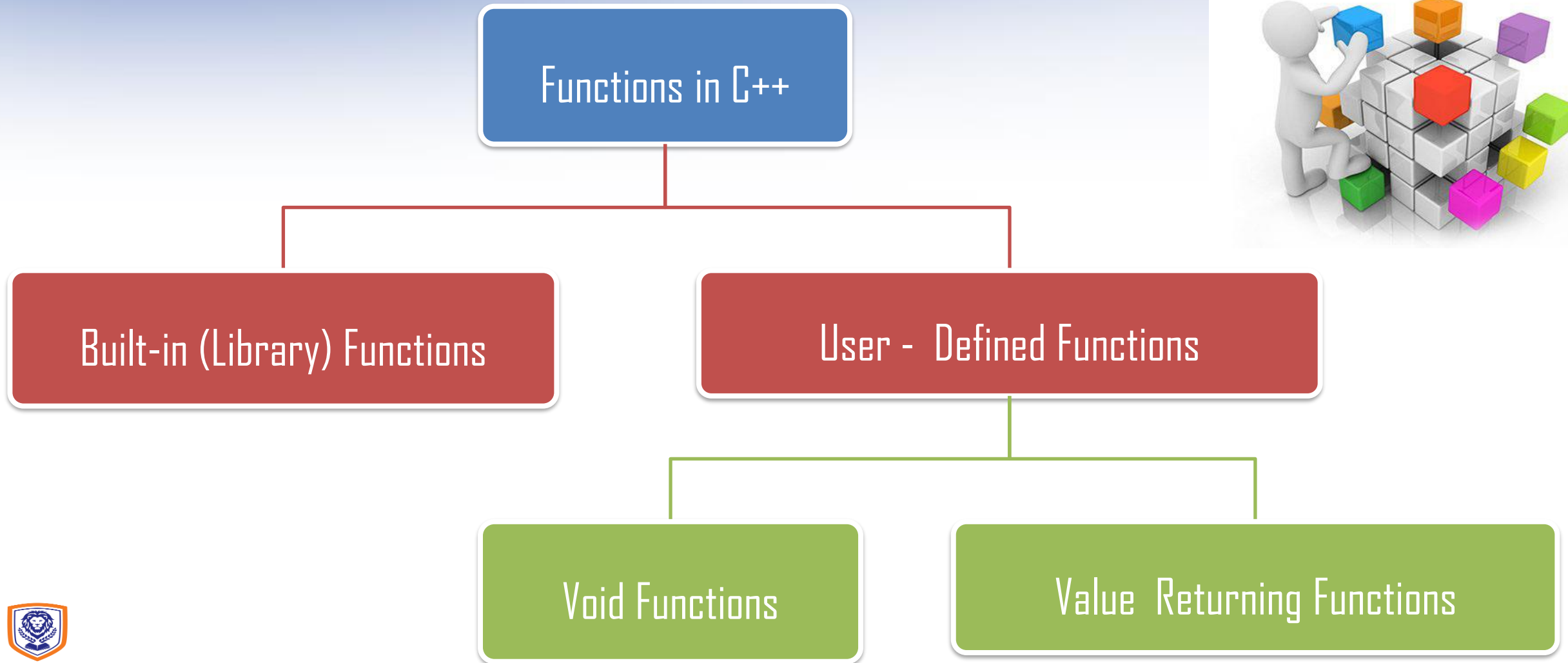
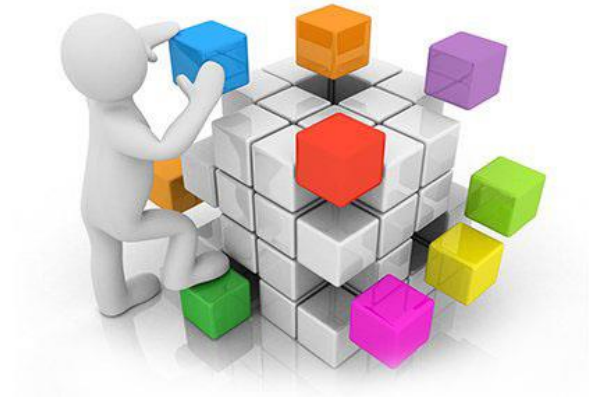


Advantages of Modularization

- **Ease of understanding:** Each module should perform just one function.
- **Reusable code:** Modules used in one program can also be used in other programs.
- **Elimination of redundancy:** Using modules can help to avoid the repetition of writing out the same segment of code more than once.
- **Efficiency of maintenance:** Each module should be self-contained and have little or no effect on other modules within the program.



Functions in C++



C++ Built-in (Library) Functions

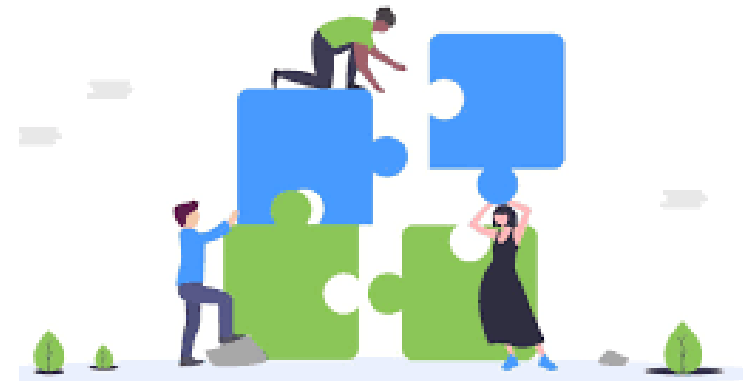
- Built-in functions are also called library functions. These are the functions that are provided by C++, and we need not write them ourselves. We can directly use these functions in our code.
- These functions are placed in the header files of C++.

Example : `<cmath>`, `<string>` are the headers that have in-built math functions and string functions respectively.



C++ User-Defined Functions

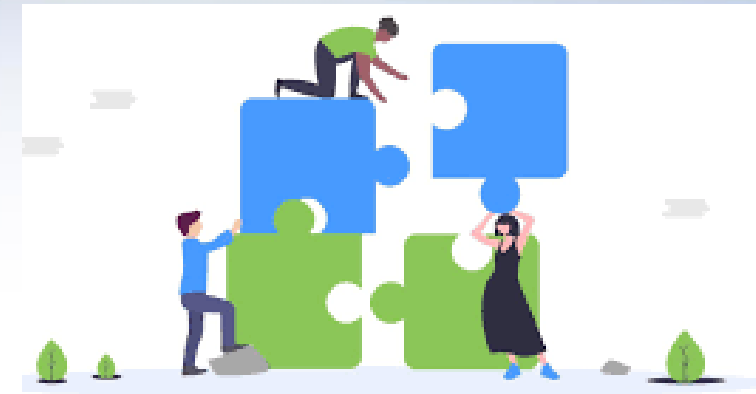
- C++ allows the programmer to define their own function.
- A user-defined function groups code to perform a specific task and that group of code is given a name (identifier).
- When the function is invoked from any part of the program, it all executes the codes defined in the body of the function.



C++ Function Definition

A function has two principal components.

- The function header(including the argument declarations)
- The body of the function



```
return-type function-name(parameter1, parameter2, ...)  
{  
    // function-body  
}
```

C++ Function Definition

A C++ function definition consists of a function header and a function body.

- **Return Type** – A function may return a value. The **return_type** is the data type of the value the function returns. Some functions perform the desired operations without returning a value. In this case, the return_type is the keyword **void**.
- **Function Name** – This is the actual name of the function. The function name and the parameter list together constitute the **function signature**.

Passing Parameters

- A particularly efficient method of inter module communication is the **passing of parameters or arguments between modules.**
- Parameters are simply data items transferred from a **calling module** to its **subordinate module** at the time of calling.
- When the subordinate module terminates and returns control to its caller, the values in the parameters are transferred back to the calling module.

Parameters

- A parameter is like a placeholder. The arguments are called **formal arguments** . Also known as **parameters or formal parameters** . Each argument is preceded by its associated type declaration.
- The identifiers used as **formal arguments are "local"**. In the sense that they are not recognized outside of the function.
- When a function is invoked, you pass a value to the parameter. This value is referred to as **actual parameter or argument**. Parameters are optional; that is, a function may contain no parameters.

C++ Function Definition

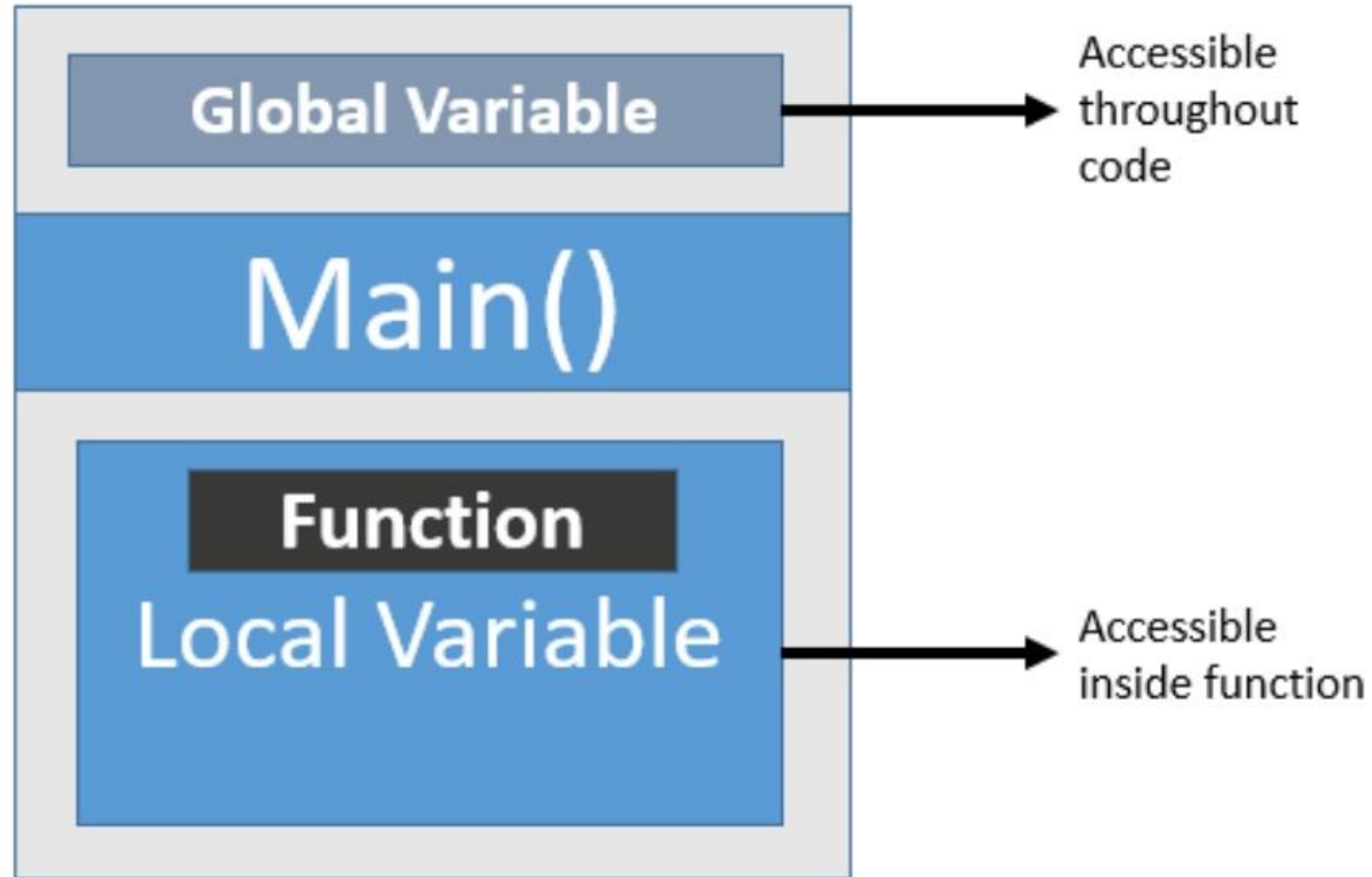
- **Function Body** – The function body contains a collection of statements that define what the function does.

Scope of Variable

- The scope of a variable is the portion of a program in which that variable has been defined and to which it can be referenced.
- The scope of variables can be defined with their declaration, and variables are declared mainly in two ways:
 - **Global Variable** : where the scope of the variable is the whole program.
 - **Local Variable** : where the scope of the variable is simply the module in which it is defined.



Local Variables Vs Global Variables



Practice Question 01

Write a C++ function to print hello world as the output.

```
void greet()  
{  
    cout<<"Hello World"<<endl;  
}
```



Practice Question 02

Write a C++ function to read two numbers, add them together and print their total.

```
int calculate_sum(int a ,int b)
{
    int sum ;
    sum = a + b;
    return sum;
}
```



C++ Function Declaration

- A function declaration tells the compiler about a function name and how to call the function.
- The actual body of the function can be defined separately.

```
return-type function-name (parameter1, parameter2, ...);
```

- For the defined function **sum()**, following is the function declaration:

```
int calculate_sum(int a , int b);
```


- Parameter names are not important in function declaration only their type is required,

```
int calculate_sum(int,int);
```

Calling a Function in C++

- While creating a C++ function, give a definition of what the function must do. To use a function, need to **call or invoke** that function.
- When a program calls a function, program control is transferred to the called function. A called function performs defined task and when it's return statement is executed or when its function-ending closing brace is reached, it returns program control back to the main program.
- To call a function, need to pass the required parameters along with function name, and if function returns a value, then can store returned value.

The Mainline (Main Program)

- Since each module performs a single specific task, a **mainline routine must provide the master control that ties all the modules together and coordinates their activity.**
- This program mainline should show the main processing functions, and the order in which they are to be performed.
- It should also show the flow of data and the major control structures.
-  The mainline should be easy to read, be of manageable length, and show sound logic structure.

Practice Question 03

Implement the main method and call the `greet()` and `calculate_sum()` functions.



Argument passing methods in C++

While calling a function, there are two ways that arguments can be passed to a function.

- Pass by value
- Pass by reference

pass by reference



fillCup()

pass by value



fillCup()

www.penjee.com

Pass by value

- By default, arguments in C++ are passed by value.
- When passing arguments by value, the only way to return a value back to the caller is via the function's return value.
- When arguments are passed by value, a copy of the argument is passed to the function.
- Because a copy of the argument is passed to the function, the original argument can not be modified by the function.

Pass by value

- When arguments are passed by value, **the called function creates a new variable of the same type as the argument and copies the argument's value into it.**
- As we noted, the function cannot access the original variable in the calling program, only the copy it created.
- Passing arguments by value is useful when the function does not need to modify the original variable in the calling program. In fact, it offers insurance that the function cannot harm the original variable.

Pass by Reference

- Passing arguments by reference uses a different mechanism.
- Instead of a value being passed to the function, a reference to the **original variable, in the calling program, is passed.**
- An important advantage of passing by reference is that **the function can access the actual variables in the calling program.**
- This provides a mechanism for passing more than one value from the function back to the calling program.

Pass by Reference

- Reference arguments are indicated by the ampersand (&) following the data type: `int &a`
- When the function is called, variable `a` will become a reference to the argument. **Since a reference to a variable is treated exactly the same as the variable itself**, then any changes made to the reference are passed through to the argument.

Pass by value vs Pass by Reference

```
#include <iostream>
using namespace std;
int squareByValue(int);           // function prototype
void squareByReference(int &);    // function prototype

int main()
{
    int x = 2 , z = 4;

    cout << "x = " << x << " before squareByValue\n";
    cout << "Value returned by squareByValue: " << squareByValue( x ) << endl;
    cout << "x = " << x << " after squareByValue\n" << endl;
    cout << "z = " << z << " before squareByReference" << endl;
    squareByReference( z );
    cout << "z = " << z << " after squareByReference" << endl;
    return 0;
}
```

Pass by value vs Pass by Reference

```
int squareByValue(int number)
{
    return number *= number;  // caller's argument not modified
}
```

```
void squareByReference(int &numberRef)
{
    numberRef *= numberRef;    // caller's argument modified
}
```

Summary

- Modular approach to program design.
- Top-down design.
- Functions in C++
- Defining , Implementing and calling a function.
- The differences between formal and actual parameters and value and reference parameters.