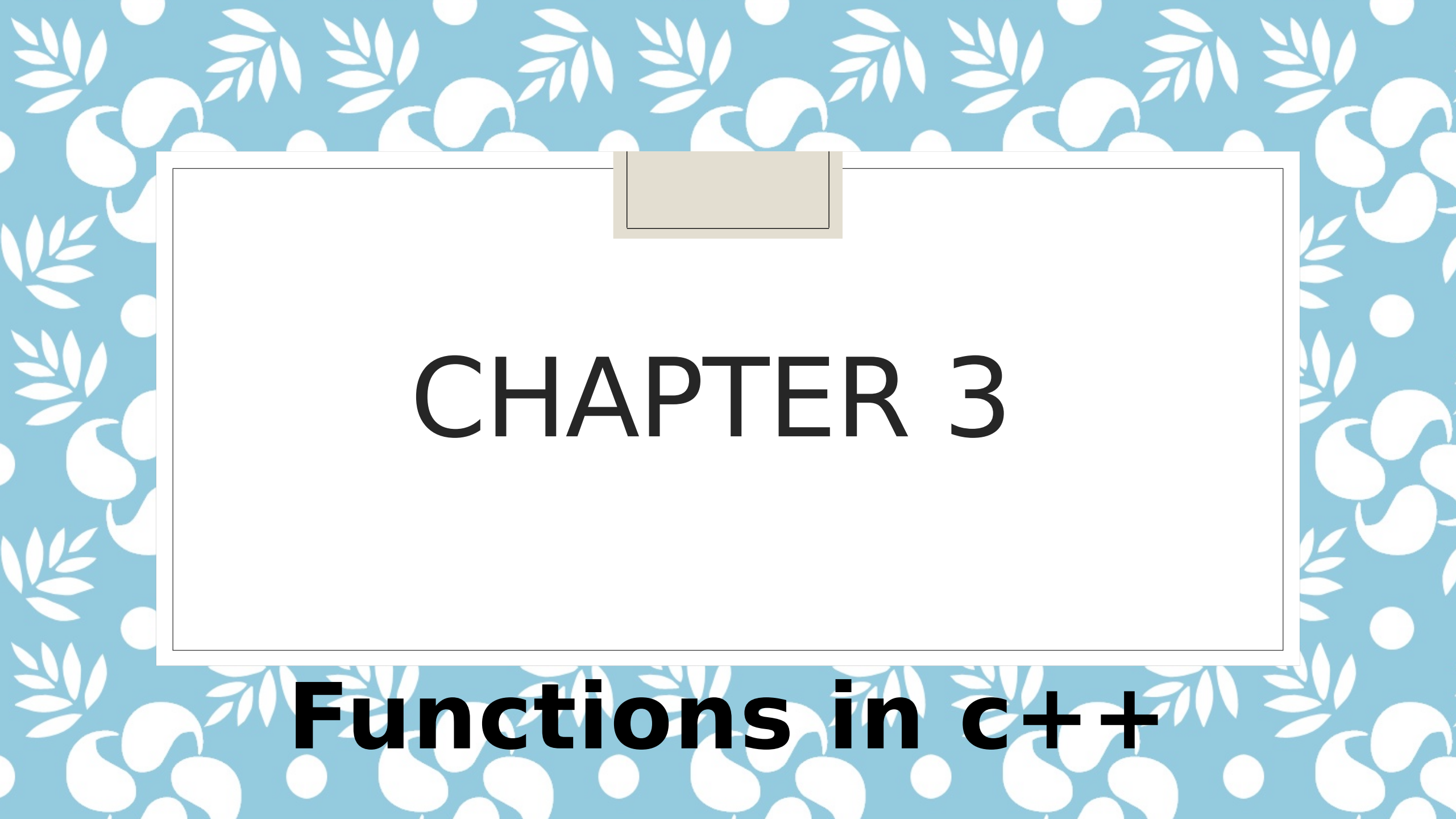




CHAPTER 3

Functions in c++

Definition

- Functions in C++ are a fundamental **building blocks of code** in a program that **perform a specific task**.
- They **allow code reuse, modularization, and abstraction, enhancing the readability and maintainability** of the code.
- A function is a **group of statements** that together **perform a task**.
- Every C++ program has at least one function, which is **main()**, and all the most trivial programs can define additional functions.
- You can divide up your code into separate functions.
- How you divide up your code among different functions is up to you, but logically the division usually is such that each function performs a specific task.

Create a Function

- C++ provides some pre-defined functions, such as `main()`, which is used to execute code.
- But you can also create your own functions to perform certain actions.
- To create (often referred to as declare) a function, specify the name of the function, followed by parentheses ():
- Function Declaration and Definition
- A C++ function consist of two parts:
 - **Declaration:** the return type, the name of the function, and parameters (if any)
 - A function **declaration** tells the compiler about a function's name, return type, and parameters.
 - **Definition:** the body of the function (code to be executed)
 - A function **definition** provides the actual body of the function.

Function Declaration

- In C++, if you **want to define the function** after the **main function**, then you **have to declare** that function first.
- A function declaration **tells the compiler** about the **number of parameters, data types of parameters, and returns type of function**.
- Writing parameter names in the function declaration is optional but it is necessary to put them in the definition.
- The syntax for a function declaration is:

0



Function Declaration

- The **return type** is the **data type** of the value that the **function will return when it is executed**.
- Then there is the function name, followed by the parameters which are not mandatory, which means a function may or may not contain parameters.
- In most programming languages, every function has a return type, even if it's implicitly defined.
- The return type specifies the kind of data that the function will produce as output when it is called.

Function Definition

- A function declaration introduces the function name and its type.
- It associates the function name/type with the function body.
- *Definition* specifies the body for a function, associates an identifier with the function, and allocates storage for it.
- A function definition specifies the body of the function.

```
return_type function_name ( parameter 1, parameter 2, . . . . )  
{  
  
    //function body  
  
}
```

Function Definition

- Ø **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.
- Ø **Function Name** – This is the actual name of the function. The function name and the parameter list together constitute the function signature.
- Ø **Parameters** – A parameter is like a placeholder. When a function is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument. The parameter list refers to the type, order, and number of the parameters of a function. Parameters are optional; that is, a function may contain no parameters.
- Ø **Function Body** – The function body contains a collection of statements that define what the function does.

```
return_type function_name ( parameter 1, parameter 2, . . . )  
{  
  
    //function body  
  
}
```

How to Call a Function in C++

- To call a function in C++, you need to provide the name of the function, along with any necessary arguments or parameters.
- The syntax for calling a function is:
- **function_name(argument1, argument2, ..., argumentN);**
- **function_name** is the name of the function you want to call, and **argument1, argument2, ..., argumentN** are any inputs or parameters that the function takes.
- If the function takes no parameters, you can simply use an empty pair of parentheses after the function name.

- **Function Declaration and Definition**

- **Declaration**

- `// Function declaration (prototype)`
 - `void functionName(int arg1, float arg2);`

- **Definition**

- `Function definition`
 - `void functionName(int arg1, float arg2) {`
`// Function body`
`// Code to perform the task`
`}`

Class Work

- Write a function that takes two integers as parameters and returns their sum.
- Write a function that takes two doubles as parameters and returns their product.
- Write a function that takes two integers as parameters and returns the maximum of the two.

Write a function that takes two integers as parameters and returns their sum.

```
1  #include <iostream>
2  using namespace std;
3  int sum(int x, int y); //Function Declaration
4  // Function to calculate the sum of two integers or Function Definition
5  int sum(int num1, int num2) {
6      return num1 + num2;
7  }
8  int main() {
9      int a, b;
10     // Input two integers
11     cout << "Enter first integer: ";
12     cin >> a;
13     cout << "Enter second integer: ";
14     cin >> b;
15     // Call the sum function and print the result
16     cout << "The sum of " << a << " and " << b << " is: " << sum(a, b) << endl;
17     //sum(a, b) this is function call
18     return 0;
19 }
```

Write a function that takes two integers as parameters and returns the maximum of the two.

```
1  #include <iostream>
2  using namespace std;
3  int maximum_of_two(int a, int b) {
4      int largest;
5      if(a>b)
6          largest=a;
7      else
8          largest=b;
9      return largest;
10 }
11
12 int main() {
13     int num1, num2;
14
15     // Prompting the user to input the values
16     cout << "Enter the first integer: ";
17     cin >> num1;
18     cout << "Enter the second integer: ";
19     cin >> num2;
20     // Calling the function and printing the result
21     cout << "Maximum of "<<num1 << " and " <<num2<< " is: "<<maximum_of_two(num1, num2)<< endl;
22     return 0;
23 }
```

Quiz (5%)

- Write a program to print the sum of two numbers entered by user by defining your own function.
- Write a program to print the circumference and area of a circle of radius entered by user by defining your own function.
- Define a program to find out whether a given number is even or odd using function.
- Write a program to find the factorial of a given number by using a function in C++ programming language.

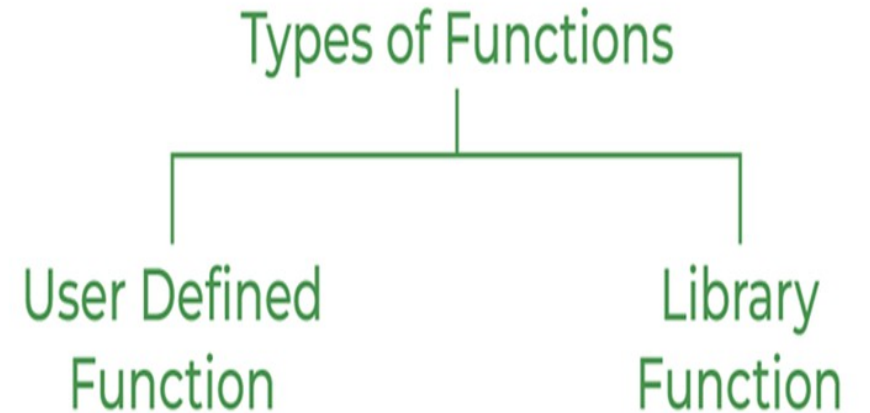
Factorial by Function

```
1  #include<iostream>
2  using namespace std;
3  int fact(int n){
4      int factorial = 1;
5      for (int i = 2; i <= n; i++){
6          factorial *= i;
7      }
8      return factorial;
9  }
10 int main(){
11     int n;
12     cout<<"Enter the number you want a factorial:";
13     cin>>n;
14     int ans = fact(n);
15     cout<<ans<<endl;
16     return 0;
17 }
```

Types of functions

➤ There are two types of function:

- **1. Standard Library Functions:** Predefined in C++
- **2. User-defined Function:** Created by users



- Library functions perform specific tasks and are ready to use without the need for explicit implementation by the programmer.
- Examples include functions for mathematical operations (sqrt, sin, cos), input/output operations (cout, cin), string manipulation (strlen, strcpy), and many others.
- Library functions save time and effort by providing commonly used functionalities that programmers can leverage in their programs.

Assignment: The C++ standard library provides numerous built-in functions that your program can call.

Different Types of User-defined Functions in C++

- A user-defined function is one that is defined by the user when writing any program, as we do not have library functions that have predefined definitions.
- To meet the specific requirements of the user, the user has to develop his or her own functions.
- Such functions must be defined properly by the user.
- There are four types of user-defined functions divided on the basis of arguments they accept and the value they return:
 - 1. Function with no arguments and no return value
 - 2. Function with no arguments and a return value
 - 3. Function with arguments and no return value
 - 4. Function with arguments and with return value

Function with No Arguments and No Return Value

- Functions that have no arguments and no return values can either be used to display information or to perform any task on global variables.
- In this type, as the name suggests, it passes no arguments to the function, and you print the output within the function. So, there is no return value.

```
// void return type with no arguments
void function_name()
{
    // no return value
    return;
}
```

```
void sum()
{
    int x, y;
    cout<<"Enter x and y\n";
    cin>>x>>y;
    cout<<"Sum of x and y is: "<<x
+ y;
}
```

Function with No Arguments and With Return Value

- Functions that have no arguments but have some return values are used to perform specific operations and return their value. In this type, it passes no arguments to the function, but there is a return value.

```
return_type function_name()
{
    // program
    return value;
}
```

```
int sum()
{
    int x, y, s = 0;
    cout<<"Enter x and y\n";
    cin>>x>>y;
    s = x + y;
    return s;
}
```

Function With Arguments and No Return Value

- Functions that have arguments but no return values. Such functions are used to display or perform some operations on given arguments.
- In this type of function, it passes arguments, but there is no return value. And it prints the output within the function.

```
void function_name(type1 argument1, type2 argument2,...typeN argumentN)
{
    // Program
    return;
}
```

```
void sum(int x, int y)
{
    cout<<"Sum of x and y is: "<<x + y;
}
```

Function With Arguments and With Return Value

- Functions that have arguments and some return value. These functions are used to perform specific operations on the given arguments and return their values to the user.
- In this type of function, it passes arguments, and there is also a return value.

```
return_type function_name(type1 argument1, type2 argument2,...typeN argumentN)
{
    // program
    return value;
}
```

```
int sum(int x, int y)
{
    return x + y;
}
```

Function Pass

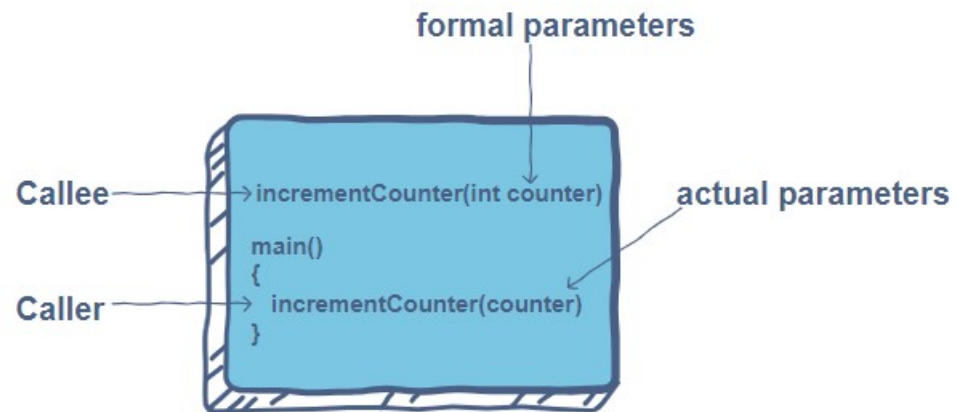
- Several ways exist in which data (or variables) could be sent as an argument to a function.
- Two of the common ones are **Passing by Value** and **Passing by Reference**.
- Passing by reference allows a function to modify a variable without creating a copy.
- We have to declare reference variables.
- The memory location of the passed variable and parameter is the same.
- Therefore, any change to the parameter also reflects in the variable inside its parent function.

Function Call By Value

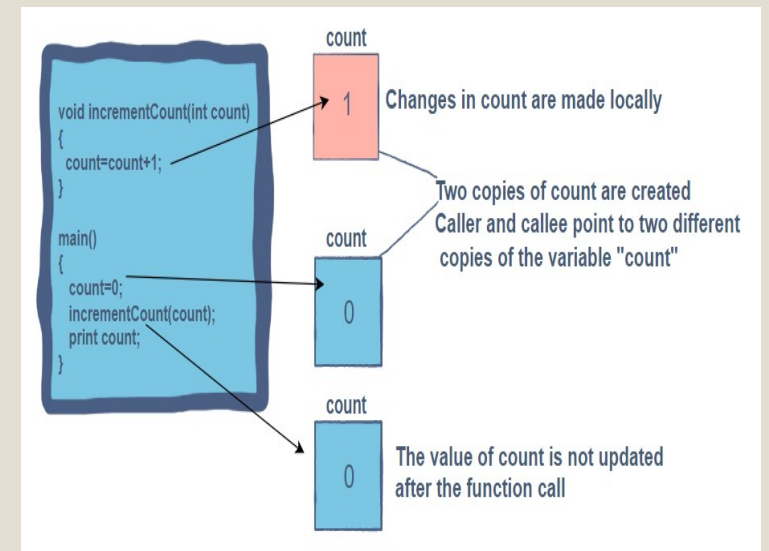
- A function is a collection of statements that accept inputs, carry out certain calculations, and output the results.
- The concept is to group similar or often performed actions into a function so that we may call the function rather than writing the same code again for various inputs.
- A function is a section of code that executes only when it is called.

Diagram illustrating the components of a function signature: `int sum ( int a , int b );`

- `int`: return type
- `sum`: Function Name
- `( int a , int b )`: Parameter Type and Parameter Name
- `;`: Ending Statement Semicolon

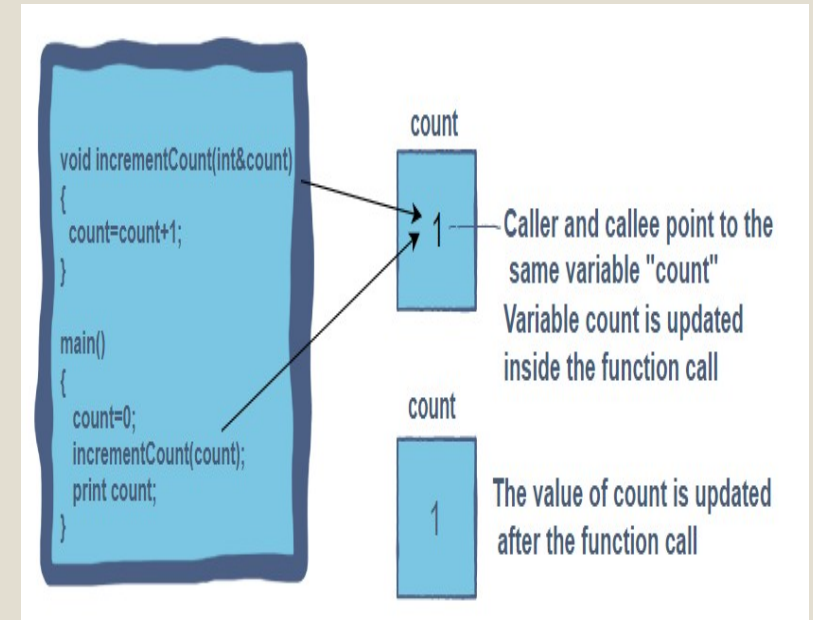


- **Pass by value** means that a copy of the actual parameter's value is made in memory, i.e. the caller and callee have two independent variables with the same value.
 - If the callee modifies the parameter value, the effect is not visible to the caller.
1. Passes an argument by value.
 2. Callee does not have any access to the underlying element in the calling code.
 3. A copy of the data is sent to the callee.
 4. Changes made to the passed variable do not affect the actual value.



Pass by reference

- **Pass by reference** (also called pass by address) means to pass the reference of an argument in the calling function to the corresponding formal parameter of the called function so that a copy of the address of the actual parameter is made in memory, i.e. the caller and the callee use the same variable for the parameter. If the callee modifies the parameter variable, the effect is visible to the caller's variable.
- Overview:
 1. Passes an argument by reference.
 2. Callee gives a direct reference to the programming element in the calling code.
 3. The memory address of the stored data is passed.
 4. Changes to the value have an effect on the original data.



Assignment –Presentation (10%)

- Function call by value and by reference- Theory
- **Write a c++ function that Swap two numbers using Pass-By-Reference and pass by value**

Function overloading and function overriding

- Function overloading and function overriding are two important concepts in C++ that involve the use of functions in classes and inheritance. Here's an explanation of each:
- **Function Overloading:**
- Function overloading refers to the ability to define multiple functions in the same scope (either within a class or a namespace) with the same name but different parameter lists.
- The compiler determines which version of the function to call based on the number and types of arguments passed to it.

Example of Function Overloading:

```
1  #include <iostream>
2  using namespace std;
3  class Calculator {
4  public:
5      // Overloaded function to add two integers
6      int add(int a, int b) {
7          return a + b;
8      }
9      // Overloaded function to add three integers
10     int add(int a, int b, int c) {
11         return a + b + c;
12     }
13 };
14 int main() {
15     Calculator calc;
16     cout << "Sum of 2 and 3: " << calc.add(2, 3) << endl;
17     cout << "Sum of 2, 3, and 4: " << calc.add(2, 3, 4) << endl;
18     return 0;
19 }
```

Sum of 2 and 3: 5
Sum of 2, 3, and 4: 9

In the above example, the add function is overloaded to accept different numbers of parameters.

Function Overriding:

- Function overriding occurs when a derived class provides a specific implementation of a method that is already defined in its base class.
- The function signature (name and parameters) in the derived class must match exactly with the function signature in the base class for overriding to occur.
- Function overriding is a feature of inheritance and is used to achieve runtime polymorphism.

Example of Function Overriding:

```
1 #include <iostream>
2 using namespace std;
3 class Animal {
4 public:
5     virtual void makeSound() {
6         cout << "Animal makes a sound" << endl;
7     }
8 };
9 class Dog : public Animal {
10 public:
11     // Overrides the makeSound method of the base class Animal
12     void makeSound() override {
13         cout << "Dog barks" << endl;
14     }
15 };
```

```
16 int main() {
17     Animal* animal = new Animal();
18     Dog* dog = new Dog();
19     animal->makeSound(); // Output: Animal makes a sound
20     dog->makeSound();    // Output: Dog barks
21     delete animal;
22     delete dog;
23     return 0;
24 }
```

```
Animal makes a sound
Dog barks
```

In this example, the **makeSound** function in the Dog class overrides the **makeSound** function in the Animal class.

The keyword `override` is used to explicitly indicate that the function is intended to override a base class function.

Recursive and Iterative Functions

- Recursive: Recursion is a programming technique where a function calls itself directly or indirectly to solve a problem.
- Recursive functions break down a problem into smaller, similar subproblems and solve each subproblem recursively.
- Recursion uses the call stack to manage function calls and track state, which can lead to stack overflow errors if not carefully implemented.
- Recursive solutions are often elegant and concise, especially for problems with recursive structure like tree traversal, factorial calculation, and Fibonacci sequence generation.
- Recursion is commonly used for tasks that can be naturally expressed in terms of self-similarity or when the problem is inherently recursive.

- Iterative Functions:
- Iteration is a programming technique where a loop structure (e.g., for, while, do-while) is used to repeatedly execute a block of code until a condition is met.
- Iterative functions use loop constructs to iterate through a sequence of steps or calculations, often updating mutable variables in each iteration.
- Iteration is generally more efficient in terms of memory usage and execution time compared to recursion, as it avoids the overhead of function calls and maintaining the call stack.
- Iterative solutions are preferred for problems that can be naturally expressed using loops and do not require the overhead of recursion, especially when performance is critical.

Example

- Write C++ program to find factorial of given number

```
1 // C++ program to find factorial of given number
2 #include<bits/stdc++.h>
3 using namespace std;
4 // ----- Recursion method to find factorial of given number
5 int factorialUsingRecursion(int n)
6 {
7     if (n == 0)
8         return 1;
9     // recursion call
10    return n * factorialUsingRecursion(n - 1);
11 }
12 // ----- Iteration find the factorial of a given number
13 int factorialUsingIteration(int n)
14 {
15     int res = 1, i;
16     // using iteration
17     for (i = 2; i <= n; i++)
18         res *= i;
19     return res;
20 }
```

```
21 // Driver method
22 int main()
23 {
24     int num = 5;
25     cout << "Factorial of " << num <<
26         " using Recursion is: " <<
27         factorialUsingRecursion(5) << endl;
28     cout << "Factorial of " << num <<
29         " using Iteration is: " <<
30         factorialUsingIteration(5);
31     return 0;
32 }
```

Comparison

Property	Recursion	Iteration
Definition	Function calls itself.	A set of instructions repeatedly executed.
Application	For functions.	For loops.
Termination	Through base case, where there will be no function call.	When the termination condition for the iterator ceases to be satisfied.
Usage	Used when code size needs to be small, and time complexity is not an issue.	Used when time complexity needs to be balanced against an expanded code size.
Code Size	Smaller code size	Larger Code Size.
Memory	Recursion uses more memory as compared to iteration.	Iteration uses less memory as compared to recursion.
Infinite Repetition	If the recursive function does not meet to a termination condition or the base case is not defined or is never reached then it leads to a stack overflow error and there is a chance that the an system may crash in infinite recursion.	If the control condition of the iteration statement never becomes false or the control variable does not reach the termination value, then it will cause infinite loop. On the infinite loop, it uses the CPU cycles again and again.

Exercises

- Write a function that takes an integer as a parameter and returns its factorial.
- Write a function that takes a double base and an integer exponent as parameters and returns the result of base raised to the power of exponent.
- Write a function that takes a string name as a parameter and prints a greeting message.
- Write a function that takes a string as a parameter and returns true if it's a palindrome (reads the same forwards and backwards), false otherwise.
- Write a function that takes an integer year as a parameter and returns true if it's a leap year, false otherwise.
- Write a function that takes an integer as a parameter and returns true if it's a prime number, false otherwise.
- Write a function that takes an integer **n** as a parameter and prints the first n Fibonacci numbers.