Programming with C++

[18K3MACS1]

Unit III,IV,V

E-Content Preparation Staff name

G.MUTHAMIZ SELVI

N.ANURADHA

Syllabus

Programming with C++ [18K3MACS1]

Unit-III:

Functions in C++ :Introduction – Main Function – Function prototyping – Call by Reference – Return by Reference – Inline Functions – Default Arguments – Const Arguments – Function Overloading – Friend and virtual functions – Math Library Functions.

Unit-IV:

Classes and Objects:Introduction – Specifying a class – Defining Member Functions – Nesting of Member Functions – Private Member Functions – Arrays within a Class – Memory Allocation for Object – Arrays of Objects – Object as Function Arguments – Friendly Functions – Returning Objects – Local Classes.

Unit-V:

Constructors and Destructors:Constructors – Parameterized Constructors – Multible Constructors – Dynamic Initialization of objects – Copy Constructors – Destructors – Exception Handling: Introduction – Basics of Exception Handling – Exception Handling Mechanisms – Throwing Mechanisms – Caching Mechanism – Rethrowing Mechanism – Specifying Exceptions.

TEXT BOOK:

Object Oriented Programming with C++"- E.Balagurusamy – TATA MC Graw Hill.

Functions in C++

Functions:

Dividing a program into functions is one of the major principles of Top down,Structured Programming.Functions is that it is possible to reduce the size of a program by calling and using them at different places in the program.

Dividing a complex problem into smaller chunks makes our program easy to understand and reusable.

- A function is a block of code which only runs when it is called.
- You can pass data, known as parameters, into a function.
- Functions are used to perform certain actions, and they are important for reusing code:

Define the code once, and use it many times.

There are two types of function:

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

User-defined Function:

- 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.

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 **()**:

Syntax:

```
void show()
{
 // code to be executed
}
```

show() is the name of the function

- void means that the function does not have a return value.
- inside the function (the body), add code that defines what the function should do

Call a Function:

- Declared functions are not executed immediately. They are "saved for later use", and will be executed later, when they are called.

- To call a function, write the function's name followed by two parentheses () and a semicolon ;

In the following example, Show() is used to print a text (the action), when it is called:

Inside main, call Show():

```
                    // Create a function
void Show()
 {
  cout << "I just got executed!";
}
int  main()
{
  Show();         // call the function
  return 0;
}
```

```
// Outputs "I just got executed!"
```

Function Declaration and Definition

A C++ function consist of two parts:

- **Declaration:** the function's name, return type, and parameters (if any)
- **Definition:** the body of the function (code to be executed)

```
// Function declaration:
void show();

// The main method
int main() {
  show();  // call the function
  return 0;
}

// Function definition
void show() {
  cout << "I just got executed!";
}
```

Function Parameters:

```
Return type function name(parameter1,parameter2…)
{
Function body
}
```

As mentioned above, a function can be declared with parameters (arguments). A parameter is a value that is passed when declaring a function.

For example, let us consider the function below:

```
void printNum(int num) {
   cout << num;
}
```

Return Statement:

In the above programs, we have used void in the function declaration. For example,

```
void displayNumber() {
   // code
}
```

This means the function is not returning any value.

It's also possible to return a value from a function. For this, we need to specify the `returnType` of the function during function declaration.
Then, the `return` statement can be used to return a value from a function.
For example,

```
int add (int a, int b) {
  return (a + b);
}
```

Here, we have the data type `int` instead of `void`. This means that the function returns an `int` value.
The code `return (a + b);` returns the sum of the two parameters as the function value.
The `return` statement denotes that the function has ended. Any code after `return` inside the function is not executed.

Main function:

main() function is the entry point of any C++ program. It is the point at which execution of program is started. When a C++ program is executed, the execution control goes directly to the main() function. Every C++ program have a main() function.

```
Void main()
{
……..
……..
}
```

- **void:** is a keyword in C++ language, void means nothing, whenever we use void as a function return type then that function nothing return. here main() function no return any value.
- In place of void we can also use **int** return type of main() function, at that time main() return integer type value.
- **main:** is a name of function which is predefined function in C++ library.

Function Prototype:

In C++, the code of function declaration should be before the function call. However, if we want to define a function after the function call, we need to use the function prototype. Syntax

Type function name (argument list);

The argument list contains the types and names of argument that must be passed to the function.

For example,

```
// function prototype
void add(int, int);

int main() {
   // calling the function before declaration.
   add(5, 3);
   return 0;
}

// function definition
void add(int a, int b) {
   cout << (a + b);
}
```

Call by reference:

- The **call by reference** method of passing arguments to a function copies the reference of an argument into the formal parameter.

- Inside the function, the reference is used to access the actual argument used in the call. This means that changes made to the parameter affect the passed argument.
- To pass the value by reference, argument reference is passed to the functions just like any other value.
- So accordingly you need to declare the function parameters as reference types as in the following function **swap()**, which exchanges the values of the two integer variables pointed to by its arguments.

function definition to swap the values.

```
void swap(int &x, int &y)
{
  int temp;
  temp = x; /* save the value at address x */
  x = y;    /* put y into x */
  y = temp; /* put x into y */

  return;
}
```

let us call the function swap() by passing values by reference as in the following example −

```
#include <iostream>
using namespace std;

// function declaration
void swap(int &x, int &y);

int main ()
{
  // local variable declaration:
  int a = 100;
  int b = 200;

  cout << "Before swap, value of a :" << a << endl;
  cout << "Before swap, value of b :" << b << endl;

  /* calling a function to swap the values using variable reference.*/
```

```
  swap(a, b);

  cout << "After swap, value of a :" << a << endl;
  cout << "After swap, value of b :" << b << endl;

  return 0;
}
```

Inline functions:

- C++ inline function is powerful concept that is commonly used with classes. If a function is inline, the compiler places a copy of the code of that function at each point where the function is called at compile time.
- To inline a function, place the keyword **inline** before the function name and define the function before any calls are made to the function. The compiler can ignore the inline qualifier in case defined function is more than a line.

Syntax:

Inline function-header

{

Function body

}

Default arguments:

- In the function definitions, if we declare default argument for one variable, then all the variables after that need...
- The default value for arguments are being copied if the argument value is provided while calling the function
- It shouldn't lead to ambiguity which will lead to a compilation error.

```
int mul (int i,int j=5,int k=10);   //legal
int mul (int i=5,int j);            //illegal
int mul (int i=0,int j,int k=10);  \\illegal
int mul (int i=2,int j=5,int k=10);  \\legal
```

const arguments:

- An argument to a function can be declared as const

  int strlen(const char*p);

  int length(const string &s);

- The qualifier const tells the compiler that the function should not modify the argument.this type of declaration is significant only when we pass arguments by reference or pointers**.**

Function Overloading:

- function overloading or method overloading is the ability to create multiple functions of the same name with different implementations.
- Function overloading is a feature in C++ where two or more functions can have the same name but different parameters.
- Function overloading can be considered as an example of polymorphism feature in C++.
- Following is a simple C++ example to demonstrate function overloading.

```
//Declaration
int add(int a,int b);
int add(int a,int b,int c);
double add(double x,double y);
double add(int p,double q);
double add(double p,int q);
//functions call
Cout<<add(5,10);
Cout<<add(15,10.0);
Cout<<add(12.5,7.5);
Cout<<add(5,10,15);
Cout<<add(0.75,5);
```

Friend Functions:

- A friend function of a class is defined outside that class' scope but it has the right to access all private and protected members of the class. Even though the prototypes for friend functions appear in the class definition, friends are not member functions.

- A friend can be a function, function template, or member function, or a class or class template, in which case the entire class and all of its members are friends.
- To declare a function as a friend of a class, precede the function prototype in the class definition with keyword **friend**

Virtual Functions:

Virtual function is the member function defined in the base class and can further be defined in the child class as well. While calling the derived class, the overwritten function will be called.

Math Library Functions:

The Standard C++ Supports many math functions that can be used for performing certain commonly used calculations.To use the math library functions,we must include the header file **math.h** in conventional C++ and **cmath** in ANSI C++

Chapter 5

Class and Object

class:

Both data and functions that are binded into a single unit.

A class in C++ is a user-defined type or data structure declared with keyword class that has data and functions as its members whose access is governed by the three access specifiers private, protected or public.

By default access to members of a C++ class is private. The private members are not accessible outside the class; they can be accessed only through methods of the class.

The public members form an interface to the class and are accessible outside the class. Instances of a class data type.

Specifying a class:

A class specifications has two parts:

1.Class declaration.

2.Class function definitions.

- The keyword class specifies,that what follows is an abstract data of type class_name.The body of the class is enclosed within braces and terminated by a semicolon.
- The class body contains the declarations of variables and functions.These functions and variables are collectively called class members.
- The keyword **private** and **public** are known as visibility labels.To denote which of the members are private and which of them are public.
- Variables declared inside the class are known as data members and the functions are known as member functions.
  Example:

```
Class item
{
int number;//var declarations
float cost;
public:
void getdata(int a,float b); //fun declarations
void putdata();
};
```

Object creations

Object : Run Time Entities.

- Once a class has been declared ,we can create variables of that type using the class name.

  Item X;

- The class variables are known as objects.x is called an object of type item.we may also declare more than one object in one statement.

  Example:

  Item x,y,z;

- The necessary memory space is allocated to an object at this stage.

Accessing class members:

- Private data of a class can be accessed only through the member functions of that class .
  For ex,Calling a Member functions

```
Object name.Function name(actual arguments);
```

- A variable declared as public can be accessed by the objects directly.

Defining Member functions:

Member functions can be defined Two parts

- **Outside the class definitions**.
  Member functions that are declared inside a class have to be defined separately outside the class.
  The general form:
  Return type Class name::function name(argument declaration)
  {
  Function body
  }

  Label (class name):: tells the compiler that the function (function name)belongs to the class (class name).The symbol :: is called the Scope resolution operator.
  Characteristics are:
    - Several different classes can use the same function name.The 'membership label' will resolve their scope.

- Member functions can access the private data of the class.A non-member functions cannot do so.
- A Member function can call another member function directly, without using the dot operator.

- **Inside the class definitions:**
  Another method of defining a member function is to replace the function declaration by the actual function definition inside the class.

```
Class item
{
int number;
float cost;
public:
void getdata(int a,float b);

//definition inside the class
void putdata(void)
{
Cout<<number<<"\n";
Cout<<cost<<"\n";
}
};
```

Nesting of member functions:

A member functions can be called by using its name inside another member function of the same class.This is known as nesting of member functions.

Ex:

```
int set :: largest()

{

if(m>=n)

Return(m);
```

```
else

Return(n);

}

Void set :: display(void)

{

Cout<<”Largest value=”<<largest()<<”\n”;

}
```

Private Member function

What are private functions?

Private Functions is a function can only be used inside of it's parent function or module. ... A public function can be used inside or outside of it. Public functions can call private functions inside them, some situation may require certain functions to be hidden from outside

```
Class sample
{
        Int m;
        Void read(void);
        Public :
        Void update();
        Void write();
};
```

How do you use private members in C++?

To access the private member, you can declare a function/class as friend of that particular class, and then the member will be accessible inside that function or class

Arrays within a class

The arrays can be used as member variables in a class

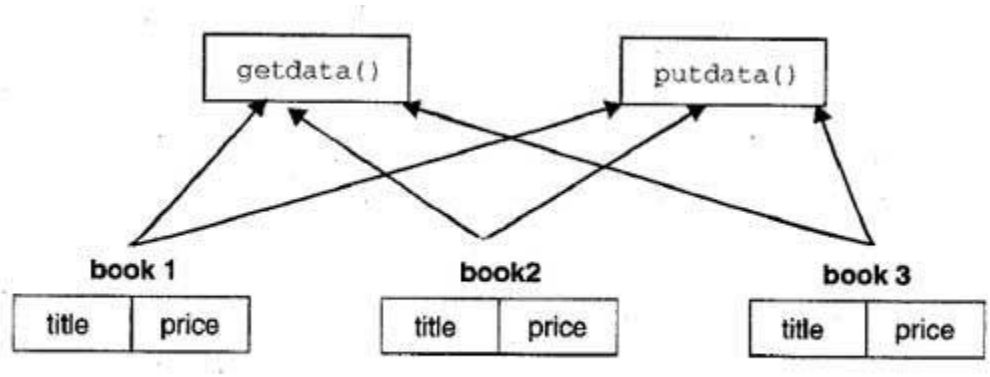
```
 Const int size = 10;

Class array

{

        int a[size];

        public:

           void setval(void);

          void display(void);

}
```

Memory Allocation for Objects

Before using a member of a class, it is necessary to allocate the required memory space tothat member. The way the memory space for data members and member functions is allocated is different regardless of the fact that both data members and member functions belong to the same class.

The memory space is allocated to the data members of a class only when an object of the class is declared, and not when the data members are declared inside the class. Since a single data member can have different values for different objects at the same time, every object declared for the class has an individual copy of all the data members.

On the other hand, the memory space for the member functions is allocated only once when the class is defined. In other words, there is only a single copy of each member function, which is shared among all the objects. For instance, the three objects, namely, book1, book2 and book3 of the class book have individual copies of the data members title and price. However, there is only one copy of the member functions getdata () and putdata () that is shared by all the three objects



Memory Allocation for the Objects of the Class book

Array of Objects

An array of objects, all of whose elements are of the same class, can be declared just as an array of any built-in type. Each element of the array is an object of that class. Being able to declare arrays of objects in this way underscores the fact that a class.

```
Class employee
{
Char name[30];
Float age;
Public:
Void getdata(void);
Void putdata(void);
};
```

objects that relate to different categories of the employees.

employee manager[3];

employee forman[3];

Object as Function Arguments

An object as a function argument.This can be done in two way

- A copy of the entire object is passed to the function.
- Only the address of the object is transferred to the function.

The first method is called ***pass-by-value*** a copy of the object is passed to the function any changes made to the object inside the function do not affect the object used to call the function.

The second method is called ***pass – by – reference*** .When an address of the object is passed the called function works directly on the actual object used in the call.

Friendly Functions

The private member function cannot access the outside the class and the non-member function cannot have an access to the private data of a class.

C++ allows the common function to be made friendly with both these classes,thereby allowing the function to have access to the private data of these classes,Such a function need not be a member of any of these classes.

To make an outside function "**friendly**"to a class , we have to simply declare this function as a friend of the class

```
Class ABC
{
          -------
   -------
Public:
   --------
  ---------
Friend void xyz(void);
 };
```

- The function declaration should be preceded by the keyword **friend**
- The function is elsewhere in the program. The function definition does not use either the keyword **friend or the scope operator ::** .
- The function that are declared with the keyword friend are known as friend functions.
- A function can be declared as a friend in any number of classes.

A friend function possesses certion special characteristics:

- ❖ It is not in the scope of the class to which it has been declared as friend.
- ❖ It is not in the scope of the class.it cannot be called using the object of that class.
- ❖ It can be invoked like a normal function without the help of any object.

UNIT V

CONSTRUCTORS AND DESTRUCTORS

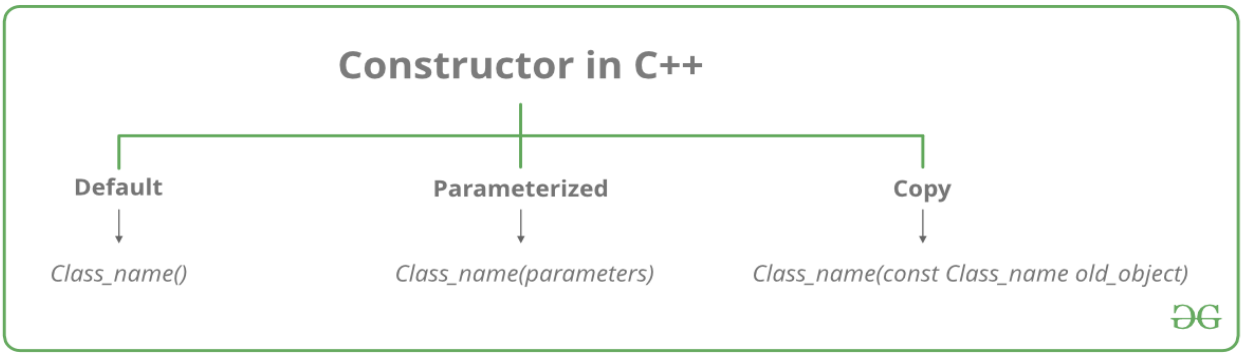
What is constructor?

A constructor is a member function of a class which initializes objects of a class. In C++, Constructor is automatically called when object(instance of class) create. It is special member function of the class.

How constructors are different from a normal member function?

A constructor is different from normal functions in following ways:

- ➢ Constructor has same name as the class itself
- ➢ Constructors don't have return type
- ➢ A constructor is automatically called when an object is created.



Types of Constructors

Default Constructors: Default constructor is the constructor which doesn't take any argument. It has no parameters.

```
#include <iostream>
using namespace std;

class construct
{
public:
   int a, b;

   // Default Constructor
   construct()
   {
      a = 10;
      b = 20;
   }
};

int main()
{
   // Default constructor called automatically
   // when the object is created
   construct c;
   cout << "a: " << c.a << endl
        << "b: " << c.b;
   return 1;
}
```

2. Parameterized Constructors: It is possible to pass arguments to constructors. Typically, these arguments help initialize an object when it is created. To create a parameterized constructor, simply add parameters to it the way you would to any other function. When you define the constructor's body, use the parameters to initialize the object.

```
#include <iostream>
using namespace std;

class Point
{
private:
```

```
    int x, y;

public:
    // Parameterized Constructor
    Point(int x1, int y1)
    {
        x = x1;
        y = y1;
    }

    int getX()
    {
        return x;
    }
    int getY()
    {
        return y;
    }
};

int main()
{
    // Constructor called
    Point p1(10, 15);

    // Access values assigned by constructor
    cout << "p1.x = " << p1.getX() << ", p1.y = " << p1.getY();

    return 0;
}
```

Uses of Parameterized constructor:

- ❖ It is used to initialize the various data elements of different objects with different values when they are created.
- ❖ It is used to overload constructors.

Can we have more than one constructor in a class?

Yes, It is called Constructor Overloading.

Copy Constructor: A copy constructor is a member function which initializes an object using another object of the same class.

```
#include "iostream"
using namespace std;

class point
{
private:
  double x, y;

public:

  // Non-default Constructor &
  // default Constructor
  point (double px, double py)
  {
    x = px, y = py;
  }
};

int main(void)
{

    point b = point(5, 6);
}
```

DYNAMIC CONSTRUCTORS

The constructors can also be used to allocate memory while creating object .This will enable the system to allocate the right amount of memory for each object when the objects are not the same size.,thus the resulting in the saving memeory.

Allocation of memery to object at the time of their construction is known as dynamic construction of object.

Destructors:

A destructor as the name implies is used to destroy the objects that have been created a constructor. The destructors is a member function whose namr is the same as the class name but is preceded by a **tilde (~)**

~ interger() { }

EXCEPTION HANDLING

An exception is a problem that arises during the execution of a program. A C++ exception is a response to an exceptional circumstance that arises while a program is running, such as an attempt to divide by zero.

- Exceptions provide a way to transfer control from one part of a program to another. C++ exception handling is built upon three keywords: try, catch, and throw.

- throw − A program throws an exception when a problem shows up. This is done using a throw keyword.

- catch − A program catches an exception with an exception handler at the place in a program where you want to handle the problem. The catch keyword indicates the catching of an exception.

- try − A try block identifies a block of code for which particular exceptions will be activated. It's followed by one or more catch blocks.

Assuming a block will raise an exception, a method catches an exception using a combination of the try and catch keywords. A try/catch block is placed around the code that might generate an exception. Code within a try/catch block is referred to as protected code, and the syntax for using try/catch as follows –

```
try {
  // protected code
                } catch( ExceptionName e1 ) {
                 // catch block
                } catch( ExceptionName e2 ) {
                 // catch block
                } catch( ExceptionName eN ) {
                 // catch block
}
```

Throwing Exceptions

Exceptions can be thrown anywhere within a code block using throw statement. The operand of the throw statement determines a type for the exception and can be any expression and the type of the result of the expression determines the type of exception thrown.

Following is an example of throwing an exception when dividing by zero condition occurs −

```
double division(int a, int b) {
  if( b == 0 ) {
    throw "Division by zero condition!";
  }
  return (a/b);
}
```

Catching Exceptions

The catch block following the try block catches any exception. You can specify what type of exception you want to catch and this is determined by the exception declaration that appears in parentheses following the keyword catch.

```
try {
  // protected code
} catch( ExceptionName e ) {
 // code to handle ExceptionName exception
}
```

Above code will catch an exception of ExceptionName type. If you want to specify that a catch block should handle any type of exception that is thrown in a try block, you must put an ellipsis, ..., between the parentheses enclosing the exception declaration as follows −

```
try {
  // protected code
} catch(...) {
 // code to handle any exception
}
```

The following is an example, which throws a division by zero exception and we catch it in catch block.

Live Demo

```
#include <iostream>
using namespace std;

double division(int a, int b) {
  if( b == 0 ) {
    throw "Division by zero condition!";
  }
```

```
   return (a/b);
}

int main () {
   int x = 50;
   int y = 0;
   double z = 0;

   try {
      z = division(x, y);
      cout << z << endl;
   } catch (const char* msg) {
     cerr << msg << endl;
   }

   return 0;
}
```

Because we are raising an exception of type const char*, so while catching this exception, we have to use const char* in catch block. If we compile and run above code, this would produce the following result −

```
Division by zero condition!
```