**LAB**

**MANUAL**

**COURSE Title: Object Oriented Programming**

**Course Code: CSC-102**

**Credit Hours: 4(3-3)**



Prepared By:

**Dr. Kashif Sattar (Coordinator)**

**Dr. Tariq Ali (Member)**

**Dr. Saqib Majeed (Member)**

**Mr. M. Ahsan Abbasi (Member)**

**Dr. Rubina Ghazal (Member)**

**Dr. Sidra Tahir (Member)**

University Institute of Information Technology

PMAS-Arid Agriculture University, Rawalpindi

Table of Contents

|  |  |  |
| --- | --- | --- |
| **Lab #** | **Topics Covered** | **Page #** |
|  |  |  |
| Lab # 01 | Programming Exercise for Classes and Objects | 3 |
| Programming Exercise that how to access private & public members of class |
| Programming Exercise that how functions defined outside the class |
| Lab # 02 | Programming Exercise for constructor | 7 |
| Programming Exercise for destructor |
| Programming Exercise for Constructor Overloading |
| Lab # 03 | Programming Exercise for constant variables and function access. | 12 |
| Programming Exercise for Static variables and function access. |
| Programming Exercise for concept of composition vs inheritance. |
| Lab # 04 | Programming Exercise for friend functions. | 18 |
| Programming Exercise for friend classes. |
| Programming Exercise for inheritance, constructors call sequence. |
| Lab # 05 | Programming Exercise for function overriding. | 25 |
| Programming Exercise for multilevel inheritance. |
| Programming Exercise for multiple inheritance and constructor calling. |
| Lab # 06 | Programming Exercise for string class built-in functions. | 36 |
| Programming Exercise for defining own String class. |
| Lab # 07 | Programming Exercises for unary and binary operator overloading. | 55 |
| Programming Exercises for multiple operator overloading. |
| Programming Exercises for (), [] and = operator overloading. |
| Lab # 08 | Programming Exercise for conversion between objects of different classes. | 60 |
| Programming Exercise for virtual functions. |
| Programming Exercise for abstract class. |
| Lab # 09 | Programming Exercise for dynamic memory creation. | 66 |
| Programming Exercise for dynamic memory deletion |
| Lab # 10 | Programming Exercise for template functions. | 69 |
| Programming Exercise for template classes. |
| Programming Exercise for template classes with multi template variables. |
| Lab # 11 | Programming Exercise for concept of exception | 74 |
| Programming Exercise for exception with multi error classes with arguments. |  |
| Lab # 12 | Programming Exercise for STL (Stack library) | 81 |
| Programming Exercise for Vector and List |
| Lab # 13 | Java installation | 83 |
| Program Compilation and Execution in Java. |
| Object Filing (Reading, Writing, Update & delete) in C++ |
| Object Filing (Reading, Writing, Update & delete) in Java. |



**LAB # 01**

**Lab Objective:**

Objective of this lab is to make students understand the use of classes and objects, Data members and Member functions of class, Member functions defined inside and outside the class.

**Activity Outcomes:**

The student will understand the advantages of classes and objects, Data members and Member functions of class, Member functions defined inside and outside the class.

**Instructor Note:**

The Students should have knowledge about C++ compiler, variables and functions.

**Introduction:**

Object-oriented programming, or OOP, is an approach to problem-solving where all computations are carried out using objects. An object is a component of a program that knows how to perform certain actions and how to interact with other elements of the program. Objects are the basic units of object-oriented programming. A simple example of an object would be a person. Logically, you would expect a person to have a name. This would be considered a property of the person. You would also expect a person to be able to do something, such as walking. This would be considered a method of the person.

A method in object-oriented programming is like a procedure in procedural programming. The key difference here is that the method is part of an object. In object-oriented programming, you organize your code by creating objects, and then you can give those objects properties and you can make them do certain things. Most modern programming languages including Java C++, C#, Python and PHP are Object Oriented.

One of the most important characteristics of procedural programming is that it relies on procedures that operate on data - these are two separate concepts. In object-oriented programming, these two concepts are bundled into objects. This makes it possible to create more complicated behavior with less code. The use of objects also makes it possible to reuse code. Once you have created an object with more complex behavior, you can use it anywhere in your code.

**Advantages of OOP:**

• OOP provides a clear modular structure for programs.

• It is good for defining abstract data types.

• Implementation details are hidden from other modules and other modules has a clearly defined interface.

• It is easy to maintain and modify existing code as new objects can be created with small differences to existing ones.

• objects, methods, instance, message passing, inheritance are some important properties provided by these particular languages

• Encapsulation, polymorphism, abstraction are also counts in these fundamentals of programming language.

• It implements real life scenario.

• In OOP, programmer not only defines data types but also deals with operations applied for data structures.

**Activity-1(Solved):**

Write a C++ class named as **Student** having data members ***rollno, grade*** and ***cgpa***. The class also consists of two functions **getData()** which takes input from user and **showData()** which displays data members on the console respectively. Create an object in main function and call both functions with the help of an object. Also add screenshot of the output.

**Solution:**

#include <iostream>

using namespace std;

class Student

{

//1. data members/attributes

//By default all members of the class are private

int rollno;

char grade;

float cgpa;

//2. Member functions/behaviors

public:

void getData()

{

cout<<"\nThis is getData function.";

cout<<"\nEnter roll no:";

cin>>rollno;

cout<<"Enter grade:";

cin>>grade;

cout<<"Enter cgpa:";

cin>>cgpa;

}

void showData()

{

cout<<"\nThis is showData function.";

cout<<endl<<rollno<<'\t'<<grade<<'\t'<<cgpa;

}

};

int main()

{

Student s1; //9 bytes of memory allocated

//s1.rollno=10; Error: cannot access private members of the class

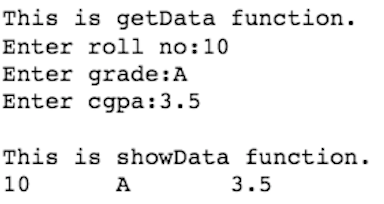
s1.getData();

s1.showData();

return 0;

}

**Output Screenshot:**

****

**Activity-2:**

Rewrite the program of Activity-1 and define all the functions of the class outside the class. You need to use class name and scope resolution operator Student:: between the return type and function name while defining functions outside the class. Declaration of functions within the class is compulsory. Also add screenshot of the output.

**Solution:**

**Output Screenshot:**

**Activity-3:**

In Activity-1, create an array of 3 Students in main function and take input from user for all the objects and then displays data of all the 3 students. Also add screenshot of the output.

**Solution:**

**Output Screenshot:**



**LAB # 02**

**Lab Objective:**

Objective of this lab is to make students understand the use of constructor, destructor and constructor overloading.

**Activity Outcomes:**

The student will understand the need and advantages constructor, destructor and constructor overloading.

**Instructor Note:**

The Students should have knowledge about function overloading.

**Introduction:**

Object-oriented programming, or OOP, is an approach to problem-solving where all computations are carried out using objects. An object is a component of a program that knows how to perform certain actions and how to interact with other elements of the program. Objects are the basic units of object-oriented programming. A simple example of an object would be a person. Logically, you would expect a person to have a name. This would be considered a property of the person. You would also expect a person to be able to do something, such as walking. This would be considered a method of the person.

A method in object-oriented programming is like a procedure in procedural programming. The key difference here is that the method is part of an object. In object-oriented programming, you organize your code by creating objects, and then you can give those objects properties and you can make them do certain things. Most modern programming languages including Java C++, C#, Python and PHP are Object Oriented.

One of the most important characteristics of procedural programming is that it relies on procedures that operate on data - these are two separate concepts. In object-oriented programming, these two concepts are bundled into objects. This makes it possible to create more complicated behavior with less code. The use of objects also makes it possible to reuse code. Once you have created an object with more complex behavior, you can use it anywhere in your code.

**Activity-1(Solved):**

Write a program to change the age of Ali and Ahmed, which are initialized in the default constructor (Constructor with no arguments) as 0 to 30 the first student and 29 for the second student respectively.

**Solution:**

#include<iostream>

using namespace std;

class age

{

int Ali;

int Ahmad;

public:

age()

{

Ali = 0;

Ahmad = 0;

}

void update\_age()

{

Ali += 30;

Ahmad += 29;

cout << "The updated age are " << endl;

cout << "For Ali : " << Ali << endl;

cout << "For Ahmad : " << Ahmad << endl;

}

void simple\_age()

{

cout << "The Simple ages are" << endl;

cout << "For Ali:0" << endl;

cout << "For Ahmad:0" << endl;

}

};

int main()

{

age a;

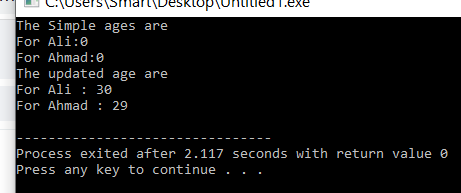
a.simple\_age();

a.update\_age();

return 0;

}

**Output Screenshot:**

****

**Activity-2:**

Create a class which have a member function write the definition of function outside the class and set the value of length by passing a parameter to that function.

**Solution:**

#include <iostream>

using namespace std;

class Line {

public:

void setLength( double len );

double getLength( void );

Line(); // This is the constructor

private:

double length;

};

Line::Line(void) {

cout << "Object is being created" << endl;

}

void Line::setLength( double len ) {

length = len;

}

double Line::getLength( void ) {

return length;

}

int main() {

Line line;

// set line length

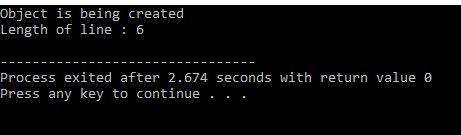
line.setLength(6.0);

cout << "Length of line : " << line.getLength() <<endl;

return 0;

}

**Output Screenshot:**

****

**Activity-3:**

Write a program that define a class with a data member to holds a serial number for each object created from the first object will be numbered 1 second 2 and on. To do this you’ll need another data member that records a count of how objects have been created so far. Then as each object is created its constructor can examine this count member variable to determine the appropriate serial number for the new object. Add a member function that primates an object to report its own serial number. Then write main() that creates three objects and quires each about its serial number.

**Solution:**

#include<iostream>

using namespace std;

class Serial

{

int serial\_number;

int count;

public:

Serial(int c)

{

count = c;

}

~Serial()

{

cout << "I am destructor of object" << count << endl;

}

void Serial\_Number()

{

cout << "Enter the Serial Number:";

cin>> serial\_number;

}

void MemFunct()

{

cout << "I am Objects Number:" << count << endl;

}

};

//int Serial::count = 0;

int main()

{

int count = 0;

Serial num(++count);

num.Serial\_Number();

num.MemFunct();

Serial num1(++count);

num1.Serial\_Number();

num1.MemFunct();

Serial num2(++count);

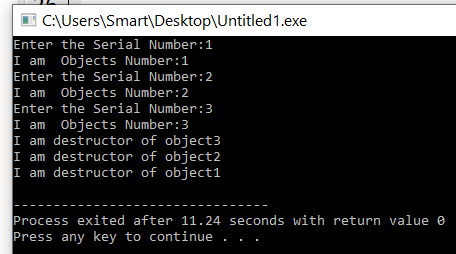
num2.Serial\_Number();

num2.MemFunct();

return 0;

}

**Output Screenshot:**

****

**Lab#03**

**Lab Objective:**

Objective of this lab is to make students understand the behavior of objects and memory allocation mechanism.

**Activity Outcomes:**

The memory allocation and de-allocation using constructors and destructors.

**Instructor Note:**

Memory size taken by each objects and data types.

**Introduction:**

Object-oriented programming, or OOP, is an approach to problem-solving where all computations are carried out using objects. An object is a component of a program that knows how to perform certain actions and how to interact with other elements of the program. Objects are the basic units of object-oriented programming. A simple example of an object would be a person. Logically, you would expect a person to have a name. This would be considered a property of the person. You would also expect a person to be able to do something, such as walking. This would be considered a method of the person.

A method in object-oriented programming is like a procedure in procedural programming. The key difference here is that the method is part of an object. In object-oriented programming, you organize your code by creating objects, and then you can give those objects properties and you can make them do certain things. Most modern programming languages including Java C++, C#, Python and PHP are Object Oriented.

One of the most important characteristics of procedural programming is that it relies on procedures that operate on data - these are two separate concepts. In object-oriented programming, these two concepts are bundled into objects. This makes it possible to create more complicated behavior with less code. The use of objects also makes it possible to reuse code. Once you have created an object with more complex behavior, you can use it anywhere in your code.

**Activity-1(Solved):**

Imagine a tollbooth at a bridge. Cars passing by the booth are expected to pay a 50 cent toll. Mostly they do, but sometimes a car goes by without paying. The tollbooth keeps track of the number of cars that have gone by, and of the total amount of money collected. Model this tollbooth with a class called tollBooth. The two data items are a type unsigned int to hold the total number of cars, and a type double to hold the total amount of money collected. A constructor initializes both of these to 0. A member function called payingCar() increments the car total and adds 0.50 to the cash total. Another function, called nopayCar(), increments the car total but adds nothing to the cash total. Finally, a member function called display() displays the two totals. Make appropriate member functions const. Include a program to test this class. This program should allow the user to push one key to count a paying car, and another to count a nonpaying car. Pushing the [Escape] key should cause the program to print out the total cars and total cash and then exit

**Solution:**

#include<iostream>

#include<conio.h>

using namespace std;

#define esc 27

class toolbooth

{

unsigned int Cno;

double tool;

unsigned int tpayed;

public:

toolbooth()

{

Cno = 0;

tool = 0;

tpayed = 0;

}

void payingCars()

{

Cno++;

tool += 0.50;

tpayed++;

cout << "car" << Cno << "Passed";

}

void noPayCars()

{

Cno++;

cout << "car" << Cno << "Passed" << endl;

}

void display()

{

cout << "\n\n Total Cars pass by :" << Cno;

cout << "\n\n Total tool paid:" << tool << "cents" << endl;

cout << "\n\nTotal cars that paid tool tax:" << tpayed << endl;

}

};

int main()

{

toolbooth t;

char ch;

cout << "Enter 1 to add car which payed tool tax!!" << endl;

cout<<"Enter 1 to add car which payed tool tax!!" << endl;

cout << "Enter Esc key to display total cars pass by and total tax payed !!" << endl;

do

{

ch = getch();

if (ch == '1')

t.payingCars();

else if (ch == '2')

t.noPayCars();

else

cout << "Wrong Input" << endl;

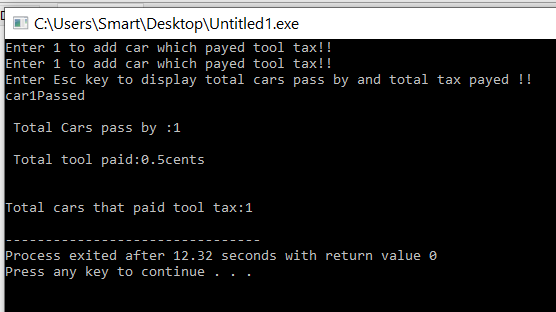
} while (ch == esc);

t.display();

return 0;

}

**Output Screenshot:**

****

**Activity-2:**

Create the number of objects and output which counts the total number of objects created by a single class.

**Solution:**

#include<iostream>

using namespace std;

class series

{

int count;

public:

void show()

{

cout << "The Number of series is : " << count++ << endl;

}

};

int main()

{

series s1, s2, s3, s4, s5;

s1.show();

s2.show();

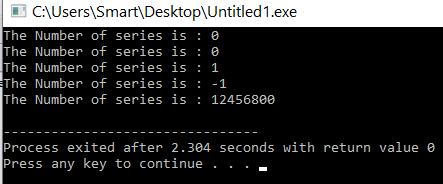
s3.show();

s4.show();

s5.show();

}

**Output Screenshot:**



**Activity-3:**

Write a c++ program which use two constructor one for initialized variable and second constructor should initialized that uninitialized variable.

**Solution:**

#include<iostream>

using namespace std;

class data

{

int a;

public:

data();

{

a = 0;

}

void add(int a, int b)

{

cout << "The Initialized value is:" << a << endl;

cout << "The unintialized value is :" << b << endl;

cout << "The sum of two unintialized value is :" << a + b << endl;

}

};

int main()

{

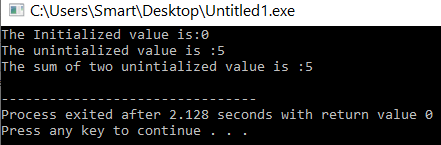
data D;

D.add(0, 5)

return 0;

}

**Output Screenshot:**



**Activity-4:**

Write class to test to check how many constructors and destructors have been generated.

**Solution:**

#include<iostream>

using namespace std;

class test

{

static int count;

public:

test()

{

count++;

}

~test()

{

cout << "The Object" << count << "Destroyed" << endl;

}

void show()

{

cout << "I am Number " << count << " Object" << endl;

}

};

int test::count = 0;

int main()

{

test t1;

t1.show();

test t2;

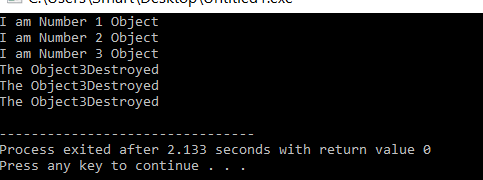
t2.show();

test t3;

t3.show();

return 0;}

**Output Screenshot:**





**LAB # 04**

**Lab Objective:**

1. Concept of Friend Functions and Classes
2. Practical walk through.
3. Exercises

**Activity Outcomes:**

The student will understand the advantages of classes and objects using friend function and friend class.

**Instructor Note:**

The student will understand about the friend functions and friend classes.

1. Programming Exercise for Friends Function
2. Programming Exercise for Friends Classes

**Introduction:**

**Classes and Objects**

A class is like a blueprint of data member and functions and object is an instance of class. For example, let’s say we have a class Car which has data members (variables) such as speed, weight, price and functions such as gearChange(), slowDown(), brake() etc. Now let’s say I create an object of this class named FordFigo which uses these data members and functions and give them its own values. Similarly, we can create as many objects as we want using the blueprint(class).

**Example**

class Car

{

//Data members

char name[20];

int speed;

int weight;

public:

//Functions

void brake(){

}

void slowDown(){

}

};

int main()

{

//ford is an object

Car ford;

}

**Constructor**

A class constructor is a special member function of a class that is executed whenever we create new objects of that class.

A constructor will have exact same name as the class and it does not have any return type at all, not even void. Constructors can be very useful for setting initial values for certain member variables.

**Activity 1-(Solved):**

**Example**

#include <iostream>

using namespace std;

class Line {

private:

double length;

public:

void setLength( double len );

double getLength( void );

Line(); // This is the constructor

};

// Member functions definitions including constructor

Line::Line(void) {

cout << "Object is being created" << endl;

}

void Line::setLength( double len ) {

length = len;

}

double Line::getLength( void ) {

return length;

}

// Main function for the program

int main() {

Line line;

// set line length

line.setLength(6.0);

cout << "Length of line : " << line.getLength() <<endl;

return 0;

}

**Parameterize Constructor**

A default constructor does not have any parameter, but if you need, a constructor can have parameters. This helps you to assign initial value to an object at the time of its creation as shown in the following example.

**Activity 2-(Solved):**

**Example**

// example: class constructor

#include <iostream>

using namespace std;

class Rectangle {

int width, height;

public:

Rectangle (int,int);

int area () {

return (width\*height);

}

};

Rectangle::Rectangle (int a, int b) {

width = a;

height = b;

}

int main () {

Rectangle rect (3,4);

Rectangle rectb (5,6);

cout << "rect area: " << rect.area() << endl;

cout << "rectb area: " << rectb.area() << endl;

return 0;

}

class Rectangle has no member function set\_values, and has instead a constructor that performs a similar action: it initializes the values of width and height with the arguments passed to it.

Notice how these arguments are passed to the constructor at the moment at which the objects of this class are created:

1. Rectangle rect (3,4);

**Friends function**

A friend function is a function that is specified outside a class but has the ability to access the class members’ [protected and private data](https://www.xspdf.com/resolution/59954476.html). A friend can be a member’s function, function template, or function, or a class or class template, in which case the entire class and all of its [members are friends](https://www.xspdf.com/resolution/10067734.html).

**Syntax of friend functions:**

class className {

    ... .. ...

    friend returnType functionName(arguments);

    ... .. ...

}

**Activity 3-(Solved):**

A simple example of a C++ friend function used to print the length of the box.

#include <iostream>

using namespace std;

class Box

{

   private:

        int length;

   public:

         Box (): length (0) {}

   friend int printLength (Box); //friend function

};

int printLength (Box b)

{

    b. length +=10;

    return b. length;

}

int main ()

{

   Box b;

   cout <<” Length of box:” <<printLength (b)<<endl;

    return 0;

}

**Activity-4**

**Write a C++ program that creates a class called laptop. The data members of the class are**

* brand (string)
* model(string)
* serial (int)
* color (string)
* price (float)
* processor speed (float)
* RAM (int)
* screen size(float).
* The constructor should accept the laptop brand, model, serial, color, price, processor speed, ram and screen size.
* Make a member function name display to display the value of all attributes.
* Make 3 objects of laptop and display its values.

**Solution:**

**Output Screenshot:**

**Activity-5:**

Write a C++ Program to illustrate default constructor, parameterized constructor and copy constructors.

**Solution:**

**Output Screenshot:**

**Activity-6:**

Write a Program to demonstrate friend function and friend class.

**Solution:**

**Output Screenshot:**



**LAB # 05**

**Lab Objective:**

**Activity Outcomes:**

1. What is Function Overriding
2. What is Inheritance
3. Types of Inheritance
4. Inheritance Practical Walk through
5. Exercises

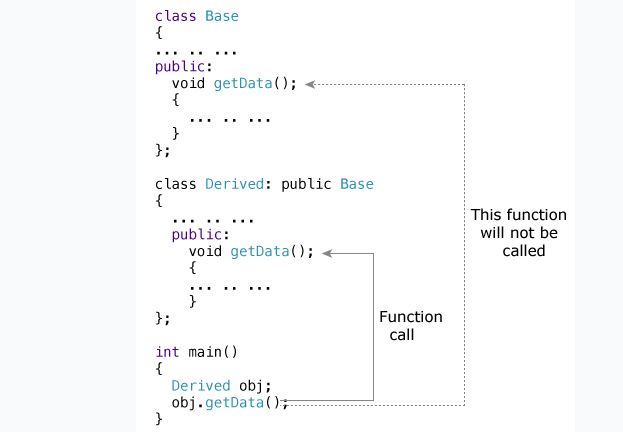
**Instructor Note:**

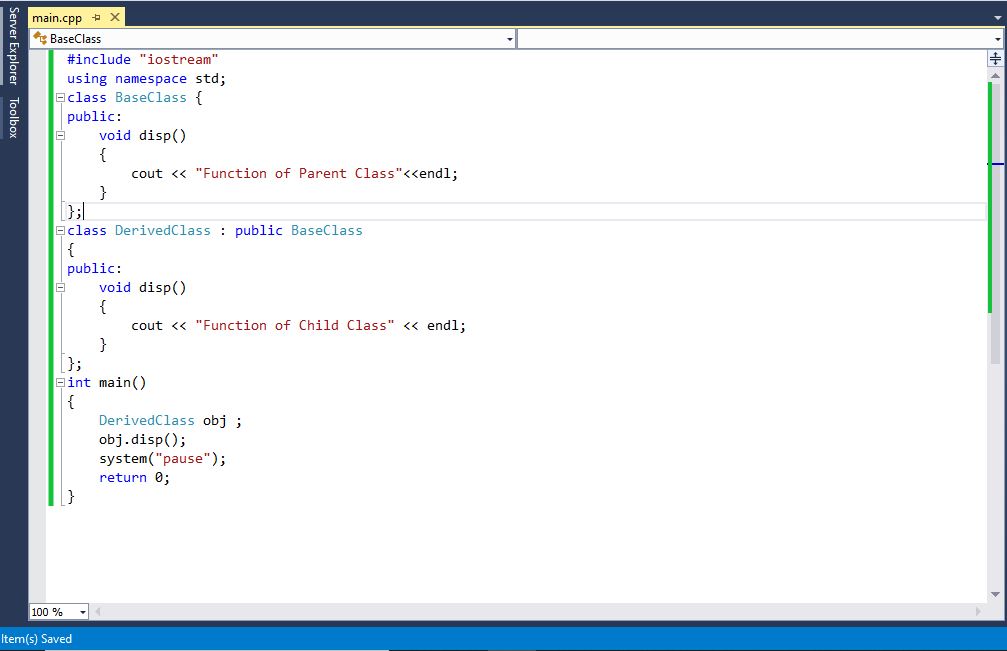
The Students should have knowledge about the function overriding concept and its working, inheritance in OOP, its type and working.

**Introduction:**

**Function is Overriding:**

Function overriding is a type of runtime polymorphism. When there are multiple functions with same name with same parameters then these functions are said to be **overridden**. If you create an object of the derived class and call the member function which exists in both classes (base and derived), the member function of the derived class is invoked and the function of the base class is ignored.





**Inheritance:**

Inheritance is a mechanism in which one object acquires all the properties and behaviors of a parent object. It is an important part of OOPs (Object Oriented programming system).

The idea behind inheritance in C++ is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of the parent class. Moreover, you can add new methods and fields in your current class also.

Inheritance represents the **IS-A relationship** which is also known as a *parent-child* relationship. Inheritance is a compile-time mechanism. A super-class can have any number of subclasses and a subclass can have more than one superclass in C++.

**Types of Inheritance:**

Following are the types of inheritance:

1. **Single Inheritance:**

In single inheritance, a class is allowed to inherit from only one class. i.e. one sub class is inherited by one base class only.

1. **Multiple Inheritance:**

 Multiple Inheritance is a feature of C++ where a class can inherit from more than one classes. i.e one **sub class** is inherited from more than one **base classes**.

1. **Multilevel Inheritance:**

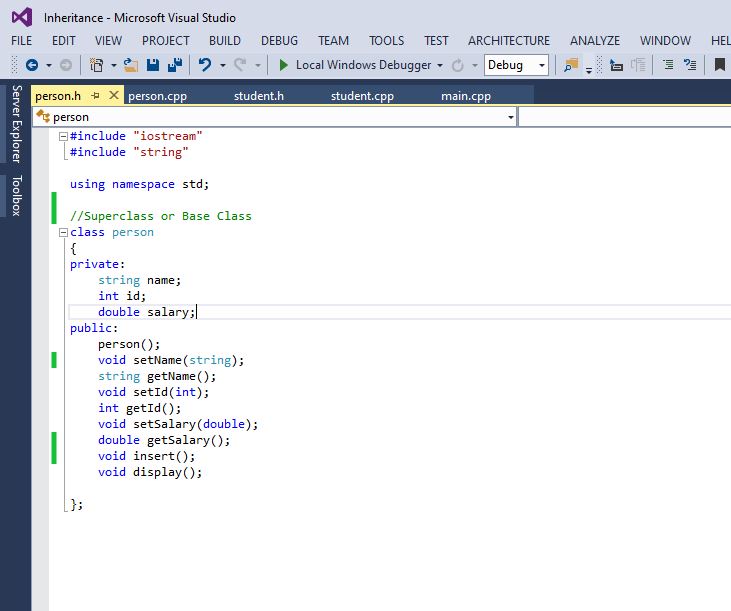
When a class extends a class, which extends anther class then this is called **multilevel inheritance**. this type of inheritance is known as multilevel inheritance.

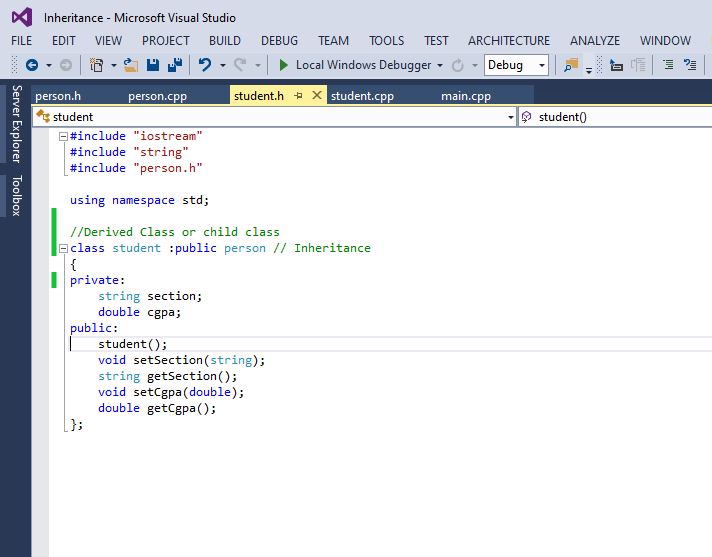
1. **Hierarchical Inheritance:**

When more than one classes inherit a same class then this is called hierarchical inheritance.

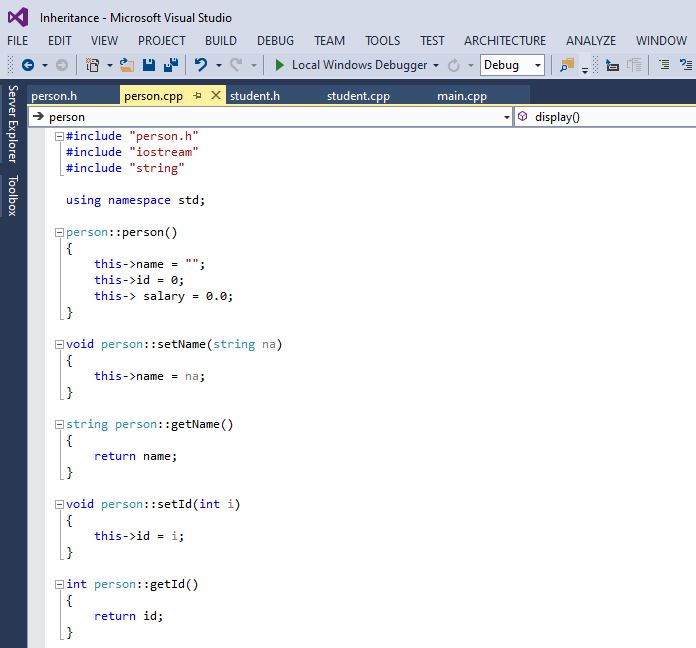
**Inheritance Practical Walk Through:**

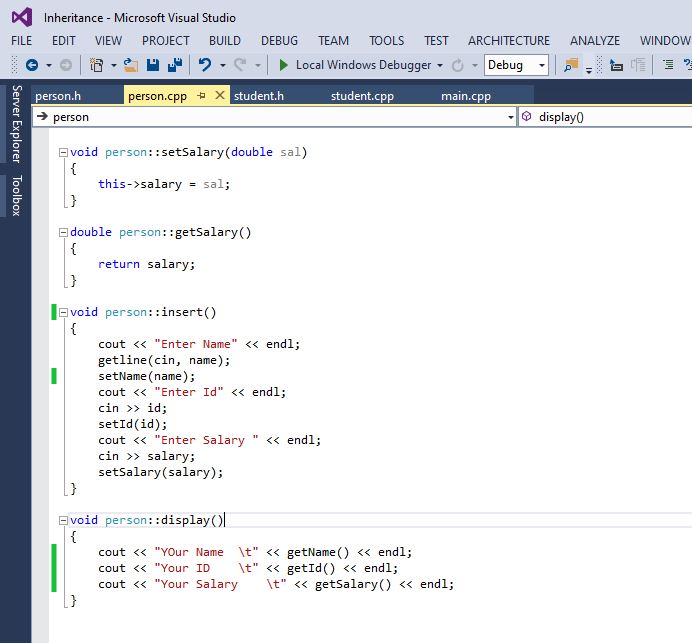
**Activity-1(Solved):**

**Person Class Header Code:**

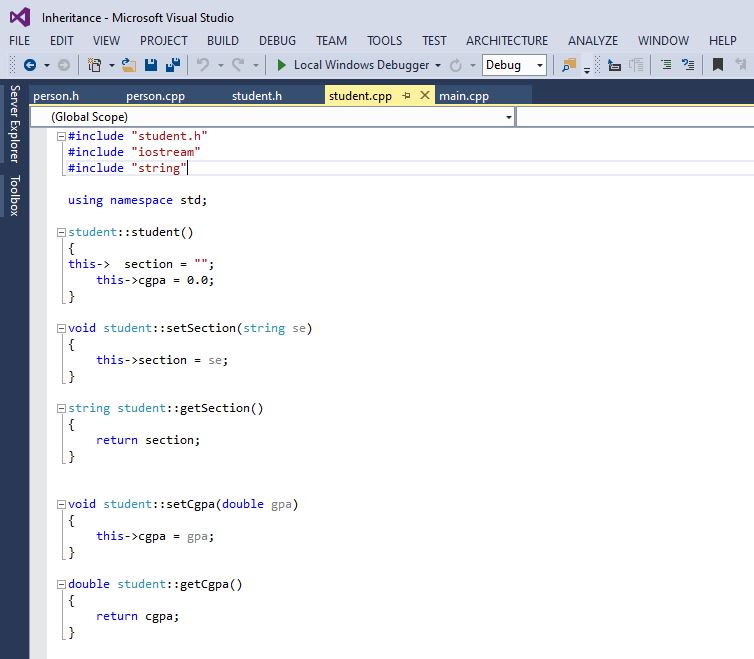
**Student Class Header Code:**

**Person Class Source Code:**

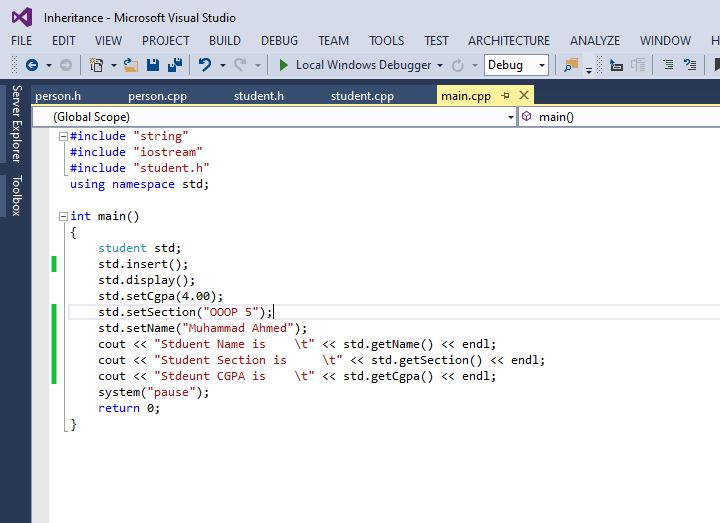




**Student Class Source Code:**



**Main Code:**



**Activity-2:**

Create a class named ***Person***, which contains

* A function named ***print()***
* Two data fields i.e. ***personName*** and ***age***

A class named ***Patient*** inherits ***Person*** class, which contains

* Two data fields i.e. ***diseaseType*** and ***recommendedMedicine***
* Overridden function ***print()*** to display all ***details*** relevant to a patient

A class named ***MedicarePatient*** inherited from class Patient, which holds

* A data field representing the ***name of the hospital***
* A data filed representing the ***name of the ward***
* A data field representing ***room number***
* Overridden function ***print()*** to display all details relevant to a patient

In the ***main*** function, create instances of derived classes to access respective ***print()*** function using ***dynamic binding.***

**Solution:**

**Output Screenshot:**

**Activity-3:**

Design a class named Person and its two subclasses named Student and Employee. Design two more classes; Faculty and Staff and inherit them from Employee. The detail of classes is as under:

* A person has a name, email address, phone number and address.
* A student has a status (String).
* An employee has an office, salary, and date hired
* A faculty member has office hours and rank
* A staff member has a title.
* Create display method in each class.

Also create getter, setter and default constructor in above task.

**Solution:**

**Output Screenshot:**

**Activity-4:**

Write a class Currency and makes the following attribute and member function.

* Currency name (String)
* Make a function name converter which converts the currency.
* Display function which display the value
* **Make 3 derived classes name Dollar, euro and yen from Class Currency.**
* In dollar class make a function converter and converts the minimum 50 dollars in rupees.
* In euro class make a function name converter which converts 1000 rupees into euro.
* User enter the amount less than 1000 display error message” invalid amount”

Also create getter, setter and default constructor in above task.

**Solution:**

**Output Screenshot:**

**Activity-5:**

Write a Program that implements Base class Bank account and a Derived class saving account and current account. Declare all the attributes as public, Show the concept of Multiple inheritance.

**Solution:**

**Output Screenshot:**

**Activity-6:**

Write a base class Computer that contains data members of word size (in bits), memory size (in megabytes), storage size (in megabytes) and speed (in megahertz). Derive a Laptop class that is a kind of computer but also specifies the object’s length, width, height, and weight. Member functions for both classes should include a default constructor, a function to display data members

**Solution:**

**Output Screenshot:**



**LAB # 06**

**Lab Objective:**

Objective of this lab is to understand the concepts of strings by using the built=in libraries as well as by defining own string class and its functions.

**Activity Outcomes:**

The student will understand the concepts of strings.

**Instructor Note:**

The Students should have knowledge about basic programming concepts, particularly loops and arrays

**Introduction:**

C++ is based on the OOPs concept; it enables you to represent the string as an object of the C++ String class (std:: string). The class allows you to declare a string variable quickly, and store any sequence of characters in it. C++ has in its definition a way to represent a sequence of characters as an object of the class. This class is called std:: string. String class stores the characters as a sequence of bytes with the functionality of allowing access to the single-byte character.

Here’s an example of representing a string with the help of the String class.

**Activity 1: implement the following functions**

1. **Input Functions:**

#include <iostream>

#include <string> // for string class

using namespace std;

// Driver Code

int main()

{

    // Declaring string

    string str;

    // Taking string input using getline()

    getline(cin, str);

    // Displaying string

    cout << "The initial string is : ";

    cout << str << endl;

    // Inserting a character

    str.push\_back('s');

    // Displaying string

    cout << "The string after push\_back operation is : ";

    cout << str << endl;

    // Deleting a character

    str.pop\_back();

    // Displaying string

    cout << "The string after pop\_back operation is : ";

    cout << str << endl;

    return 0;

}

Input

geeksforgeek

Output

The initial string is : geeksforgeek

The string after push\_back operation is : geeksforgeeks

The string after pop\_back operation is : geeksforgeek

1. **Capacity Functions:**

#include <iostream>

#include <string> // for string class

using namespace std;

// Driver Code

int main()

{

    // Initializing string

    string str = "geeksforgeeks is for geeks";

    // Displaying string

    cout << "The initial string is : ";

    cout << str << endl;

    // Resizing string using resize()

    str.resize(13);

    // Displaying string

    cout << "The string after resize operation is : ";

    cout << str << endl;

    // Displaying capacity of string

    cout << "The capacity of string is : ";

    cout << str.capacity() << endl;

    // Displaying length of the string

    cout << "The length of the string is :" << str.length()

         << endl;

    // Decreasing the capacity of string

    // using shrink\_to\_fit()

    str.shrink\_to\_fit();

    // Displaying string

    cout << "The new capacity after shrinking is : ";

    cout << str.capacity() << endl;

    return 0;

}

Output

The initial string is : geeksforgeeks is for geeks

The string after resize operation is : geeksforgeeks

The capacity of string is : 26

The length of the string is :13

The new capacity after shrinking is : 15

1. **Iterator Functions:**

#include <iostream>

#include <string> // for string class

using namespace std;

// Driver Code

int main()

{

    // Initializing string`

    string str = "geeksforgeeks";

    // Declaring iterator

    std::string::iterator it;

    // Declaring reverse iterator

    std::string::reverse\_iterator it1;

    // Displaying string

    cout << "The string using forward iterators is : ";

    for (it = str.begin(); it != str.end(); it++)

        cout << \*it;

    cout << endl;

    // Displaying reverse string

    cout << "The reverse string using reverse iterators is "

            ": ";

    for (it1 = str.rbegin(); it1 != str.rend(); it1++)

        cout << \*it1;

    cout << endl;

    return 0;

}

Output

The string using forward iterators is : geeksforgeeks

The reverse string using reverse iterators is : skeegrofskeeg

1. **Manipulating Functions:**

// C++ Program to demonstrate the working of

// copy() and swap()

#include <iostream>

#include <string> // for string class

using namespace std;

// Driver Code

int main()

{

// Initializing 1st string

string str1 = "geeksforgeeks is for geeks";

// Declaring 2nd string

string str2 = "geeksforgeeks rocks";

// Declaring character array

char ch[80];

// using copy() to copy elements into char array

// copies "geeksforgeeks"

str1.copy(ch, 13, 0);

// Displaying char array

cout << "The new copied character array is : ";

cout << ch << endl;

// Displaying strings before swapping

cout << "The 1st string before swapping is : ";

cout << str1 << endl;

cout << "The 2nd string before swapping is : ";

cout << str2 << endl;

// using swap() to swap string content

str1.swap(str2);

// Displaying strings after swapping

cout << "The 1st string after swapping is : ";

cout << str1 << endl;

cout << "The 2nd string after swapping is : ";

cout << str2 << endl;

return 0;

}

Output

The new copied character array is : geeksforgeeks

The 1st string before swapping is : geeksforgeeks is for geeks

The 2nd string before swapping is : geeksforgeeks rocks

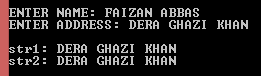
The 1st string after swapping is : geeksforgeeks rocks

The 2nd string after swapping is : geeksforgeeks is for geeks

**Activity 2: Defining Your Own string functions.**

**# 01 strcpy() //(Built In)**

char str1[50];

 char str2[20];

cout << "\n\nENTER NAME: ";

gets\_s(str1);

cout << "ENTER ADDRESS: ";

gets\_s(str2);

strcpy\_s(str1, str2);

cout << "\nstr1: " << str1 << endl;

cout << "str2: " << str2 << endl;

**P# 02 //User Defined**

char str1[50];

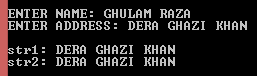
char str2[20];

cout << "\n\nENTER NAME: ";

gets\_s(str1);

cout << "ENTER ADDRESS: ";

gets\_s(str2);

 int i = 0;

while (str2[i]!='\0')

{

str1[i] = str2[i];

i++;

}

str1[i] = '\0';

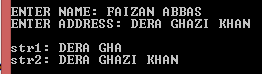
cout << "\nstr1: " << str1 << endl;

cout << "str2: " << str2 << endl;

**P# 03 strncpy() //Built In**

char str1[50];

char str2[20];

 cout << "\n\nENTER NAME: ";

gets\_s(str1);

cout << "ENTER ADDRESS: ";

gets\_s(str2);

strncpy\_s(str1, str2, 8);

cout << "\nstr1: " << str1 << endl;

cout << "str2: " << str2 << endl;

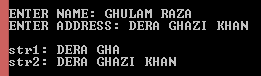
**P# 04 //user defined**

char str1[50];

char str2[20];

cout << "\n\nENTER NAME: ";

gets\_s(str1);

 cout << "ENTER ADDRESS: ";

gets\_s(str2);

int i = 0;

while (str2[i] != str2[8])

{

str1[i] = str2[i];

i++;

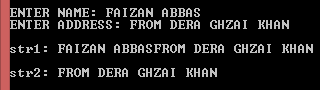
}

str1[i] = '\0';

cout << "\nstr1: " << str1 << endl;

cout << "str2: " << str2 << endl;

**P# 05 strcat() //Built In**

char str1[50];

char str2[22];

cout << "\n\nENTER NAME: ";

gets\_s(str1);

cout << "ENTER ADDRESS: ";

gets\_s(str2);

strcat\_s(str1, str2);

cout << "\nstr1: " << str1 << endl;

cout << "\nstr2: " << str2 << endl;

**Activity 2: do user defined implementation**

**Solution: P# 06 //User Defined**

char str1[50];

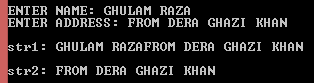
char str2[22];

cout << "\n\nENTER NAME: ";

gets\_s(str1);

cout << "ENTER ADDRESS: ";

gets\_s(str2);

 int i , j = 0;

for (i = 0; str1[i] != '\0'; i++)

{}

while (str2[j] != '\0')

{

str1[i + j] = str2[j];

j++;

}

str1[i + j] = '\0';

cout << "\nstr1: " << str1 << endl;

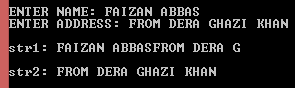
cout << "\nstr2: " << str2 << endl;

**P# 07 strncat() //(Built In)**

char str1[50];

char str2[22];

cout << "\n\nENTER NAME: ";

 gets\_s(str1);

cout << "ENTER ADDRESS: ";

gets\_s(str2);

strncat\_s(str1, str2, 11);

cout << "\nstr1: " << str1 << endl;

cout << "\nstr2: " << str2 << endl;

**P# 08 //User Defined**

char str1[50];

char str2[22];

cout << "\n\nENTER NAME: ";

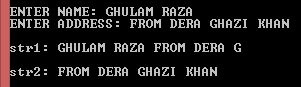
gets\_s(str1);

cout << "ENTER ADDRESS: ";

gets\_s(str2);

int i , j;

for (i = 0; str1[i] != '\0'; i++)

 {}

str1[i++]=' ';

j = 0;

while (str2[j] != str2[11])

{

str1[i + j] = str2[j];

j++;

}

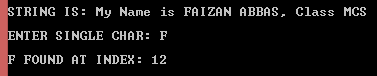
str1[i + j] = '\0';

cout << "\nstr1: " << str1 << endl;

cout << "\nstr2: " << str2 << endl;

**P# 09 strchr() 1st Occurrence in String //(Built In)**

char str1[]="My Name is FAIZAN ABBAS, Class MCS";

 char \*res,ch;

cout << "\nSTRING IS: "<< str1<<endl;

cout << "\nENTER SINGLE CHAR: ";

cin >> ch;

res = strchr(str1, ch);

if (res != NULL)

cout << "\n" << ch << " FOUND AT INDEX: " << res - str1 + 1 << endl;

else

cout << "\n" << ch << " NOT FOUND...!! " << endl;

**P# 10 //User Defined**

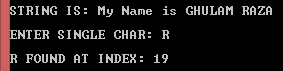
char str1[]="My Name is GHULAM RAZA";

char ch;

cout << "\nSTRING IS: " << str1 << endl;

cout << "\nENTER SINGLE CHAR: ";

cin >> ch;

 int res=-1,i = 0;

while (str1[i] != '\0')

{

if (str1[i] == ch)

{

res = i;

break;

}

else

i++;

}

if (res != -1)

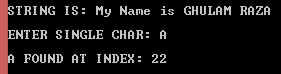
cout << "\n" << ch << " FOUND AT INDEX: " << res+1 << endl;

else

cout << "\n" << ch << " NOT FOUND...!! " << endl;

**P# 11 strrchr() Last Occurrence in String //(Built In)**

char str[]="My Name is GHULAM RAZA";

 char \*res,ch;

cout << "\nSTRING IS: " << str1 << endl;

cout << "\nENTER SINGLE CHAR: ";

cin >> ch;

res = strrchr(str, ch);

if (res != NULL)

cout << "\n" << ch << " FOUND AT INDEX: " << res-str+1 << endl;

else

cout << "\n" << ch << " NOT FOUND...!! " << endl;

**P# 12 //User Defined**

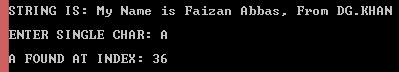
char str1[]="My Name is Faizan Abbas, From DG.KHAN ";

char ch;

cout << "\nSTRING IS: " << str1 << endl;

cout << "\nENTER SINGLE CHAR: ";

cin >> ch;

 int res=-1,i = 0;

while (str1[i] != '\0')

{

if (str1[i] == ch)

{

res = i;

i++;

}

else

i++;

}

if (res != -1)

cout << "\n" << ch << " FOUND AT INDEX: " << res+1 << endl;

else

cout << "\n" << ch << " NOT FOUND...!! " << endl;

**P# 13 strrev()**

char str[50];

cout << "\nENTER STRING: ";

gets\_s(str);

cout << "\nREVERS ORDER: " << \_strrev(str);

**Activity 3: do user defined implementation**

**Solution: P# 14 strrev()**

string name;

cout << "\nENTER YOUR NAME: ";

 getline(cin, name);

reverse(name.begin(), name.end());

cout << "\nNAME REVERSED IS: " << name << endl;

**P# 15 //User Defined**

string str;

cout << "ENTER STRING: ";

getline(cin,str);



cout << "REVERSE ORDER: ";

for (int i = str.length() - 1; i >= 0; i--)

{

cout << str[i];

}

//system("pause");

**P# 16 //User Defined**

char str[50];

int temp;

cout << "\nENTER STRING: ";

cin.getline(str, 50);



for (temp = 0; str[temp] != '\0'; temp++)

{}

cout << "\nREVERSE ORDER: ";

int j = temp;

while (j >= 0)

{

cout << str[j];

j--;

}

**P# 17 strlen() //User Defined**

char str[50];

cout << "\nENTER STRING: ";

 cin.getline(str, 50);

int i = 0;

while (str[i] != '\0')

{

i++;

}

cout << "\nLENGTH OF STRING: "<<i;

**P# 18 strlen()**

char str[50];

cout << "\nENTER STRING: ";

cin.getline(str, 50);

cout << "\nLENGTH OF STRING: " << strlen(str);

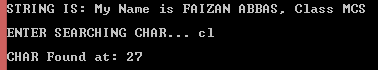
**P# 19 strcspn()**

int res;

char str1[] = "My Name is FAIZAN ABBAS, Class MCS";

char str2[10];

cout << "\nSTRING IS: " << str1 << endl;

 cout << "\nENTER SEARCHING CHAR...";

gets\_s(str2);

res = strcspn(str1, str2);

if (res != strlen(str1))

cout << "\nCHAR Found at: " << res+1 << endl;

else

cout << "\nCHAR not Found...!!" << endl;

**P# 20 //User Defined**

int res=-1,i,j;

char str1[] = "My Name is FAIZAN ABBAS, Class MCS";

char str2[10];

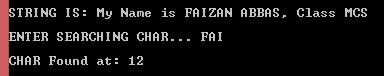
cout << "\nSTRING IS: " << str1 << endl;

cout << "\nENTER SEARCHING CHAR... ";

gets\_s(str2);

for (i = 0; str2[i] != '\0'; i++)

{

 j = 0;

while (str1[j] != '\0')

{

if (str2[i] == str1[j])

{

res = j;

goto end;

}

j++;

}

}

end:

if (res != -1)

cout << "\nCHAR Found at: " << res+1 << endl;

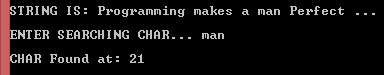
else

cout << "\nCHAR not Found...!!" << endl;

**P# 21 strstr()**

char \*res;

char str1[] = "Programming makes a man Perfect ...";

 char str2[10];

cout << "\nSTRING IS: "<<str1<<endl;

cout << "\nENTER SEARCHING CHAR... ";

gets\_s(str2);

res = strstr(str1, str2);

if (res != NULL)

cout << "\nCHAR Found at: " << res-str1+1 << endl;

else

cout << "\nCHAR not Found...!!" << endl;

**P# 22 //User Defined**

int res=0;

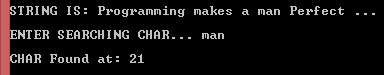
char str1[1][50] = { "Programming makes a man Perfect ..." };

char str2[1][10];

cout << "\nSTRING IS: " << str1[0] << endl;

cout << "\nENTER SEARCHING CHAR... ";

gets\_s(str2[0]);



int i, c, j;

for (c = 0; str2[0][c]; c++)

{}

i = 0;

while (str1[0][i] != '\0')

{

if (str1[0][i] == str2[0][0])

{

res++;

for (j = 1; j < c;j++)

if (str1[0][i + j] == str2[0][j])

{

res++;

}

if (res == c)

goto end;

else

res = 0;

}

i++;

}

end:

if (res == c)

cout << "\nCHAR Found at: " << i+1 << endl;

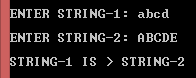
else

cout << "\nCHAR not Found...!!" << endl;

**P# 23 strcmp() add ascii code char by char case-sen**

int res;

char str1[50];

 char str2[10];

cout << "\nENTER STRING-1: ";

gets\_s(str1);

cout << "\nENTER STRING-2: ";

gets\_s(str2);

res = strcmp(str1, str2);

if (res < 0 )

cout << "\nSTRING-1 IS < STRING-2 "<< endl;

else if (res == 0)

cout << "\nSTRING-1 IS = STRING-2 " << endl;

else

cout << "\nSTRING-1 IS > STRING-2 " << endl;

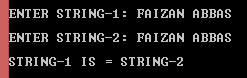
**P# 24 //User Defined**

int res,i,s1=0,s2=0;

char str1[50];

char str2[50];

cout << "\nENTER STRING-1: ";

 gets\_s(str1);

cout << "\nENTER STRING-2: ";

gets\_s(str2);

for (i = 0; str1[i] != '\0'; i++)

{

res = str1[i];

s1 += res;

}

for (i = 0; str2[i] != '\0'; i++)

{

res = str2[i];

s2 += res;

}

if (s1 < s2 )

cout << "\nSTRING-1 IS < STRING-2 "<< endl;

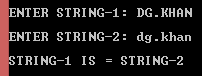
else if (s1 == s2)

cout << "\nSTRING-1 IS = STRING-2 " << endl;

else

cout << "\nSTRING-1 IS > STRING-2 " << endl;

**P# 25 stricmp() no-case-sen**



int res;

char str1[50];

char str2[50];

cout << "\nENTER STRING-1: ";

gets\_s(str1);

cout << "\nENTER STRING-2: ";

gets\_s(str2);

res = \_stricmp(str1, str2);

if (res == 0)

cout << "\nSTRING-1 IS = STRING-2 " << endl;

else if (res < 0 )

cout << "\nSTRING-1 IS < STRING-2 "<< endl;

else

cout << "\nSTRING-1 IS > STRING-2 " << endl;

**P# 26 stricmp() //User Defined no-case-sen**

int res,i,s1=0,s2=0;

char str1[50];

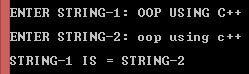
char str2[50];

cout << "\nENTER STRING-1: ";

gets\_s(str1);

cout << "\nENTER STRING-2: ";

gets\_s(str2);

 for (i = 0; str1[i] != '\0'; i++)

{

if (str1[i]>+65 && str1[i] <= 90)

{

res = str1[i];

s1 += (res + 32);

}

else

{

res = str1[i];

s1 += res;

}

}

for (i = 0; str2[i] != '\0'; i++)

{

if (str2[i]>+65 && str2[i] <= 90)

{

res = str2[i];

s2 += (res + 32);

}

else

{

res = str2[i];

s2 += res;

}

}

if (s1 == s2)

cout << "\nSTRING-1 IS = STRING-2 " << endl;

else if (s1 < s2 )

cout << "\nSTRING-1 IS < STRING-2 "<< endl;

else

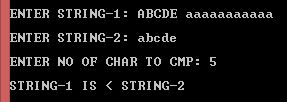
cout << "\nSTRING-1 IS > STRING-2 " << endl;

**P# 27 strncmp() case-sen**

int res, n;

char str1[50];

char str2[50];

 cout << "\nENTER STRING-1: ";

gets\_s(str1);

cout << "\nENTER STRING-2: ";

gets\_s(str2);

cout << "\nENTER NO OF CHAR TO CMP: ";

cin >> n;

res = strncmp(str1, str2,n);

if (res == 0)

cout << "\nSTRING-1 IS = STRING-2 " << endl;

else if (res < 0 )

cout << "\nSTRING-1 IS < STRING-2 "<< endl;

else

cout << "\nSTRING-1 IS > STRING-2 " << endl;

**P# 28 //User Defined case-sen**

int res,n,i,s1=0,s2=0;

char str1[50];

char str2[50];

cout << "\nENTER STRING-1: ";

gets\_s(str1);

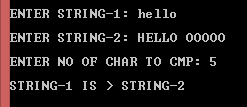
cout << "\nENTER STRING-2: ";

gets\_s(str2);

cout << "\nENTER NO OF CHAR TO CMP: ";

cin >> n;

for (i = 0; i<5; i++)

 {

res = str1[i];

s1 += res;

}

for (i = 0; i<5; i++)

{

res = str2[i];

s2 += res;

}

if (s1 > s2)

cout << "\nSTRING-1 IS > STRING-2 " << endl;

else if (s1 == s2)

cout << "\nSTRING-1 IS = STRING-2 " << endl;

else

cout << "\nSTRING-1 IS < STRING-2 " << endl;

**P# 29 strlwr()**

char str1[50];

cout << "\nENTER STRING: ";

gets\_s(str1);

\_strlwr\_s(str1);

cout << "\nSTRING Lower Case: " << str1 << endl;

**P# 30 //User Defined**

char str1[50];

cout << "\nENTER STRING: ";

gets\_s(str1);

int i = 0;

while (str1[i] != '\0')

 {

if (str1[i] > +65 && str1[i] <= 90)

{

str1[i] += 32;

}

i++;

}

cout << "\nLOWER CASE STRING: " << str1;

**P# 31 strupr()**

char str1[50];

 cout << "\nENTER STRING: ";

gets\_s(str1);

\_strupr\_s(str1);

cout << "\nUPPER CASE STRING: " << str1;

**P# 32 //User Defined**

char str1[50];

cout << "\nENTER STRING: ";

 gets\_s(str1);

int i = 0;

while (str1[i] != '\0')

{

if (str1[i] >= 97 && str1[i] <= 122)

{

str1[i] -= 32;

}

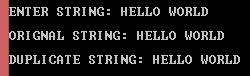
i++;

}

cout << "\nUPPER CASE STRING: " << str1;

**P# 33 strdup()**

char str1[50];

 char \*dup;

cout << "\nENTER STRING: ";

gets\_s(str1);

cout << "\nORIGNAL STRING: " << str1 << endl;

dup=\_strdup(str1);

cout << "\nDUPLICATE STRING: " << dup;

**P# 34 //User Defined**

char str1[15]="FAIZAN ABBAS";

char \*dup;

cout << "\nORIGNAL STRING: " << str1 << endl;

dup = str1;

cout << "\nDUPLICATE STRING: " << dup;

**P# 35 strset()**

char str1[15]="FAIZAN ABBAS";

char ch='\*';



cout << "\nORIGNAL STRING: " << str1 << endl;

\_strset\_s(str1,ch);

cout << "\nAFTER APPLY \* : " << str1;

**P# 36 //User Defined**

char str1[15];

char ch='\*';

cout << "\nENTER STRING: ";

 gets\_s(str1);

int i, n = strlen(str1);

i = 0;

cout << "\nAFTER APPLY \*: ";

while (i < n)

{

cout << ch;

i++;

}

**P# 37 strnset()**

char str1[15]="HELLO WORLD...";

char ch='@';

cout << "\nORIGNAL STRING: " << str1 << endl;

\_strnset\_s(str1, ch, 7);

cout << "\nAFTER APPLY @ : " << str1;

**P# 38 //User Defined**

int n,i,j;

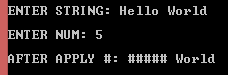
char str1[15];

char ch='#';

cout << "\nENTER STRING: ";

gets\_s(str1);

cout << "\nENTER NUM: ";

 cin >> n;

i = 0;

cout << "\nAFTER APPLY #: ";

while (i < n)

{

cout << ch;

i++;

}

j = 0;

while (str1[i+j]!='\0')

{

cout << str1[i + j];

j++;

}

**P# 39 var.swap()**

string str1 = "FAIZAN";

string str2 = "ABBAS";

cout << "\nstr1: " << str1 << endl;

cout << "str2: " << str2 << endl;

str1.swap(str2);

cout << "\nstr1 is now: " << str1 << endl;

cout << "str2 is now: " << str2 << endl;

**P# 40 var.insert()**

string str1="FAIZAN ABBAS DERA GHAZI KHAN";

cout << "\nORIGNAL STRING: " << str1 << endl;

str1.insert(13, "FROM ");

cout << "\nNew STRING: " << str1 << endl;



**P# 41 var.replace()**

string str1="FAIZAN ABBAS (frm) DERA GHAZI KHAN";

cout << "\nORIGNAL STRING: " << str1 << endl;

str1.replace(13, 5, "FROM");

cout << "\nNEW STRING: " << str1 << endl;



**P# 42 var.append()**

string str1="FAIZAN ABBAS, 13-ARID-1416";

cout << "\nORIGNAL STRING: " << str1 << endl;

str1.append(", CLASS MCS");

cout << "\nNEW STRING: " << str1 << endl;



**P# 43 var.empty()**

string str1("PAKISTAN");

cout << "\nSTRING: " << str1 << endl;



if (str1.empty())

cout << "\nString is Empty" << endl;

else

cout << "\nString is Not Empty" << endl;

**P# 43 var.length()**

**var.size()**



string str1("PAKISTAN");

cout << "\nSTRING: " << str1 << endl;

cout << "\nSTR LENGTH: " << str1.length() << endl;

**NOTE: All Programs are properly working on MS Visual Studio 2013.**



**LAB # 07**

**Lab Objective:**

Objective of this lab is to understand the concepts of Operator overloading, unary and binary operator overloading, particularly the rational behind ().[] and Assignment operator overloading.

**Activity Outcomes:**

The student will understand that how to overload all possible operators in C++

**Instructor Note:**

The Students should have knowledge about function overloading and argument passing and returning.

**Introduction:**

One of the Object-oriented programming advantage is to overload operators for the developer’s ease. Instead of functions names we use operators to easily understand the functionality. We are able to write function calls in the form of mathematical equations.

**Activity `1: Unary Operator Overloading**

// countpp1.cpp

// increment counter variable with ++ operator

#include <iostream>

using namespace std;

class Counter

{

private:

unsigned int count; //count

public:

Counter() : count(0) //constructor

{ }

unsigned int get\_count() //return count

{

return count;

}

void operator ++ () //increment (prefix)

{

++count;

}

};

int main()

{

Counter c1, c2; //define and initialize

cout << "\nc1 = " << c1.get\_count(); //display

cout << "\nc2 = " << c2.get\_count();

++c1; //increment c1

++c2; //increment c2

++c2; //increment c2

cout << "\nc1 = " << c1.get\_count(); //display again

cout << "\nc2 = " << c2.get\_count() << endl;

}

**Activity 2: Define a counter that counts the number of student while entering or leaving the lab. On entry the count should get incremented and on exit should be decremented. Total number of students present in the lab can also be displayed.**

**Activity 3: Binary/Arithematic Operator Overloading**

// englplus.cpp

// overloaded ‘+’ operator adds two Distances

#include <iostream>

using namespace std;

class Distance //English Distance class

{

private:

int feet;

float inches;

public: //constructor (no args)

Distance() : feet(0), inches(0.0)

{ } //constructor (two args)

Distance(int ft, float in) : feet(ft), inches(in)

{ }

void getdist() //get length from user

{

cout << "\nEnter feet : ";

cin >> feet;

cout << "Enter inches : ";

cin >> inches;

}

void showdist() const //display distance

{

cout << feet << "\' - " << inches << "\"";

}

Distance operator + (Distance) const; //add 2 distances

};

//--------------------------------------------------------------

//add this distance to d2

Distance Distance::operator + (Distance d2) const //return sum

{

int f = feet + d2.feet; //add the feet

float i = inches + d2.inches; //add the inches

if (i >= 12.0) //if total exceeds 12.0,

{ //then decrease inches

i -= 12.0; //by 12.0 and

f++; //increase feet by 1

} //return a temporary Distance

return Distance(f, i); //initialized to sum

}

int main()

{

Distance dist1, dist3, dist4; //define distances

dist1.getdist(); //get dist1 from user

Distance dist2(11, 6.25); //define, initialize dist2

dist3 = dist1 + dist2; //single ‘+’ operator

dist4 = dist1 + dist2 + dist3; //multiple ‘+’ operators

//display all lengths

cout << "dist1 = "; dist1.showdist(); cout << endl;

cout << "dist2 = "; dist2.showdist(); cout << endl;

cout << "dist3 = "; dist3.showdist(); cout << endl;

cout << "dist4 = "; dist4.showdist(); cout << endl;

return 0;

}

**Activity 4: Binary/Arithematic Operator Overloading**

**Modify the above program to subtract a smaller distance from the larger one using – operator.**

**Activity 5: Multiple Operator Overloading**

// engless.cpp

// overloaded ‘<’ operator compares two Distances

#include <iostream>

using namespace std;

class Distance //English Distance class

{

private:

int feet;

float inches;

public: //constructor (no args)

Distance() : feet(0), inches(0.0)

{ } //constructor (two args)

Distance(int ft, float in) : feet(ft), inches(in)

{ }

void getdist() //get length from user

{

cout << "\nEnter feet : ";

cin >> feet;

cout << "Enter inches : ";

cin >> inches;

}

void showdist() const //display distance

{

cout << feet << "\' - " << inches << "\"";

}

bool operator < (Distance) const; //compare distances

};

//compare this distance with d2

bool Distance::operator < (Distance d2) const //return the sum

{

float bf1 = feet + inches / 12;

float bf2 = d2.feet + d2.inches / 12;

return (bf1 < bf2) ? true : false;

}

int main()

{

Distance dist1; //define Distance dist1

dist1.getdist(); //get dist1 from user

Distance dist2(6, 2.5); //define and initialize dist2

//display distances

cout << "\ndist1 = ";

dist1.showdist();

cout << "\ndist2 = ";

dist2.showdist();

if (dist1 < dist2) //overloaded ‘<’ operator

cout << "\ndist1 is less than dist2";

else

cout << "\ndist1 is greater than(or equal to) dist2";

cout << endl;

return 0;

}

**Activity 6: Overloaded [] Operator**

#include <iostream>

using namespace std;

#include <process.h> //for exit()

const int LIMIT = 100; //array size

class safearay

{

private:

int arr[LIMIT];

public:

int& operator [](int n) //note: return by reference

{

if (n < 0 || n >= LIMIT)

{

cout << "\nIndex out of bounds";

exit(1);

}

return arr[n];

}

};

int main()

{

safearay sa1;

for (int j = 0; j < LIMIT; j++) //insert elements

sa1[j] = j \* 10; //\*left\* side of equal sign

for (int j = 0; j < LIMIT; j++) //display elements

{

int temp = sa1[j]; //\*right\* side of equal signa

cout << "Element " << j << " is " << temp << endl;

}

}

**Activity 7: () Operator overloaded**

#include <iostream>

using namespace std;

#include <process.h> //for exit()

const int LIMIT = 100; //array size

class safearay

{

private:

int arr[LIMIT];

public:

int& access(int n) //note: return by reference

{

if (n < 0 || n >= LIMIT)

{

cout << "\nIndex out of bounds";

exit(1);

}

return arr[n];

}

};

int main()

{

safearay sa1;

for (int j = 0; j < LIMIT; j++) //insert elements

sa1.access(j) = j \* 10; //\*left\* side of equal sign

for (int j = 0; j < LIMIT; j++) //display elements

{

int temp = sa1.access(j); //\*right\* side of equal sign

cout << "Element " << j << " is " << temp << endl;

}

}



**LAB # 08**

**Lab Objective:**

Objective of this lab is to make students understand the use of virtual functions while using class.

**Activity Outcomes:**

The student will understand use of virtual functions while applying inheritance concepts.

**Instructor Note:**

The Students should have knowledge about C++ compiler, variables and functions, classes, parent child relationship.

**Introduction:**

One of the most important concepts in object-oriented programming is that of inheritance. Inheritance allows us to define a class in terms of another class, which makes it easier to create and maintain an application. This also provides an opportunity to reuse the code functionality and fast implementation time.

When creating a class, instead of writing completely new data members and member functions, the programmer can designate that the new class should inherit the members of an existing class. This existing class is called the base class, and the new class is referred to as the derived class.

A virtual function is a member function which is declared within a base class and is re-defined (overridden) by a derived class. When you refer to a derived class object using a pointer or a reference to the base class, you can call a virtual function for that object and execute the derived class’s version of the function.

* Virtual functions ensure that the correct function is called for an object, regardless of the type of reference (or pointer) used for function call.
* They are mainly used to achieve Runtime polymorphism
* Functions are declared with a virtual keyword in base class.
* The resolving of function call is done at runtime.

Data abstraction is one of the most essential and important feature of object oriented programming in C++. Abstraction means displaying only essential information and hiding the details. Data abstraction refers to providing only essential information about the data to the outside world, hiding the background details or implementation.

**Abstraction using Classes:** We can implement Abstraction in C++ using classes. Class helps us to group data members and member functions using available access specifiers. A Class can decide which data member will be visible to outside world and which is not.

**Advantages of Data Abstraction**:

* Helps the user to avoid writing the low level code
* Avoids code duplication and increases reusability.
* Can change internal implementation of class independently without affecting the user.
* Helps to increase security of an application or program as only important details are provided to the user.

**Activity-1(Solved):**

Write a C++ program that uses show function as a virtual function in inherited classes. The parent class A should have a virtual function show(), Child class B will inherit it. Check effect at run time using dynamic object association.

**Solution:**

#include <iostream>

using namespace std;

class A

{

private:

int a;

public:

A()

{

a = 1;

}

virtual void show()

{

cout<<"Value is="<<a;

}

};

class B: public A

{

private:

int b;

public:

B()

{

b = 2;

}

virtual void show()

{

cout<<"Value is="<<b;

}

};

int main()

{

A \*pA;

B oB;

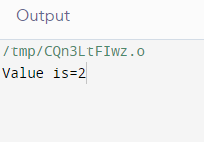
pA = &oB;

pA->show();

return 0;

}

**Output Screenshot:**



**Activity-2:**

Write a C++ program that uses show function as a virtual function in inherited classes. Tha parent class A should have a virtual function show(), Child class B will inherit it. Check effect at run time using different run time object assignments.

**Solution:**

#include <iostream>

using namespace std;

class A

{

public:

virtual void show()

{

cout<<"Parent Class A………"<<endl;

}

};

class B: public A

{

public:

void show()

{

cout <<"Child Class B………"<<endl;

}

};

int main()

{

A obj1;

B obj2;

A \*ptr;

ptr = &obj1;

ptr->show();

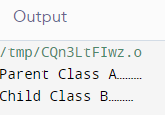
ptr = &obj2;

ptr->show();

return 0;

}

**Output Screenshot:**



**Activity-3:**

Write a C++ program that uses area function as a virtual function in inherited classes. The parent class Shape should have a virtual function area (), Child class Rectangle will inherit it. Check effect of child class object at run time while making parent class as an Abstract class.

**Solution:**

#include <iostream>

using namespace std;

class Shape {

public:

virtual int Area() = 0; // Pure virtual function is declared as follows.

// Function to set width.

void setWidth(int w) {

width = w;

}

// Function to set height.

void setHeight(int h) {

height = h;

}

protected:

int width;

int height;

};

// A rectangle is a shape; it inherits shape.

class Rectangle: public Shape {

public:

// The implementation for Area is specific to a rectangle.

int Area() {

return (width \* height);

}

};

// A triangle is a shape too; it inherits shape.

class Triangle: public Shape {

public:

// Triangle uses the same Area function but implements it to

// return the area of a triangle.

int Area() {

return (width \* height)/2;

}

};

int main() {

Rectangle R;

Triangle T;

R.setWidth(5);

R.setHeight(10);

T.setWidth(20);

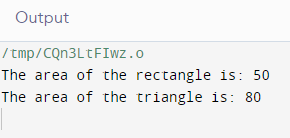
T.setHeight(8);

cout << "The area of the rectangle is: " << R.Area() << endl;

cout << "The area of the triangle is: " << T.Area() << endl;

}

**Output Screenshot:**



**Activity-4:**

Write a C++ program having a base class and a derived class. Make a virtual function and execute it in main class with child class object.

**Solution:**

#include<iostream>

using namespace std;

class Base

{

int x;

public:

virtual void fun() = 0;

int getX()

{

return x;

}

};

// This class inherits from Base and implements fun()

class Derived: public Base

{

int y;

public:

void fun() { cout << " Derived class function called"; }

};

int main(void)

{

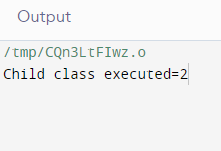
Derived d;

d.fun();

return 0;

}

**Output Screenshot:**





**LAB # 09**

**Lab Objective:**

Objective of this lab is to enable students understand and implement dynamic memory creation and deletion.

**Activity Outcomes:**

The student will understand dynamic memory creation and deletion.

**Instructor Note:**

The Students should have knowledge about C++ compiler, variables and functions, classes and objects life cycle.

**Introduction:**

Dynamic memory allocation in C/C++ refers to performing memory allocation manually by programmer. Dynamically allocated memory is allocated on Heap and non-static and local variables get memory allocated on Stack.

**new operator**

The new operator denotes a request for memory allocation on the Free Store. If sufficient memory is available, new operator initializes the memory and returns the address of the newly allocated and initialized memory to the pointer variable.

**delete operator**

Since it is programmer’s responsibility to deallocate dynamically allocated memory, programmers are provided delete operator by C++ language.

**Activity-1(Solved):**

Write a C++ program that uses new operator to make an object of class Box.

**Solution:**

#include <iostream>

using namespace std;

class Box {

public:

Box() {

cout << "Constructor called!" <<endl;

}

~Box() {

cout << "Destructor called!" <<endl;

}

};

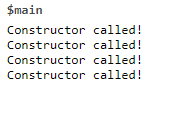
int main() {

Box\* myBoxArray = new Box[4];

return 0;

}

**Output Screenshot:**



**Activity-2:**

Write a C++ program that uses delete keyword to deallocate memory from class Box.

**Solution:**

#include <iostream>

using namespace std;

class Box {

public:

Box() {

cout << "Constructor called!" <<endl;

}

~Box() {

cout << "Destructor called!" <<endl;

}

};

int main() {

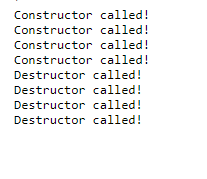
Box\* myBoxArray = new Box[4];

delete [] myBoxArray; // Delete array

return 0;

}

**Output Screenshot:**





**LAB # 10**

**Lab Objective:**

Objective of this lab is to make students understand the template functions and template classes.

**Activity Outcomes:**

The student will understand the advantages of template functions and template classes. Students will learn that how template functions and classes can be used to write generalize code.

**Instructor Note:**

The Students should have knowledge about functions and classes.

**Introduction:**

Templates are powerful features of C++ which allows us to write generic programs. The simple idea is to pass data type as a parameter so that we don’t need to write the same code for different data types. For example, if we need to sort data of different data types then we don’t need to write different function to sort data of each type rather we can write a single template function and send the data type at runtime.

Similar to function templates, we can use class templates to create a single class to work with different data types. Class templates come in handy as they can make our code shorter and more manageable.

A function template starts with the keyword **template** followed by template parameter(s) inside <> which is followed by the function definition.

template <typename T>

T functionName(T parameter1, T parameter2, ...) {

// code

}

Like function templates, we can also create class templates for generic class operations.

Sometimes, you need a class implementation that is same for all classes, only the data types used are different. Normally, we might need to create a different class for each data type OR create different member variables and functions within a single class.

Syntax

template <class T>

class className {

private:

T var;

... .. ...

public:

T functionName(T arg);

... .. ...

};

In the above declaration, T is the template argument which is a placeholder for the data type used. Inside the class body, a member variable var and a member function someOperation() are both of type T.

**Activity-1(Solved):**

Write a C++ template function to accept data type and two values and return the larger value. Write also the main function to test this template function.

**Solution:**

#include<iostream>

Using namespace std;

template <typename T>

T myMax(T x, T y)

{

return (x > y)? x: y;

}

int main()

{

cout << myMax<int>(3, 7) << endl; // Call myMax for int

cout << myMax<double>(3.0, 7.0) << endl; // call myMax for double

cout << myMax<char>('g', 'e') << endl; // call myMax for char

return 0;

}

**Output Screenshot:**

A picture containing text

Description automatically generated

**Activity-2:**

Write a template function to implement Bubble Sort such that it should be able to sort data of any type. Write also the main function to test this template function.

**Solution:**

#include<iostream>

using namespace std;

template <class T>

void bubbleSort(T a[], int n) {

for (int i = 0; i < n - 1; i++)

for (int j = n - 1; i < j; j--)

if (a[j] < a[j - 1])

swap(a[j], a[j - 1]);

}

int main() {

int a[5] = {10, 50, 30, 40, 20};

int n = sizeof(a) / sizeof(a[0]);

cout << " Original array 1: ";

for (int i = 0; i < n; i++)

cout << a[i] << " ";

cout<<endl;

// calls template function by sending integer array

bubbleSort<int>(a, n);

cout << " Sorted array : ";

for (int i = 0; i < n; i++)

cout << a[i] << " ";

cout << endl<<endl;

float b[5] = {1.1, 5.5, 3.3, 4.4, 2.3};

int s = sizeof(b) / sizeof(b[0]);

cout << " Original array 2: ";

for (int i = 0; i < n; i++)

cout << b[i] << " ";

cout<<endl;

// calls template function by sending float array

bubbleSort<float>(b, n);

cout << " Sorted array : ";

for (int i = 0; i < n; i++)

cout << b[i] << " ";

cout << endl;

return 0;

}

**Output Screenshot:**

Text

Description automatically generated

**Activity-3:**

Write a template class to perform the basic arithmetic operations (add, subtract, multiply, divide) on two values where the data type of these values will be given at runtime.

Write also the main function to test this template class.

**Solution:**

#include <iostream>

using namespace std;

template <class T>

class Calculator

{

private:

T num1, num2;

public:

Calculator(T n1, T n2)

{ num1 = n1;

num2 = n2;

}

void displayResult()

{ cout << "Numbers are: " << num1 << " and " << num2 << "." << endl;

cout << "Addition is: " << add() << endl;

cout << "Subtraction is: " << subtract() << endl;

cout << "Product is: " << multiply() << endl;

cout << "Division is: " << divide() << endl;

}

T add() { return num1 + num2; }

T subtract() { return num1 - num2; }

T multiply() { return num1 \* num2; }

T divide() { return num1 / num2; }

};

int main()

{

Calculator<int> intCalc(2, 1);

Calculator<float> floatCalc(2.4, 1.2);

cout << "Int results:" << endl;

intCalc.displayResult();

cout << endl << "Float results:" << endl;

floatCalc.displayResult();

return 0;

}

**Output Screenshot:**

Text

Description automatically generated



**LAB # 11**

**Lab Objective:**

Objective of this lab is to make students understand the concept of exceptions . This lab will also cover exceptions with multi error classes with arguments.

**Activity Outcomes:**

The student will understand that what are exceptions, why exceptions are necessary to handle and how the exceptions are handled.

**Instructor Note:**

The Students should have knowledge about classes, objects and different types of errors.

**Introduction:**

Exceptions are run-time anomalies or abnormal conditions that a program encounters during its execution. Exceptions are also sometimes called run-time errors. These are the errors raised at the time of program execution. When a run-time occurs at the time of program execution then the execution is stopped and the remaining code of the program is not executed. This abnormal termination of program execution may lead to serious anomalies therefore they should be handled.

In C++, exceptions are handled by using **try**, **throw** and **catch** keywords. The **try** statement allows you to define a block of code to be tested for errors while it is being executed. The **throw** keyword throws an exception when a problem is detected, which lets us create a custom error. The **catch** statement allows you to define a block of code to be executed, if an error occurs in the try block.

try-thow-catch can be used as below:

try {

// block of code to try

throw exception; // throw an exception when a problem arise

}

catch ( ) {

// block of code to handle errors

}

**Activity-1(Solved):**

Write a C++ program to accept age of the user. If the user has age greater than or equal to 18 then grant him/her access otherwise deny the access.

**Solution:**

#include<iostream>

using namespace std;

int main()

{

int age;

try

{

cout<<"Enter your age ";

cin>>age;

if (age >= 18)

cout << "Access granted - you are old enough.";

else

throw (age);

}

catch (int myNum)

{

cout << "Access denied - You must be at least 18 years old.\n";

cout << "Your age is: " << myNum;

}

}

**Output Screenshot:**

Text

Description automatically generated

Text

Description automatically generated with low confidence

**Activity-2:**

Write a C++ program to accept two integer values from user. If first number is less than second number than an exception “"First number should be greater than second number” should be raised. If the second number is 0 then “"Divide by zero error" should be raised.

* This example shows that how multiple exceptions can be handled.

**Solution:**

#include<iostream>

using namespace std;

int main()

{

int a,b;

cout<<"Enter 1st Number ";

cin>>a;

cout<<"Enter 2nd Number ";

cin>>b;

try

{

if(b==0)

throw "Divide by zero error";

else if(a<b)

throw (a);

else

{

float c=(float)a/b;

cout<<"Result = "<<c<<endl;

}

}

catch (const char\* msg )

{

cout<<msg<<endl;

}

catch (int firstNum)

{

cout << "First number should be greater than second number\n";

cout << "You entered: " << firstNum<<endl;

}

}

**Output Screenshot:**

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

**Activity-3:**

We can also write a single catch block to handle any kind of exception. This is done by writing three dots inside the catch keyword as illustrated below.

Write a C++ program to accept two integer values from user. If the first value is less than the second value or the second value is 0 then an exception "You have entered wrong input" should be raised.

* This activity illustrates that how multiple exceptions can be handled by a single catch block.

**Solution:**

#include<iostream>

using namespace std;

int main()

{

int a,b;

cout<<"Enter 1st Number ";

cin>>a;

cout<<"Enter 2nd Number ";

cin>>b;

try

{

if(b==0)

throw "Divide by zero error";

else if(a<b)

throw (a);

else

{

float c=(float)a/b;

cout<<"Result = "<<c<<endl;

}

}

catch(...)

{

cout<<"You have entered wrong input\n";

}

}

**Output Screenshot:**

Text

Description automatically generated

Text

Description automatically generated

**Activity-4:**

Write a C++ class to accept program to accept two integer values. If first number is less than second number than an exception “First number should be greater than second number” should be raised. If the second number is 0 then “Divide by zero error" should be raised.

Write main( ) function to implement the above class.

* This example shows that how multiple exceptions can be handled by using a class.

**Solution:**

#include<iostream>

using namespace std;

class myClass

{

public:

int a;

int b;

myClass(int x, int y)

{

a=x;

b=y;

}

void function()

{

if(b==0)

throw "Divide by zero error";

else if(a<b)

throw (a);

else

{

float c=(float)a/b;

cout<<"Result = "<<c<<endl;

}

}

};

int main()

{

int a,b;

cout<<"Enter 1st Number ";

cin>>a;

cout<<"Enter 2nd Number ";

cin>>b;

myClass obj(a,b);

try

{

obj.function();

}

catch (const char\* msg )

{

cout<<msg<<endl;

}

catch (int firstNum)

{

cout << "First number should be greater than second number\n";

cout << "You entered: " << firstNum<<endl;

}

}

**Output Screenshot:**

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated



**LAB # 12**

**Lab Objective:**

Objective of this lab is to ….

**Activity Outcomes:**

The student will understand the ….

**Instructor Note:**

The Students should have knowledge about …...

**Introduction:**

Object-oriented programming…….

**-------------- Coming Soon ----------------**

**Activity-1(Solved):**

Write a C++ program….

**Solution:**

**Output Screenshot:**

**Activity-2:**

Write a C++ program….

**Solution:**

**Output Screenshot:**

**Activity-3:**

Write a C++ program….

**Solution:**

**Output Screenshot:**

**Activity-4:**

Write a C++ program….

**Solution:**

**Output Screenshot:**



**LAB # 13**

**Lab Objective:**

Objective of this lab is to ….

**Activity Outcomes:**

The student will understand the ….

**Instructor Note:**

The Students should have knowledge about …...

**Introduction:**

Object-oriented programming…….

**-------------- Coming Soon ----------------**

**Activity-1(Solved):**

Write a C++ program….

**Solution:**

**Output Screenshot:**

**Activity-2:**

Write a C++ program….

**Solution:**

**Output Screenshot:**

**Activity-3:**

Write a C++ program….

**Solution:**

**Output Screenshot:**

**Activity-4:**

Write a C++ program….

**Solution:**

**Output Screenshot:**