Q4. Wap to swap 2 numbers using template function

#include <iostream>

**using** **namespace** std;

**template** <**class** T>

**int** swap\_numbers(T& x, T& y)

{

    T t;

    t = x;

    x = y;

    y = t;

**return** 0;

}

**int** main()

{

**int** a, b;

    a = 10, b = 20;

    swap\_numbers(a, b);

    cout << a << " " << b << endl;

**return** 0;

}

Q6. Define class distance which contains km,cm and m data member. Using

Parameterized and copy constructor and overload " \*" operator

#include <iostream>

**using** **namespace** std;

**class** construct

{

**public**:

**float** area;

    construct()

    {

        area = 0;

    }

    construct(**int** a, **int** b)

    {

        area = a \* b;

    }

**void** disp()

    {

        cout<< area<< endl;

    }

};

**int** main()

{

    construct o;

    construct o2( 10, 20);

    o.disp();

    o2.disp();

**return** 1;

}

Q7. Create a class MAT of size m\*n. Define all possible matrix operations for MAT

type objects

#include "bits/stdc++.h"

#define rows 50

#define cols 50

**using** **namespace** std;

**int** N;

// Class for Matrix operator overloading

**class** Matrix {

    // For input Matrix

**int** arr[rows][cols];

**public**:

    // Function to take input to arr[][]

**void** input(vector<vector<**int**> >& A);

**void** display();

    // Functions for operator overloading

**void** operator+(Matrix x);

**void** operator-(Matrix x);

**void** operator\*(Matrix x);

};

// Functions to get input to Matrix

// array arr[][]

**void** Matrix::input(vector<vector<**int**> >& A)

{

    // Traverse the vector A[][]

**for** (**int** i = 0; i < N; i++) {

**for** (**int** j = 0; j < N; j++) {

            arr[i][j] = A[i][j];

        }

    }

}

// Function to display the element

// of Matrix

**void** Matrix::display()

{

**for** (**int** i = 0; i < N; i++) {

**for** (**int** j = 0; j < N; j++) {

            // Print the element

            cout << arr[i][j] << ' ';

        }

        cout << endl;

    }

}

// Function for addition of two Matrix

// using operator overloading

**void** Matrix::operator+(Matrix x)

{

    // To store the sum of Matrices

**int** mat[N][N];

    // Traverse the Matrix x

**for** (**int** i = 0; i < N; i++) {

**for** (**int** j = 0; j < N; j++) {

            // Add the corresponding

            // blocks of Matrices

            mat[i][j] = arr[i][j]

                        + x.arr[i][j];

        }

    }

    // Display the sum of Matrices

**for** (**int** i = 0; i < N; i++) {

**for** (**int** j = 0; j < N; j++) {

            // Print the element

            cout << mat[i][j] << ' ';

        }

        cout << endl;

    }

}

// Function for subtraction of two Matrix

// using operator overloading

**void** Matrix::operator-(Matrix x)

{

    // To store the difference of Matrices

**int** mat[N][N];

    // Traverse the Matrix x

**for** (**int** i = 0; i < N; i++) {

**for** (**int** j = 0; j < N; j++) {

            // Subtract the corresponding

            // blocks of Matrices

            mat[i][j] = arr[i][j]

                        - x.arr[i][j];

        }

    }

    // Display the difference of Matrices

**for** (**int** i = 0; i < N; i++) {

**for** (**int** j = 0; j < N; j++) {

            // Print the element

            cout << mat[i][j] << ' ';

        }

        cout << endl;

    }

}

// Function for multiplication of

// two Matrix using operator

// overloading

**void** Matrix::operator\*(Matrix x)

{

    // To store the multiplication

    // of Matrices

**int** mat[N][N];

    // Traverse the Matrix x

**for** (**int** i = 0; i < N; i++) {

**for** (**int** j = 0; j < N; j++) {

            // Initialise current block

            // with value zero

            mat[i][j] = 0;

**for** (**int** k = 0; k < N; k++) {

                mat[i][j] += arr[i][k]

                             \* (x.arr[k][j]);

            }

        }

    }

    // Display the multiplication

    // of Matrices

**for** (**int** i = 0; i < N; i++) {

**for** (**int** j = 0; j < N; j++) {

            // Print the element

            cout << mat[i][j] << ' ';

        }

        cout << endl;

    }

}

// Driver Code

**int** main()

{

    // Dimension of Matrix

    N = 3;

    vector<vector<**int**> > arr1

        = { { 1, 2, 3 },

            { 4, 5, 6 },

            { 1, 2, 3 } };

    vector<vector<**int**> > arr2

        = { { 1, 2, 3 },

            { 4, 5, 16 },

            { 1, 2, 3 } };

    // Declare Matrices

    Matrix mat1, mat2;

    // Take Input to matrix mat1

    mat1.input(arr1);

    // Take Input to matrix mat2

    mat2.input(arr2);

    // For addition of matrix

    cout << "Addition of two given"

         << " Matrices is : \n";

    mat1 + mat2;

    cout << "Subtraction of two given"

         << " Matrices is : \n";

    mat1 - mat2;

    cout << "Multiplication of two"

         << " given Matrices is : \n";

    mat1\* mat2;

**return** 0;

}

Q9. Write a program calculate arithmetic calculator using class template.

#include<iostream>

**using** **namespace** std;

**template** <**class** T>

**class** Calculator{

**protected**:

T a, b;

**public**:

Calculator(){

cout<<"\nEnter the value of a and b : ";

cin>>a>>b;

}

**void** display(){

cout<<"\nA = "<<a<<"\nB = "<<b;

}

**inline** T add(){

**return** (a+b);

}

**inline** T sub(){

**return** (a-b);

}

**inline** T div(){

**return** (a/b);

}

**inline** T mul(){

**return** (a\*b);

}

};

**int** main(){

Calculator<**float**>c;

c.display();

**int** ch;

label1:

cout<<"\n1.Addition"<<endl;

cout<<"2.Subtraction"<<endl;

cout<<"3.Division"<<endl;

cout<<"4.Multiplication"<<endl;

cout<<"\nEnter you choice : "<<endl;

cin>>ch;

**switch**(ch){

**case** 1:

cout<<"Addition = "<<c.add();

**break**;

**case** 2:

cout<<"Subtraction = "<<c.sub();

**break**;

**case** 3:

cout<<"Division = "<<c.div();

**break**;

**case** 4:

cout<<"Multiplication = "<<c.mul();

**break**;

**default**:

cout<<"\nEnter the valid choice.";

**goto** label1;

}

**return** 0;

}

Q10. Create class Person(code,name), Pay(salary), Admin(expirience), and Master

class to create, update, display record.

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <windows.h>

// Structure of the employee

**struct** emp {

**char** name[50];

**float** salary;

**int** age;

**int** id;

};

**struct** emp e;

// size of the structure

**long** **int** size = **sizeof**(e);

// In the start coordinates

// will be 0, 0

COORD cord = { 0, 0 };

// function to set the

// coordinates

**void** gotoxy(**int** x, **int** y)

{

    cord.X = x;

    cord.Y = y;

    SetConsoleCursorPosition(

        GetStdHandle(STD\_OUTPUT\_HANDLE),

        cord);

}

**FILE** \*fp, \*ft;

// Function to add the records

**void** addrecord()

{

**system**("cls");

**fseek**(fp, 0, SEEK\_END);

**char** another = 'y';

**while** (another == 'y') {

**printf**("\nEnter Name : ");

**scanf**("%s", e.name);

**printf**("\nEnter Age : ");

**scanf**("%d", &e.age);

**printf**("\nEnter Salary : ");

**scanf**("%f", &e.salary);

**printf**("\nEnter EMP-ID : ");

**scanf**("%d", &e.id);

**fwrite**(&e, size, 1, fp);

**printf**("\nWant to add another"

               " record (Y/N) : ");

**fflush**(stdin);

**scanf**("%c", &another);

    }

}

// Function to delete the records

**void** deleterecord()

{

**system**("cls");

**char** empname[50];

**char** another = 'y';

**while** (another == 'y') {

**printf**("\nEnter employee "

               "name to delete : ");

**scanf**("%s", empname);

        ft = **fopen**("temp.txt", "wb");

**rewind**(fp);

**while** (**fread**(&e, size,

                     1, fp)

               == 1) {

**if** (**strcmp**(e.name,

                       empname)

                != 0)

**fwrite**(&e, size, 1, ft);

        }

**fclose**(fp);

**fclose**(ft);

**remove**("data.txt");

**rename**("temp.txt", "data.txt");

        fp = **fopen**("data.txt", "rb+");

**printf**("\nWant to delete another"

               " record (Y/N) :");

**fflush**(stdin);

        another = getche();

    }

}

// Function to display the record

**void** displayrecord()

{

**system**("cls");

    // sets pointer to start

    // of the file

**rewind**(fp);

**printf**("\n========================="

           "==========================="

           "======");

**printf**("\nNAME\t\tAGE\t\tSALARY\t\t"

           "\tID\n",

           e.name, e.age,

           e.salary, e.id);

**printf**("==========================="

           "==========================="

           "====\n");

**while** (**fread**(&e, size, 1, fp) == 1)

**printf**("\n%s\t\t%d\t\t%.2f\t%10d",

               e.name, e.age, e.salary, e.id);

**printf**("\n\n\n\t");

**system**("pause");

}

// Function to modify the record

**void** modifyrecord()

{

**system**("cls");

**char** empname[50];

**char** another = 'y';

**while** (another == 'y') {

**printf**("\nEnter employee name"

               " to modify : ");

**scanf**("%s", empname);

**rewind**(fp);

        // While File is open

**while** (**fread**(&e, size, 1, fp) == 1) {

            // Compare the employee name

            // with ename

**if** (**strcmp**(e.name, empname) == 0) {

**printf**("\nEnter new name:");

**scanf**("%s", e.name);

**printf**("\nEnter new age :");

**scanf**("%d", &e.age);

**printf**("\nEnter new salary :");

**scanf**("%f", &e.salary);

**printf**("\nEnter new EMP-ID :");

**scanf**("%d", &e.id);

**fseek**(fp, -size, SEEK\_CUR);

**fwrite**(&e, size, 1, fp);

**break**;

            }

        }

        // Ask for modifying another record

**printf**("\nWant to modify another"

               " record (Y/N) :");

**fflush**(stdin);

**scanf**("%c", &another);

    }

}

// Driver code

**int** main()

{

**int** choice;

    // opening the file

    fp = **fopen**("data.txt", "rb+");

    // showing error if file is

    // unable to open.

**if** (fp == NULL) {

        fp = **fopen**("data.txt", "wb+");

**if** (fp == NULL) {

**printf**("\nCannot open file...");

**exit**(1);

        }

    }

**system**("Color 3F");

**printf**("\n\n\n\n\t\t\t\t============="

           "============================="

           "===========");

**printf**("\n\t\t\t\t~~~~~~~~~~~~~~~~~~~"

           "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"

           "~~~~~");

**printf**("\n\t\t\t\t==================="

           "============================="

           "=====");

**printf**("\n\t\t\t\t[|:::>:::>:::>::>  "

           "EMPLOYEE RECORD  <::<:::<:::"

           "<:::|]\t");

**printf**("\n\t\t\t\t==================="

           "============================="

           "=====");

**printf**("\n\t\t\t\t~~~~~~~~~~~~~~~~~~~~"

           "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"

           "~~~");

**printf**("\n\t\t\t\t====================="

           "==============================\n");

**printf**("\n\n\n\t\t\t\t\t\t\t\t\t\t"

           "Developer : @Sushant\_Gaurav"

           "\n\n\t\t\t\t");

**system**("pause");

**while** (1) {

        // Clearing console and asking the

        // user for input

**system**("cls");

        gotoxy(30, 10);

**printf**("\n1. ADD RECORD\n");

        gotoxy(30, 12);

**printf**("\n2. DELETE RECORD\n");

        gotoxy(30, 14);

**printf**("\n3. DISPLAY RECORDS\n");

        gotoxy(30, 16);

**printf**("\n4. MODIFY RECORD\n");

        gotoxy(30, 18);

**printf**("\n5. EXIT\n");

        gotoxy(30, 20);

**printf**("\nENTER YOUR CHOICE...\n");

**fflush**(stdin);

**scanf**("%d", &choice);

        // Switch Case

**switch** (choice) {

**case** 1:

            // Add the records

            addrecord();

**break**;

**case** 2:

            // Delete the records

            deleterecord();

**break**;

**case** 3:

            // Display the records

            displayrecord();

**break**;

**case** 4:

            // Modify the records

            modifyrecord();

**break**;

**case** 5:

**fclose**(fp);

**exit**(0);

**break**;

**default**:

**printf**("\nINVALID CHOICE...\n");

        }

    }

**return** 0;

}

Q11. WAP to copy contents of one file to another

#include <stdio.h>

#include <stdlib.h> // For exit()

**int** main()

{

**FILE** \*fptr1, \*fptr2;

**char** filename[100], c;

**printf**("Enter the filename to open for reading \n");

**scanf**("%s", filename);

    // Open one file for reading

    fptr1 = **fopen**(filename, "r");

**if** (fptr1 == NULL)

    {

**printf**("Cannot open file %s \n", filename);

**exit**(0);

    }

**printf**("Enter the filename to open for writing \n");

**scanf**("%s", filename);

    // Open another file for writing

    fptr2 = **fopen**(filename, "w");

**if** (fptr2 == NULL)

    {

**printf**("Cannot open file %s \n", filename);

**exit**(0);

    }

    // Read contents from file

    c = **fgetc**(fptr1);

**while** (c != EOF)

    {

**fputc**(c, fptr2);

        c = **fgetc**(fptr1);

    }

**printf**("\nContents copied to %s", filename);

**fclose**(fptr1);

**fclose**(fptr2);

**return** 0;

}

Q29. Write a program by using friend class to show swapping of data

#include <iostream>

**using** **namespace** std;

**class** Swap {

    // Declare the variables of Swap Class

**int** temp, a, b;

**public**:

    // Define the parameterized constructor, for inputs

    Swap(**int** a, **int** b)

    {

**this**->a = a;

**this**->b = b;

    }

    // Declare the friend function to swap, take arguments

    // as call by reference

**friend** **void** swap(Swap&);

};

// Define the swap function outside class scope

**void** swap(Swap& s1)

{

    // Call by reference is used to passed object copy to

    // the function

    cout << "\nBefore Swapping: " << s1.a << " " << s1.b;

    // Swap operations with Swap Class variables

    s1.temp = s1.a;

    s1.a = s1.b;

    s1.b = s1.temp;

    cout << "\nAfter Swapping: " << s1.a << " " << s1.b;

}

// Driver Code

**int** main()

{

    // Declare and Initialize the Swap object

    Swap s(4, 6);

    swap(s);

**return** 0;

}

Q27. Write a program for reading and writing

data to and from the file using command line arguments.

#include <iostream>

**using** **namespace** std;

**int** main(**int** argc, **char**\*\* argv)

{

    cout << "You have entered " << argc

         << " arguments:" << "\n";

**for** (**int** i = 0; i < argc; ++i)

        cout << argv[i] << "\n";

**return** 0;

}

Q24. Write a C++ program to concatenate two Strings using operator + function.

(using operator overloading)

#include <iostream>

#include <string.h>

**using** **namespace** std;

// Class to implement operator overloading

// function for concatenating the strings

**class** AddString {

**public**:

    // Classes object of string

**char** s1[25], s2[25];

    // Parameterized Constructor

    AddString(**char** str1[], **char** str2[])

    {

        // Initialize the string to class object

**strcpy**(**this**->s1, str1);

**strcpy**(**this**->s2, str2);

    }

    // Overload Operator+ to concat the string

**void** operator+()

    {

        cout << "\nConcatenation: " << **strcat**(s1, s2);

    }

};

// Driver Code

**int** main()

{

    // Declaring two strings

**char** str1[] = "Geeks";

**char** str2[] = "ForGeeks";

    // Declaring and initializing the class

    // with above two strings

    AddString a1(str1, str2);

    // Call operator function

    +a1;

**return** 0;

}

Q20. Demonstrate the use of static variable and static function

#include <iostream>

#include <string>

**using** **namespace** std;

**void** demo()

{

    // static variable

**static** **int** count = 0;

    cout << count << " ";

    // value is updated and

    // will be carried to next

    // function calls

    count++;

}

**int** main()

{

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

        demo();

**return** 0;

}

Q19. Create a class String and define necessary members and constructors to overload '+' to concatenate two strings and display it

#include <iostream>

#include <string.h>

**using** **namespace** std;

// Class to implement operator overloading

// function for concatenating the strings

**class** AddString {

**public**:

    // Classes object of string

**char** s1[25], s2[25];

    // Parameterized Constructor

    AddString(**char** str1[], **char** str2[])

    {

        // Initialize the string to class object

**strcpy**(**this**->s1, str1);

**strcpy**(**this**->s2, str2);

    }

    // Overload Operator+ to concat the string

**void** operator+()

    {

        cout << "\nConcatenation: " << **strcat**(s1, s2);

    }

};

// Driver Code

**int** main()

{

    // Declaring two strings

**char** str1[] = "Geeks";

**char** str2[] = "ForGeeks";

    // Declaring and initializing the class

    // with above two strings

    AddString a1(str1, str2);

    // Call operator function

    +a1;

**return** 0;

}

Q28. Define a class to represent a bank account. Include the following members:

Data Members :-

Name of the Depositor, Account number, Type of account, Balance amount in the

account

Member Functions: To assign initial values, To deposit an amount, To withdraw an

amount after checking thebalance, To display name and balance

#include<iostream>

**using** **namespace** std;

**class** Account

{

**char** accname[30];

**int** accno;

**char** type[10];

**int** balance;

**public**:

**void** init();

**void** deposit();

**void** withdraw();

**void** display();

};

**void** Account::init()

{

cout<<"\nEnter the Account holder's Name: "<<endl;

cin>>accname;

cout<<"\nEnter the Account Number: "<<endl;

cin>>accno;

cout<<"\nEnter the Type of Account: "<<endl;

cin>>type;

cout<<"\nEnter the Balance in Account: "<<endl;

cin>>balance;

}

**void** Account::deposit()

{

**int** amount = 0;

cout<<"\n\t\tYou are here to deposit the amount\n"<<endl;

    cout<<"\nEnter the amount to depositin the Account: "<<endl;

cin>>amount;

balance += amount;

}

**void** Account::withdraw()

{

**int** amount = 0;

cout<<"\n\t\tYou are here to withdraw the amount\n"<<endl;

cout<<"\nEnter the amount to withdraw from the Account: "<<endl;

cin>>amount;

**if**(amount >= balance)

cout<<"\tInsufficient Balance"<<endl;

**else**

balance -= amount;

}

**void** Account::display()

{

cout<<"Account Number: "<<accno<<endl;

cout<<"Account Holder's Name: "<<accname<<endl;

cout<<"Type of Account: "<<type<<endl;

cout<<"Balance: "<<balance<<endl;

}

**int** main()

{

Account A;

cout<<"\n------------------Welcome to Modern Bank------------"<<endl;

A.init();

A.deposit();

A.withdraw();

A.display();

**return** 0;

}

Q1. Define a class Animal with their basic features as class members.

Create two derived classes from Animal

named herbivores and Carnivores (type) with their own features too.

Accept name of animal with type and display all the related information.

#include<iostream>

**using** **namespace** std;

**class** Animal

{

**char** name[30];

**char** type[20];

**public**:

**void** getdata()

{

 cout<<"Enter the name of an Animal:";

 cin.getline(name,30);

 cout<<endl;

 cout<<"Enter the type of an Animal:";

 cin.getline(type,20);

}

**void** putdata()

{

 cout<<name<<" "<<type<<endl;

}

};

**class** Herbivores : **private** Animal

{

**char** nm[50];

**char** food[50];

**public**:

**void** getdata()

 {

 cout<<"Enter the name of Herbivores animal:";

 cin.getline(nm,50);

 cout<<endl;

 cout<<"Enter the food of Herbivores Animal:";

 cin.getline(food,50);

 }

**void** putdata()

 {

 cout<<endl;

 cout<<nm<<" "<<food;

 }

};

**class** Carnivores:**private** Animal

{

**char** nm[50];

**char** food[50];

**public**:

**void** getdata()

 {

 cout<<"Enter the name of the carnivores animal:";

 cin.getline(nm,50);

 cout<<endl;

 cout<<"Enter the food of the carnivores animal:";

 cin.getline(food,50);

 cout<<endl;

 }

**void** putdata()

 {

cout<<nm<<" "<<food;

 }

};

**int** main()

{

 Herbivores h1;

 h1.getdata();

 Carnivores c1;

 c1.getdata();

 cout<<endl;

 h1.putdata();

 cout<<endl;

 c1.putdata();

}

Q8. Define base class Student with Roll\_No, Name, Marks1, Marks2, Marks3 as

data members. Define Sports class with SportName, ParticipationLevel,

Achievement as data members and also find sports grade. Inherit Student Class in

Sports Class to find final grade of five students.

(District/Gold - 3 , District/Silver - 2 , District/Bronze - 1 ,

State/Gold - 6 , State /Silver - 5 , State/Bronze - 4 ,

National/Gold - 9 , National/ Silver - 8 , National/Bronze - 7)

#include<iostream>

**using** **namespace** std;

**class** stud\_info{

**protected**:

**int** roll;

**char** name[30];

**public**:

**void** get\_stud\_data(){

cout<<"Enter your roll Number: ";

cin>>roll;

cout<<"Enter your name: ";

cin>>name;

}

};

**class** test\_info:**virtual** **public** stud\_info{

**protected**:

**int** test1,test2,test3;

**public**:

**void** get\_test\_data(){

cout<<"Enter marks of first test: ";

cin>>test1;

cout<<"Enter marks of second test: ";

cin>>test2;

cout<<"Enter the marks of third test: ";

cin>>test3;

}

};

**class** sports\_info : **virtual** **public** stud\_info{

**protected**:

**char** sports\_remark[30];

**int** sports\_mark;

**public**:

**void** get\_sports\_marks(){

cout<<"Enter your sports remark: ";

cin>>sports\_remark;

**if**(sports\_remark=="dist gold"){

sports\_mark=3;

}

**else** **if**(sports\_remark=="dist silver"){

sports\_mark=2;

}

**else** **if**(sports\_remark=="dist bronze"){

sports\_mark=1;

}

**else** **if**(sports\_remark=="state g"){

sports\_mark=6;

}

**else** **if**(sports\_remark=="state s"){

sports\_mark=5;

}

**else** **if**(sports\_remark=="state b"){

sports\_mark=4;

}

**else** **if**(sports\_remark=="national g"){

sports\_mark=9;

}

**else** **if**(sports\_remark=="national s"){

sports\_mark=8;

}

**else** **if**(sports\_remark=="national b"){

sports\_mark=7;

}

}

};

**class** result : **public** test\_info,**public** sports\_info{

**protected**:

**int** per;

**public**:

**void** cal(){

per=per+sports\_mark;

per=(test1+test2+test3)/3;

}

**void** display(){

cout<<"Roll number: "<<roll<<"\nName: "<<name<<"\nYour percentage: "<<per<<"%";

}

};

**int** main(){

result obj;

obj.get\_stud\_data();

obj.get\_test\_data();

obj.get\_sports\_marks();

obj.cal();

obj.display();

**return** -1;

}

Q13. Write a program of multiple inheritance for a company that publishes and

markets books. Derive a pamphlet, book and tape from publication And tape &

book from sales. And notice from pamphlet. Display a book information.

#include <iostream>

**using** **namespace** std;

**class** pamphlet

{

**public**:

**void** showp()

 {

 cout<<"The name of the book is THE WINGS OF FIRE";

 }

};

**class** book

{

**public**:

**void** showb()

 {

 cout<<"\nIt is based on the life of APJ ABDUL KALAM.";

 }

};

**class** tape

{

**public**:

**void** showt()

 {

 cout<<"\nfirst edition of the book is released on 1999";

 }

};

**class** publication:**public** pamphlet,**public** book,**public** tape

{

**public**:

**void** pub()

 {

 cout<<"\nIt is published by UNIVERSITIES PRESS";

 }

};

**class** sales:**public** tape,**public** book

{

**public**:

**void** showsale()

 {

 cout<<"\nThe price of the book is 200.";

 }

};

**int** main()

{

 publication p;

 p.showp();

 p.showb();

 p.showt();

 p.pub();

 cout<<"\n----------------------------------------------------------------------------------";

 sales s;

 s.showsale();

 s.showb();

 s.showt();

**return** 0;

}

Q12. Write a function in c++ to count and display the number of lines not starting

with alphabet 'A' present in a text file 'MCA.TXT'.

#include <iostream>

#include <fstream>

#include <string>

**using** **namespace** std;

**void** countAndDisplay() {

 ifstream ifs("STORY.TXT");

**if** (!ifs) {

 cout << "Error: Failed to open input file STORY.TXT" << endl;

**return**;

 }

**int** count = 0;

 string line;

**while** (getline(ifs, line)) {

**if** (line[0] != 'A' && line[0] != 'a') {

 count++;

 cout << line << endl; // Display the line

 }

 }

 ifs.close();

 cout << "Number of lines not starting with 'A': " << count << endl;

}

**int** main() {

 countAndDisplay();

**return** 0;

}

Q23. write a program for reading and writing data to and from file using command

line arguments

#include <iostream>

#include <fstream>

**using** **namespace** std;

**int** main(**int** argc,**char** \*argv[]) {

fstream my\_file;

my\_file.open(argv[1], ios::out);

**if** (!my\_file) {

cout << "File not created!"<<endl;

}

**else** {

cout << "File created successfully!";

my\_file << "Hello World!";

my\_file.close();

}

 //fstream my\_file;

my\_file.open(argv[1], ios::in);

**if** (!my\_file) {

cout << "No such file";

}

**else** {

**char** ch;

**while** (1) {

my\_file >> ch;

**if** (my\_file.eof())

**break**;

cout << ch;

}

}

my\_file.close();

**return** 0;

}

Q15. Declare class STACK and handle the run time anomalies like overflow

Underflow

Display error number and error message when error is thrown (use exception

handling)

#include <iostream>

#include <exception>

**using** **namespace** std;

**class** StackException : **public** exception {

**private**:

**int** error\_number;

    string error\_message;

**public**:

    StackException(**int** error\_number, string error\_message) : error\_number(error\_number), error\_message(error\_message) {}

**const** **char**\* what() **const** **throw**() {

**return** error\_message.c\_str();

    }

**int** getErrorNumber() **const** **throw**() {

**return** error\_number;

    }

};

**class** Stack {

**private**:

**int**\* data;

**int** size;

**int** top;

**public**:

    Stack(**int** size) : size(size), top(-1) {

        data = **new** **int**[size];

    }

    ~Stack() {

**delete**[] data;

    }

**bool** isFull() {

**return** top == size - 1;

    }

**bool** isEmpty() {

**return** top == -1;

    }

**void** push(**int** element) {

**if** (isFull()) {

**throw** StackException(1, "Error: Stack Overflow");

        }

        top++;

        data[top] = element;

    }

**int** pop() {

**if** (isEmpty()) {

**throw** StackException(2, "Error: Stack Underflow");

        }

**int** element = data[top];

        top--;

**return** element;

    }

};

**int** main() {

    Stack stack(3);

**try** {

        stack.push(10);

        stack.push(20);

        stack.push(30);

        stack.push(40);

    }

**catch** (StackException& ex) {

        cout << "Exception " << ex.getErrorNumber() << ": " << ex.what() << endl;

    }

**try** {

        stack.pop();

        stack.pop();

        stack.pop();

        stack.pop();

    }

**catch** (StackException& ex) {

        cout << "Exception " << ex.getErrorNumber() << ": " << ex.what() << endl;

    }

**return** 0;

}

Q16. Create two derived classes from person class having member function to get

name and age. create student and employee details with their members functions.

Delete and update student and employee record. Display base and derived class

object address using this pointer

#include <iostream>

#include <string>

**using** **namespace** std;

**class** Person {

**public**:

    string name;

**int** age;

    Person(string name, **int** age) : name(name), age(age) {}

**virtual** **void** display() {

        cout << "Name: " << name << ", Age: " << age << endl;

    }

};

**class** Student : **public** Person {

**public**:

**int** roll\_no;

    string course;

    Student(string name, **int** age, **int** roll\_no, string course) : Person(name, age), roll\_no(roll\_no), course(course) {}

**void** update(**int** new\_roll\_no, string new\_course) {

        roll\_no = new\_roll\_no;

        course = new\_course;

    }

**void** display() {

        cout << "Name: " << name << ", Age: " << age << ", Roll No: " << roll\_no << ", Course: " << course << endl;

    }

};

**class** Employee : **public** Person {

**public**:

**int** emp\_id;

    string designation;

    Employee(string name, **int** age, **int** emp\_id, string designation) : Person(name, age), emp\_id(emp\_id), designation(designation) {}

**void** update(**int** new\_emp\_id, string new\_designation) {

        emp\_id = new\_emp\_id;

        designation = new\_designation;

    }

**void** display() {

        cout << "Name: " << name << ", Age: " << age << ", Emp ID: " << emp\_id << ", Designation: " << designation << endl;

    }

};

**int** main() {

    Person\* person\_ptr;

    Person person("John Doe", 25);

    Student student("Jane Smith", 20, 1001, "Computer Science");

    Employee employee("Bob Williams", 30, 5001, "Manager");

    person\_ptr = &person;

    cout << "Base class object address: " << person\_ptr << endl;

    person.display();

    person\_ptr = &student;

    cout << "Derived class object address: " << person\_ptr << endl;

    student.display();

    person\_ptr = &employee;

    cout << "Derived class object address: " << person\_ptr << endl;

    employee.display();

    // Updating student record

    student.update(1002, "Data Science");

    cout << "Updated student record:" << endl;

    student.display();

    // Updating employee record

    employee.update(5002, "Director");

    cout << "Updated employee record:" << endl;

    employee.display();

**return** 0;

}

Q17. Create the two functions for the manager and the scientist , write the code to

find out who has more salary ( use friend function )

#include <iostream>

#include <string>

**using** **namespace** std;

**class** Scientist;  // Forward declaration

**class** Manager {

**private**:

    string name;

**double** salary;

**public**:

    Manager(string name, **double** salary) : name(name), salary(salary) {}

**friend** **double** compare\_salary(**const** Manager& manager, **const** Scientist& scientist);

};

**class** Scientist {

**private**:

    string name;

**double** salary;

**public**:

    Scientist(string name, **double** salary) : name(name), salary(salary) {}

**friend** **double** compare\_salary(**const** Manager& manager, **const** Scientist& scientist);

};

**double** compare\_salary(**const** Manager& manager, **const** Scientist& scientist) {

**if** (manager.salary > scientist.salary) {

        cout << manager.name << " has a higher salary." << endl;

**return** manager.salary;

    }

**else** **if** (scientist.salary > manager.salary) {

        cout << scientist.name << " has a higher salary." << endl;

**return** scientist.salary;

    }

**else** {

        cout << "Both have the same salary." << endl;

**return** manager.salary;

    }

}

**int** main() {

    Manager manager("John Doe", 50000.0);

    Scientist scientist("Jane Smith", 60000.0);

**double** highest\_salary = compare\_salary(manager, scientist);

    cout << "The highest salary is " << highest\_salary << endl;

**return** 0;

}

Q25. Create two classes manager & scientist having data members salary1 &

salary 2 respectively to store salary. Write a function which finds the maximum

salary between two salaries. [Use friend function]

#include <iostream>

**using** **namespace** std;

**class** scientist;

**class** manager {

**private**:

**int** salary1;

**public**:

    manager(**int** s1) : salary1(s1) {}

**friend** **int** max\_salary(**const** manager& m, **const** scientist& s);

};

**class** scientist {

**private**:

**int** salary2;

**public**:

    scientist(**int** s2) : salary2(s2) {}

**friend** **int** max\_salary(**const** manager& m, **const** scientist& s);

};

**int** max\_salary(**const** manager& m, **const** scientist& s) {

**return** max(m.salary1, s.salary2);

}

**int** main() {

    manager m(5000);

    scientist s(6000);

    cout << "Maximum salary: " << max\_salary(m, s) << endl;

**return** 0;

}

Q26. Create a base class shape. Derive two classes as Triangle and Rectangle

from the base class shape. Take getdata()

and display\_area() as member functions of base class. Make display\_area() as a

virtual function and redefine it in derived classes to suit their requirement. Design a

program that will accept dimensions of a triangle or rectangle interactively and

display the area.

#include<iostream>

**using** **namespace** std;

**class** Shape

{

    public: **double** a,b;

        void get\_data ()

        {

            cin>>a>>b;

        }

        virtual **void** display\_area () = 0;

};

**class** Triangle:**public** Shape

{

    public: **void** display\_area ()

    {

        cout<<"Area of triangle "<<0.5\*a\*b<<endl;

    }

};

**class** Rectangle:**public** Shape

{

    public: **void** display\_area ()

    {

        cout<<"Area of rectangle "<<a\*b<<endl;

    }

};

**int** main()

{

    Triangle t;

    Shape \*st = &t;

    cout<<"Enter base and altitude: ";

    st->get\_data();

    st->display\_area();

    Rectangle r;

    Shape \*sr = &r;

    cout<<"Enter length and breadth: ";

    sr->get\_data();

    sr->display\_area();

    return 0;

}

Q22. Inherit Manager and Scientists class with Salary 1 and Salary 2. Find max

salary and display them using friend function

#include<iostream>

#include<string.h>

**using** **namespace** std;

**class** Employee

{

**public**:

**int** eno;

**char** ename[100];

**float** salary;

**public**:

**void** accept\_details()

                {

                        cout<<"\n Enter Employee Details";

                        cout<<"\n -------------------------------------------";

                        cout<<"\n Enter Employee Id           :  ";

                        cin>>eno;

                        cout<<"\n Enter Employee Name     :  ";

                        cin>>ename;

                        cout<<"\n Enter Salary                     :  ";

                        cin>>salary;

                }

**int** search(**int** empno);       //Function Overloading

**int** search(**char** \*empname); //search() function is overloaded

**int** search(**float** empsalary);

};

**int** Employee :: search(**int** empno)

{

**if**(eno==empno)

        {

                cout<<"\n Displaying Information \n";

                cout<<"\n Employee Name       :   "<<ename;

                cout<<"\n Salary                       :   "<<salary<<endl;

        }

**return** 0;

}

**int** Employee :: search(**char** \*empname)

{

**if**(strcmp(ename,empname)==0)

        {

                cout<<"\n Displaying Information \n";

                cout<<"\n Employee Id         :   "<<eno;

                cout<<"\n Salary                   :   "<<salary;

        }

**return** 0;

}

**int** Employee :: search(**float** empsalary)

{

**if**(salary==empsalary)

        {

                cout<<"\n Displaying Information \n";

                cout<<"\n Employee Id             :   "<<eno;

                cout<<"\n Employee Name       :   "<<ename;

        }

**return** 0;

}

**int** main()

{

        Employee \*emp;

**int** cnt, i, max;

**char** name[100];

**int** eno;

**float** sal;

        cout<<"\n Enter No. of Employee Details You Want  :  ";

        cin>>cnt;

        emp=**new** Employee[cnt];

**for**(i=0;i<cnt;i++)

        {

                emp[i].accept\_details();

        }

        cout<<"\n -------------------------------------------";

        cout<<"\n\n Search Employee by Id         :   ";

        cin>>eno;

**for**(i=0;i<cnt;i++)

        {

                emp[i].search(eno);

        }

        cout<<"\n -------------------------------------------";

        cout<<"\n\n Search Employee by Name       :   ";

        cin>>name;

**for**(i=0;i<cnt;i++)

        {

                emp[i].search(name);

        }

        cout<<"\n -------------------------------------------";

        cout<<"\n\n Search Employee by Salary     :   ";

        cin>>sal;

**for**(i=0;i<cnt;i++)

        {        emp[i].search(sal);

        }

        max=emp[0].salary;

**for**(i=0;i<cnt;i++)

        {

**if**(emp[i].salary>max)

                {

                        max=emp[i].salary;

                }

        }

**for**(i=0;i<cnt;i++)

        {

**if**(emp[i].salary==max)

                {

                        cout<<"\n ------------------------------------------------\n";

                        cout<<"\n Maximum Salary of Employee Name is : "<<emp[i].ename;

                        cout<<"\n And Salary is : "<<emp[i].salary;

                }

        }

**return** 0;

}

Q3. Write base class that ask the user to enter a complex number and derived

class adds the complex number of its own with the base class.Finally make third

class that is friend of derived and calculate the difference of base complex number

and its own complex number

#include <iostream>

**using** **namespace** std;

**class** Complex {

**protected**:

**float** real, imag;

**public**:

    Complex() {

        cout << "Enter real and imaginary parts of complex number: ";

        cin >> real >> imag;

    }

**void** display() {

        cout << "Complex number: " << real << " + " << imag << "i" << endl;

    }

**float** getReal() {

**return** real;

    }

**float** getImag() {

**return** imag;

    }

};

**class** Derived : **public** Complex {

**public**:

    Derived() {

        cout << "Enter real and imaginary parts of another complex number: ";

**float** real2, imag2;

        cin >> real2 >> imag2;

        real += real2;

        imag += imag2;

    }

};

**class** Friend {

**public**:

**static** **void** calculateDifference(Complex c1, Derived c2) {

**float** realDiff = c1.getReal() - c2.getReal();

**float** imagDiff = c1.getImag() - c2.getImag();

        cout << "Difference of complex numbers: " << realDiff << " + " << imagDiff << "i" << endl;

    }

};

**int** main() {

    Derived d;

    d.display();

    Complex c;

    c.display();

    Friend::calculateDifference(c, d);

**return** 0;

}

Q5. Write a c++ program to demonstrate a pure virtual function which is invoked

from the object of derived class through the pointer of the base class. Base class

Person contains getdata() and display().

Display the information of employee using this.

#include<iostream>

**using** **namespace** std;

**class** base

{

**protected**:

**int** eid;

**char** ename[30],desig[50];

**float** salary; **public**:

**void** **virtual** getdata()

 {

 cout<<"\n Get data in base class";

 }

**void** **virtual** display()

 {

 cout<<"\nDisplaying in base class";

 }

};

**class** derived:**public** base

{ **public**: **void**

getdata()

 {

 cout<<"\nEnter employee id:";

cin>>eid;

    cout<<"\nEnter employeename:";

    cin>>ename;

    cout<<"\nEnter employee designation:";

    cin>>desig;

cout<<"\nEnter employee salary:";

cin>>salary;

 }

**void** display()

 {

 cout<<"\nEmployee details are:";

cout<<"\nEmployee Id:"<<eid;

     cout<<"\nEmployee name:"<<ename;

     cout<<"\nEmployee designation:"<<desig;

     cout<<"\nEmployee salary:"<<salary;

 }

};

**int** main()

{

base \*ptr;

derived D;

ptr=&D;

    ptr->getdata();

ptr->display();

**return** 0;

}

Q21

1.inline function to obtain the largest 3 numbers

2. Describe class template to carry out bubble sorting technique

#include<iostream>

**using** **namespace** std;

**void** findLargenum(**int** a, **int** b, **int** c)

{

**if**(a > b && a> c)

 cout<<"A is largest"<<endl;

**else** **if**(b > a && b > c)

 cout<<"B is largest"<<endl;

**else**

 cout<<"C is largest"<<endl;

}

**int** main()

{

**int** num1 = 0, num2 = 0, num3 = 0;

cout<<"\nEnter first number: ";

cin>>num1;

cout<<"\nEnter second number: ";

cin>>num2;

cout<<"\nEnter third number: ";

cin>>num3;

findLargenum(num1, num2, num3);

**return** 0;

}