Q.15 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)

Solution:

#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;

}

Q.16 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.

Solution:

#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 )

Solution:

**#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;**

**}**

**Q.19** Create a class String and define necessary members and constructors to overload '+' to concatenate two strings and display it.

Solution:

**#include <iostream>**

**#include <string>**

**using namespace std;**

**class String {**

**private:**

**string value;**

**public:**

**String(const string& value) : value(value) {}**

**String operator+(const String& other) const {**

**return String(value + other.value);**

**}**

**friend ostream& operator<<(ostream& os, const String& str);**

**};**

**ostream& operator<<(ostream& os, const String& str) {**

**os << str.value;**

**return os;**

**}**

**int main() {**

**String s1("Hello");**

**String s2(" world!");**

**String result = s1 + s2;**

**cout << result << endl;**

**return 0;**

**}**

**Q.20** Demonstrate the use of static variable and static function

#include<iostream>

using namespace std;

//Static variables and Static Function

class student

{

private:

static int x;

string name;

int age;

public:

student(string n,int a)

{

x++;

name=n;

age=a;

}

void print()

{

cout<<"Name : "<<name<<endl;

cout<<"Age : "<<age<<endl;

cout<<"**\n**";

}

static int getCount()

{

return x;

}

};

//Static Variable

int student::x=0;

int main()

{

student s1("Ram",25);

student s2("Sam",22);

student s3("Kumar",12);

s1.print();

s2.print();

s3.print();

cout<<"Total Students : "<<student::getCount()<<endl;

return 0;

}

**Q.21** 1.inline function to obtain the largest 3 numbers 2. Describe class template to carry out bubble sorting technique

1. Solution:  
   Here's an example C++ program with an inline function to obtain the largest three numbers:

**#include <iostream>**

**#include <algorithm>**

**using namespace std;**

**inline void largestThree(int arr[], int n) {**

**sort(arr, arr+n, greater<int>());**

**for(int i=0; i<3; i++) {**

**cout << arr[i] << " ";**

**}**

**}**

**int main() {**

**int arr[] = {5, 1, 8, 2, 7, 3, 9, 4, 6};**

**int n = sizeof(arr) / sizeof(arr[0]);**

**cout << "Largest three numbers: ";**

**largestThree(arr, n);**

**return 0;**

**}**

Here's an example class template in C++ that carries out the bubble sorting technique:

**#include <iostream>**

**#include <vector>**

**using namespace std;**

**template<typename T>**

**class BubbleSort {**

**public:**

**void sort(vector<T>& arr) {**

**int n = arr.size();**

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

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

**if(arr[j] > arr[j+1]) {**

**swap(arr[j], arr[j+1]);**

**}**

**}**

**}**

**}**

**};**

**int main() {**

**vector<int> arr = {5, 1, 8, 2, 7, 3, 9, 4, 6};**

**BubbleSort<int> bs;**

**bs.sort(arr);**

**cout << "Sorted array: ";**

**for(auto x: arr) {**

**cout << x << " ";**

**}**

**return 0;**

**}**

**Q.22** Inherit Manager and Scientists class with Salary 1 and Salary 2. Find max salary and display them using friend function

Solution:

#include <iostream>

using namespace std;

class Employee {

protected:

int salary;

public:

Employee(int s) : salary(s) {}

};

class Manager : public Employee {

public:

Manager(int s) : Employee(s) {}

friend int max\_salary(const Manager& m, const Scientist& s);

};

class Scientist : public Employee {

public:

Scientist(int s) : Employee(s) {}

friend int max\_salary(const Manager& m, const Scientist& s);

};

int max\_salary(const Manager& m, const Scientist& s) {

return max(m.salary, s.salary);

}

int main() {

Manager m(5000);

Scientist s(6000);

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

return 0;

}

**Q.23** write a program for reading and writing data to and from file using command line arguments

Solution:

Same as question 27

**Q.24 Write a C++ program to concatenate two Strings using operator + function. (using operator overloading)**

Solution:  
// C++ Program to concatenate two string

// using unary 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;

}

Q.25 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;

}

**Q.26** 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

Solution:

#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;

}

**Q.27 Write a program for reading and writing data to and from the file using command line arguments.**

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

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

if (argc < 3) {

cout << "Usage: " << argv[0] << " <filename> [-r|-w] [data]" << endl;

return 1;

}

string filename = argv[1];

if (strcmp(argv[2], "-r") == 0) {

// Read data from file

ifstream infile(filename);

if (infile) {

string data((istreambuf\_iterator<char>(infile)), (istreambuf\_iterator<char>()));

cout << "Data from " << filename << ":" << endl;

cout << data << endl;

} else {

cerr << "Error: could not open file " << filename << endl;

return 1;

}

} else if (strcmp(argv[2], "-w") == 0) {

// Write data to file

if (argc < 4) {

cerr << "Error: no data provided" << endl;

return 1;

}

string data = argv[3];

ofstream outfile(filename);

if (outfile) {

outfile << data;

cout << "Wrote data to " << filename << endl;

} else {

cerr << "Error: could not open file " << filename << endl;

return 1;

}

} else {

cout << "Usage: " << argv[0] << " <filename> [-r|-w] [data]" << endl;

return 1;

}

return 0;

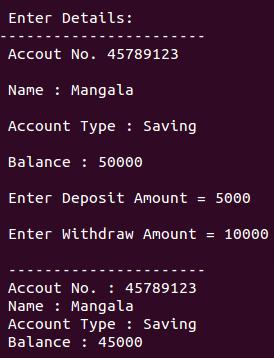
}

**Q.28 . 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.**

Solution:

#include<iostream>  
#include<stdio.h>  
#include<string.h>  
  
using namespace std;  
  
class bank  
{  
        int acno;  
        char nm[100], acctype[100];  
        float bal;    
   public:  
        bank(int acc\_no, char \*name, char \*acc\_type, float balance)  //Parameterized Constructor  
        {  
                acno=acc\_no;  
                strcpy(nm, name);  
                strcpy(acctype, acc\_type);  
                bal=balance;  
        }  
        void deposit();  
        void withdraw();  
        void display();  
};  
void bank::deposit()   //depositing an amount  
{  
        int damt1;  
        cout<<"\n Enter Deposit Amount = ";  
        cin>>damt1;  
        bal+=damt1;  
}  
void bank::withdraw()  //withdrawing an amount  
{  
        int wamt1;  
        cout<<"\n Enter Withdraw Amount = ";  
        cin>>wamt1;  
        if(wamt1>bal)  
                cout<<"\n Cannot Withdraw Amount";  
        bal-=wamt1;  
}  
void bank::display()  //displaying the details  
{  
        cout<<"\n ----------------------";  
        cout<<"\n Accout No. : "<<acno;  
        cout<<"\n Name : "<<nm;  
        cout<<"\n Account Type : "<<acctype;  
        cout<<"\n Balance : "<<bal;    
}  
int main()  
{  
        int acc\_no;  
        char name[100], acc\_type[100];  
        float balance;  
        cout<<"\n Enter Details: \n";  
        cout<<"-----------------------";  
        cout<<"\n Accout No. ";  
        cin>>acc\_no;  
        cout<<"\n Name : ";  
        cin>>name;  
        cout<<"\n Account Type : ";  
        cin>>acc\_type;  
        cout<<"\n Balance : ";  
        cin>>balance;  
    
        bank b1(acc\_no, name, acc\_type, balance);  //object is created  
        b1.deposit(); //  
        b1.withdraw(); // calling member functions  
        b1.display(); //  
        return 0;  
}

OUTPUT:



**Q.29 Write a program by using friend class to show swapping of data**

Solution:

#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;

}

OUTPUT:

Before Swapping: 4 6

After Swapping: 6 4