Agnim Gupta

2028083

A23, CSSE

Question 1

What is an exception? Can we throw class type exceptions? Explain with the help of an example. Write a program to accept 10 integers in an array. Check all numbers in the array. When any negative number is found, throw an exception.

Exception allows a method to handle exceptional conditions and errors within the program by transferring control to special programs called handlers. Yes we can throw class type exceptions.

```
#include <iostream>
using namespace std;
class demo {
};

int main()
{
	try {
		throw demo();
	}
	catch (demo d) {
		cout << "Caught exception of demo class \n";
	}
}
```

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

int main()
{
	int n{0};
	cout << "n : ";
	cin >> n;
	try
	{
		if (n < 10)
		{
			throw n;
		}
	}
	catch (int x)
	{
		printf("Exception caught! (n = %d)\n", x);
	}
	return 0;
}
```

Question 2

When do we make a virtual function “pure” ?What are the implications of making function a pure virtual function ? Write a program to sort an array of integers using a function pointer in descending order and resort this array in ascending order using virtual function.

When we want our class to be an abstract base class, we declare a virtual function pure (an abstract data type). Pure virtual functions must be overridden by a derived class or one of its derivatives, unlike virtual functions (the function remains pure virtual until it is overridden, at which point it becomes virtual). Derived classes that lack a complete implementation for all of the pure virtual functions they inherit become abstract. Derivation is the only way to instantiate an abstract base class. A pure virtual function is a base class function that a derived class must override. A default implementation of the method does not need to be provided by the underlying class. Furthermore, the base class is transformed into an abstract base class, which means that no object of that class can be instantiated other than through derivation. An abstract base class is a derived class that does not override a pure virtual function. The pure virtual function becomes an ordinary virtual function with respect to all other derivates of that class once it is overridden by a derived class. A virtual function, on the other hand, is a base class function that derived classes are expected to override. The base class must have a default implementation, and it can be instantiated as a standalone object. A virtual destructor must be declared by any class that declares at least one virtual function or at least one pure virtual function. This assures that the most-derived destructor is always invoked first, even if the most-derived class is unknown to the programmer, anytime an object of the class falls out of scope or is explicitly destroyed by a reference or pointer to one of its base classes. This guarantees that the object's class hierarchy is destructed in the order in which it was created.

```
#include<iostream>
using namespace std;
class base
{
	public:
	virtual void dsort(int *a,int s)
	{
		int i, j, temp;
		for(i=0;i<s;i++)
		{
			for(j=i+1;j<s;j++)
			{
				if( *(a+i) < *(a+j) )
				{
					temp = *(a+i);
					*(a+i) = *(a+j);
					*(a+j) = temp;
				}
			}
		}
	}
};
class derived:public base
{
	public:
	void asort(int *x,int n)
	{
		int i, j, temp;
		for(i=0;i<n;i++)
		{
			for(j=i+1;j<n;j++)
```

```
                {
                        if( *(x+i) > *(x+j) )
                        {
                                temp = *(x+i);
                                *(x+i) = *(x+j);
                                *(x+j) = temp;
                        }
                }
        }
    }
};
int main()
{
    base *bptr;
    derived d;
    bptr = &d;
    int num=0;

    printf("\nEnter size of array:");
    scanf("%d",&num);

    int a[num], i;
    int *pa;
    pa = &a[0];

    printf("\nEnter array elements: ");

    for(i=0;i<num;i++) {
        scanf("%d",pa+i);
    }
    bptr->dsort( &a[0],num);

    printf("\nSorted descending order array is: ");

    for(i=0;i<num;i++) {
        printf(" %d", *(pa+i));
    }
    d.asort( &a[0],num);

    printf("\nSorted ascending order array is: ");
```

```
        for(i=0;i<num;i++) {
                printf(" %d", *(pa+i));
        }
        return 0;
}
```

Question 3

What is abstract Polymorphism? Write a program to declare a function show() in base and derived class .Display message through the function to know name of class whose member function is executed. Use late binding concept using virtual keyword.

Polymorphism means reusing something in different ways/types to get different results. Abstraction and polymorphism can be combined to get for example a class which is both abstract and polymorphic. Implementation of polymorphism using the concept of abstract class is known as abstract polymorphism.

```
#include<iostream>
using namespace std;
class Base
{
        public:
        void print()
        {
                cout<<"\n PRINT - BASE CLASS ";
        }
        virtual void show()
        {
                cout<<"\n SHOW - BASE CLASS ";
        }
};
class Derived : public Base
{
        public:
        void print()
        {
                cout<<"\n PRINT - DERIVED CLASS ";
        }
        void show()
        {
                cout<<"\n SHOW - DERIVED CLASS ";
        }
};
int main()
{
        Base B, *bptr;
        Derived D;
        bptr = &B;
        bptr->print();
        bptr->show();
        bptr = &D;
        bptr->print();
        bptr->show();
        return 0;
}
```