

# OOP Assignment-1

Prepared by: Sethi Jyotir Aditya

Roll no: 20051222

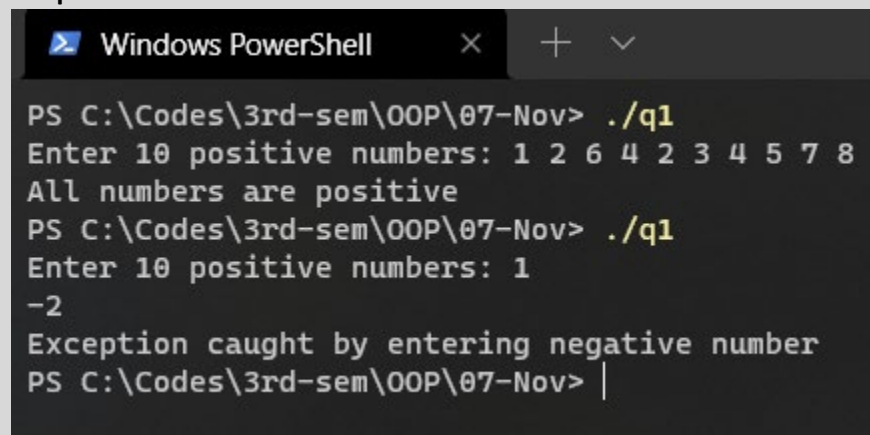
1. **What is an exception? Can we throw class type exceptions? Explain with the help of an example. Write a program to accept 10 integers in an array. Check all numbers in the array. When any negative number is found, throw an exception.**

Exception is an unexpected situation in which instead of the program executing normally, there is abnormal execution of the program which may cause the program to terminate.

Yes, we can throw class type exception. The following program shows how it can be done. Here in the following program, we perform the simple division of two numbers. We throw an exception when we detect division by 0.

```
#include <iostream>
using namespace std;
int main()
{
    int arr[10];
    printf("Enter 10 positive numbers: ");
    for (int i = 0; i < 10; i++)
    {
        scanf("%d", &arr[i]);
        try
        {
            if (arr[i] < 0)
                throw(arr[i]);
        }
        catch (int j)
        {
            cout << "Exception caught by entering negative number";
            return -1;
        }
    }
    cout << "All numbers are positive";
    return 0;
}
```

Output:



```
Windows PowerShell
PS C:\Codes\3rd-sem\OOP\07-Nov> ./q1
Enter 10 positive numbers: 1 2 6 4 2 3 4 5 7 8
All numbers are positive
PS C:\Codes\3rd-sem\OOP\07-Nov> ./q1
Enter 10 positive numbers: 1
-2
Exception caught by entering negative number
PS C:\Codes\3rd-sem\OOP\07-Nov> |
```

2. When do we make a virtual function “pure”? What are the implications of making a 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.

A virtual function is made pure virtual function when we append “= 0” at the end of a virtual function. In practical applications, the member functions of base class are rarely used for performing any task hence such type of functions are called as do-nothing functions or dummy functions or pure virtual functions. Also, a pure virtual function does not have a body or implementation.

```
#include <iostream>
using namespace std;
class a
{
public:
    virtual void sort_ascending(int *arr, int n)
    {
        int i, temp, j;
        for (i = 0; i < n; i++)
        {
            for (j = i + 1; j < n; j++)
            {
                if (arr[i] > arr[j])
                {
                    temp = arr[i];
                    arr[i] = arr[j];
                    arr[j] = temp;
                }
            }
        }

        cout << "\n** Array sorted in ASCENDING order (in class a) **" << endl;
        for (i = 0; i < n; i++)
        {
            cout << arr[i] << " ";
        }
    }
};
```

```

class b : public a
{
public:
    void sort_ascending(int *arr, int n)
    {
        int i, temp, j;
        for (i = 0; i < n; i++)
        {
            for (j = i + 1; j < n; j++)
            {
                if (arr[i] > arr[j])
                {
                    temp = arr[i];
                    arr[i] = arr[j];
                    arr[j] = temp;
                }
            }
        }

        cout << "\n** Array sorted in ASCENDING order (in class b) **" << endl;
        for (i = 0; i < n; i++)
        {
            cout << arr[i] << " ";
        }
    }
};

void sort_des(int *arr, int n)
{
    int i, temp, j;
    for (i = 0; i < n; i++)
    {
        for (j = i + 1; j < n; j++)
        {
            if (arr[i] < arr[j])
            {
                temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
    }

    cout << "\n** Array sorted in DESCENDING order (using function pointer) **" <<
endl;
    for (i = 0; i < n; i++)
    {
        cout << arr[i] << " ";
    }
}

```

```

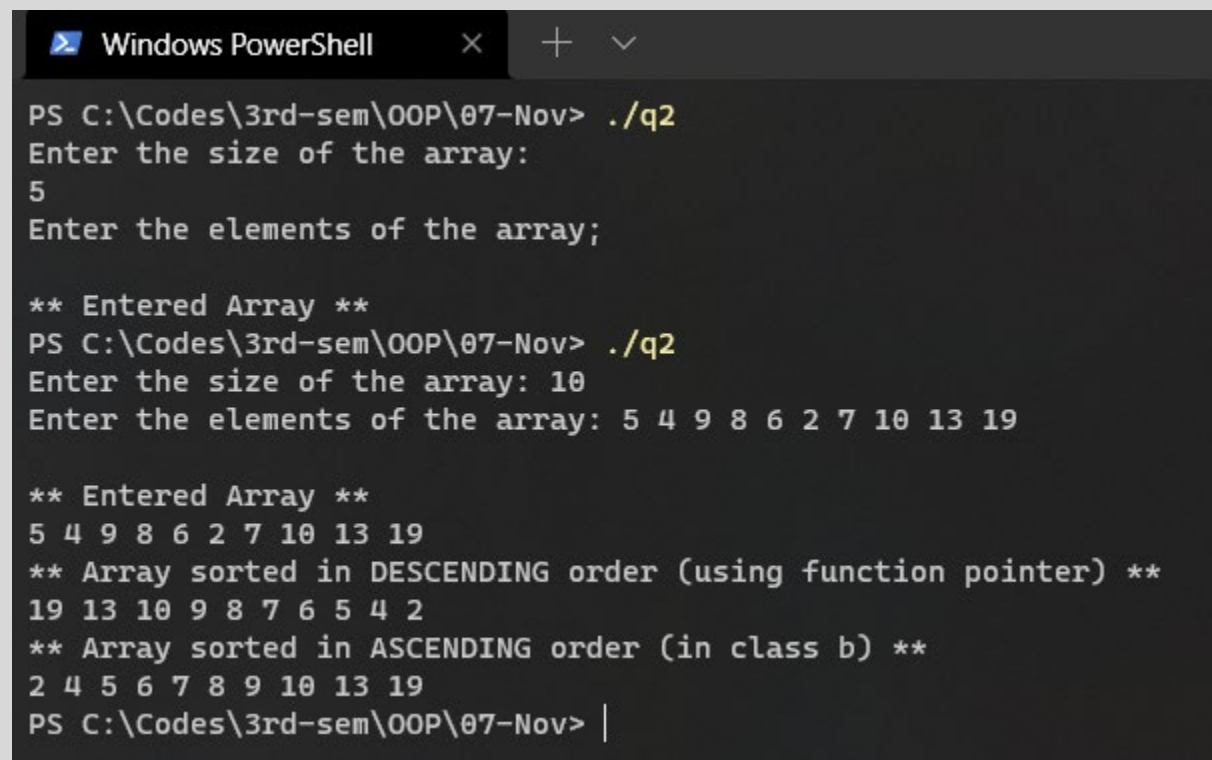
int main()
{
    int n, i;
    a a1, *ptrA;
    b b1;

    cout << "Enter the size of the array: " << endl;
    cin >> n;
    int arr[n];
    cout << "Enter the elements of the array: " << endl;
    for (i = 0; i < n; i++)
    {
        cin >> arr[i];
    }

    cout << "\n** Entered Array **" << endl;
    for (i = 0; i < n; i++)
    {
        cout << arr[i] << " ";
    }
    void (*fptr)(int *, int){&sort_des};
    fptr(arr, n);
    ptrA = &b1;
    ptrA->sort_ascending(arr, n); // sorting in ascending order using virtual function
    return 0;
}

```

Output:



```

Windows PowerShell
PS C:\Codes\3rd-sem\OOP\07-Nov> ./q2
Enter the size of the array:
5
Enter the elements of the array;

** Entered Array **
PS C:\Codes\3rd-sem\OOP\07-Nov> ./q2
Enter the size of the array: 10
Enter the elements of the array: 5 4 9 8 6 2 7 10 13 19

** Entered Array **
5 4 9 8 6 2 7 10 13 19
** Array sorted in DESCENDING order (using function pointer) **
19 13 10 9 8 7 6 5 4 2
** Array sorted in ASCENDING order (in class b) **
2 4 5 6 7 8 9 10 13 19
PS C:\Codes\3rd-sem\OOP\07-Nov> |

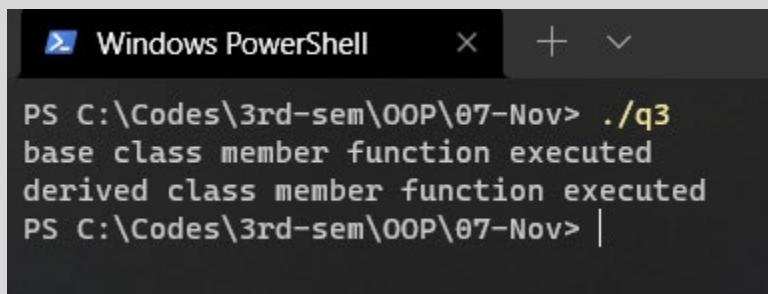
```

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.

Abstract Polymorphism is a method of declaring a abstract class or function that can be used as a base class or function. As we know that pure virtual functions are called do-nothing functions since they don't have any task to perform and have no practical use. We use the concept of polymorphism to create a function of same name as the pure virtual function in the derived classes. We then use the derived class to perform the task accordingly. This is abstract polymorphism.

```
#include <iostream>
using namespace std;
class base
{
public:
    virtual void show()
    {
        cout << "base class member function executed" << endl;
    }
};
class derived : public base
{
public:
    void show()
    {
        cout << "derived class member function executed" << endl;
    }
};
int main()
{
    base b, *p;
    derived d;
    p = &b;
    p->show(); // Early Binding
    /*** Condition: When base class member function " show() " is not made virtual
    -----
    Here Early Binding Occurs, i.e during compile time only the compiler knows which function
    to execute since it identifies the type of pointer, which is of type base class.
    Hence it executes the show function of base class
    */
    p = &d;
    p->show(); // Late Binding
    /*** Condition: Now base class member function " show() " is made virtual
    -----
    Here late binding occurs since base class member function " show() " is virtual.
    The pointer 'p' of type 'base' stores the address of object 'd' of 'derived' class
    during run time since memory is allocated during runtime. Hence here late binding occurs
    */
    return 0;
}
```

Output:



```
Windows PowerShell
PS C:\Codes\3rd-sem\OOP\07-Nov> ./q3
base class member function executed
derived class member function executed
PS C:\Codes\3rd-sem\OOP\07-Nov> |
```