Session 6

Week -2

Exception Handling Example

Example -1

1) Following is a simple example to show exception handling in C++. The output of program explains flow of execution of try/catch blocks.

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

int main()
{
    int x = -1;

    // Some code
    cout << "Before try \n";
    try {
        cout << "Inside try \n";
        if (x < 0)
        {
            throw x;
            cout << "After throw (Never executed) \n";
        }
    }
    catch (int x) {
        cout << "Exception Caught \n";
    }

    cout << "After catch (Will be executed) \n";
    return 0;
}
```

Example-2

```
#include <iostream>
using namespace std;
int main() {
       cout << "Start\n";
       try { // start a try block cout << "Inside try block\n";
              throw 100; // throw an error cout << "This will not execute";
       }
       catch (int i) { // catch an error cout << "Caught an exception -- value is: ";
              cout << i << "\n";
       }
       cout << "End";

              return 0;
}
```

Example 3

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

```
int main()
{
	cout << "Start\n";
	try { // start a try block
		cout << "Inside try block\n"; throw 100; // throw an error
		cout << "This will not execute";
	}
	catch (float  i) { // won't work for an int exception
		cout << "Caught an exception -- value is: ";
			cout << i << "\n";
	}
	cout << "End";
	return 0;
}
```

Example 4

```
void Xtest(int test) {
	cout << "Inside Xtest, test is: " << test << "\n";
	if (test)
		throw test;
}
int main()
{
	cout << "Start\n";
	try { // start a try block
		cout << "Inside try block\n";
		Xtest(0);
		Xtest(1);
		Xtest(2);
	}
	catch (int i) { // catch an error
			cout << "Caught an exception -- value is: ";
		cout << i << "\n";
	}
	cout << "End";
	return 0;
}
```

Example 5(Different Approach)

```
// Localize a try/catch to a function.
void Xhandler(int test) {
	try {
		if (test) throw test;
	}
	catch (int i) {
		cout << "Caught Exception #: " << i << '\n';
	}
}
int main()
{
	cout << "Start\n";
	Xhandler(1);

	Xhandler(0);
```

```
        Xhandler(3);
        cout << "End";
        return 0;
}
```

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

int main()
{
    try {
        throw 10;
    }
    catch (char* excp) {
        cout << "Caught " << excp;
    }
    catch (...) {
        cout << "Default Exception\n";
    }
    return 0;
}
```

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

int main()
{
    try  {
       throw 'a';
    }
    catch (int x)  {
        cout << "Caught " << x;
    }
    catch (...)  {
        cout << "Default Exception\n";
    }
    return 0;
}
```

Task

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

class Test {
public:
    Test() { cout << "Constructor of Test " << endl; }
    ~Test() { cout << "Destructor of Test " << endl; }
};

int main()
{
    try {
        Test t1;
        throw 10;
    }
    catch (int i) {
        cout << "Caught " << i << endl;
    }
}
```

Exceptional Handling in 1DArray

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

class Test {
public:
    Test() { cout << "Constructor of Test " << endl; }
    ~Test() { cout << "Destructor of Test " << endl; }
    void test(int* p)
    {
        try {
            throw 10;
        }
        catch (int i) {
            cout << "Caught " << i << endl;
        }


    }
};

int main()
{
    int a[5] = { 1,2,3,4,5 };
        Test t1;
        t1.test(a);
}
```

Problem Discussion

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

int main()
{
    int foo[] = { 16, 2, 77, 40, 12071 };
    int size = sizeof(foo) / sizeof(int);
    try {
        //throw foo;
        throw (pair<int*, int>(foo, size));
    }
    catch (int* foo)
    {
        for (int i = 0; i < size; i++)
            cout << foo[i] << ",";
    }
    catch (pair<int*, int>& ip)
    {
        cout << "pair.." << endl;
        for (int i = 0; i < ip.second; i++)
            cout << ip.first[i] << ",";
    }
    cout << endl;
}
```

Class Task

Finding Duplicate value in Unsorted array

Input

```
{ 6, 6, 7, 6, 9, 1, 9, 0, 0, 1, 4, 5, 1 }
```

Output

```
{ 6, 7, 9, 1, 0, 4, 5}
```

Algorithm Discussion –Convert This task to 1-D SafeArray

```cpp
#include <iostream>
#include <vector>
using namespace std;
void remove_duplicates(int* arr, int& n) {

	// temporary list to store all the unique elements
	vector<int> temp;

	// iterate each element of arr[]
	for (int i = 0; i < n; ++i) {

		// checking if there exist an element arr[j] ( j < i )
		// that is equal to arr[i]
		int flag = 1;
		for (int j = 0; j < i; ++j) {
			if (arr[j] == arr[i]) {
				flag = 0;
				break;
			}
		}

		// flag == 0 means arr[i] is repeated
		// flag == 1 means that no element that appears
		// on left side of arr[i] is equal to arr[i]
		// therefore, we push arr[i] to temp
		if (flag)
			temp.push_back(arr[i]);

	}

	// after the completion of above loop
	// temp will contain all the unique elements in arr[] without repetition
	// size of temp may be smaller than the original array
	// so we set n = temp.size() and overwrite arr[] with temp
	n = temp.size();
	for (int i = 0; i < n; ++i)
		arr[i] = temp[i];

}

int main() {

	int arr[] = { 6, 6, 7, 6, 9, 1, 9, 0, 0, 1, 4, 5, 1 };
	int n = sizeof(arr) / sizeof(int);

	cout << "Input Array: ";
	for (int i = 0; i < n; ++i)
		cout << arr[i] << ' ';
	cout << endl;

	remove_duplicates(arr, n);

	cout << "Output Array: ";
	for (int i = 0; i < n; ++i) cout << arr[i] << ' ';
	cout << endl;
}
```

Resize 1D Array {Implement this Task with 1D Safe Array}

```
void resize() {
    size_t newSize = size * 2;
    int* newArr = new int[newSize];
    memcpy(newArr, arr, size * sizeof(int));
    size = newSize;
    delete[] arr;
    arr = newArr;
}
```

2D Array

2D Safe Array

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

class atype {
    int ncols; int nrows;
    int** dynamicArray;
public: atype()
{
    nrows = 0; ncols = 0; dynamicArray = 0;
}
      //constructor
      atype(int row, int col)
      {
          nrows = row;
          ncols = col;
          dynamicArray = new int* [nrows];
          for (int i = 0; i < nrows; i++)
          {
              dynamicArray[i] = new int[ncols];
          }

      }

      void print()
      {
          for (int in = 0;in < nrows;in++)
          {
              for (int j = 0;j < ncols;j++)
              {
                  int value; cout << "enter values"; cin >> value;
                  dynamicArray[in][j] = value;
              }
          }


      }

      int& operator ()(int i, int j)
      {
          if (i<0 || i> nrows - 1 || j<0 || j> ncols - 1)
          {
              cout << "Boundary Error\n"; exit(1);
          }
          return dynamicArray[i][j];
```

```
    }
    //copy constructor
    atype(const atype& rhs)
    {
        nrows = rhs.nrows; ncols = rhs.ncols; dynamicArray = new int* [nrows];
        for (int i = 0; i < nrows; i++) {
            dynamicArray[i] = new int[ncols];
            memcpy(dynamicArray[i], rhs.dynamicArray[i], sizeof(int) * ncols);
        }
    }
    //assignment operator overloading
    atype& operator=(const atype& rhs) {
        if (this == &rhs)
            return *this;
        for (int i = nrows - 1; i >= 0; i--)
        {
            delete dynamicArray[i];
        }
        delete[] dynamicArray;
        nrows = rhs.nrows; ncols = rhs.ncols;
        dynamicArray = new int* [nrows];
        for (int i = 0; i < nrows; i++)
        {
            dynamicArray[i] = new int[ncols];
            memcpy(dynamicArray[i], rhs.dynamicArray[i], sizeof(int) * ncols);
        }
        return
            *this;
    }
    //not equal to operator overloading
    atype& operator!=(const atype& rhs)
    {
        for (int i = 0;i < nrows;i++)
        {
            for (int j = 0;j < ncols;j++)
            {
                if (dynamicArray[i][j] != rhs.dynamicArray[i][j])
                {
                    cout << "not equal"; break;
                }
            } break;
        }
    }
};



int main()
{
    atype b1(2,2);
    b1.print();
    atype ob2 = b1;
    atype ob3(3, 3);
    cout << b1(3,3) << endl;
    cout << ob3(4, 4) << endl; //checking bounds of array

}
```

Jaged Array Concept

```
#include <iostream>
using namespace std;
int main (){
    int rows;
    cout << "Enter no of rows of array: ";
    cin >> rows;
    int* numbers = new int[rows]; /// array to store no of columns
    int** array = new int* [rows];   /// jagged array

    for (int i = 0; i < rows; i++) {

        cout << "Enter no of col in row" << i << ": ";
        cin >> numbers[i];
        array[i] = new int[numbers[i]];
    }

        for (int i = 0; i < rows; i++) {

            for (int j = 0; j < numbers[i]; j++) {

                cout << "Rows " << i << ": Enter value" << i * numbers[i] + j << ": ";
                cin >> array[i][j];
            }
        }
    cout << "Showing all the Inputed data in a matrix form" << endl;
    for (int i = 0; i < rows; i++) {

        for (int j = 0; j < numbers[i]; j++) {
            cout << array[i][j] << " ";

        }
        cout << "\n";
    }

    cout << "Dellocating the array..." << endl;
    for (int i = 0; i < rows; i++) {
        delete[] array[i];

    }
    delete[] array;

    cout << "done!";
}
```

Write a program that creates a **2D array** of **5x5** values of **type boolean**. Suppose indices represent people and that the value at **row i**, **column j** of a **2D array** is true just in case **i and j** are **friends** and **false otherwise**. Use initializer list to instantiate and initialize your array to represent the following configuration: (*** means "friends"**)

	0	1	2	3	4
0		*		*	*
1	*		*		*
2		*			
3	*				*
4	*	*		*	

Write a method to check whether **two people** have a **common friend**. For example, in the example above, **0 and 4** are **both friends with 3** (so they have a common friend), whereas **1 and 2** have **no common friends**.

```
#include<iostream>
using namespace std;
int main()
{
	char a[3][3] = { {'0','x','x'},{'x','0','x'},{'0','x','x'}};

	for (int i = 0; i <= 2; i++)
	{
		for (int j=0; j<=2; j++ )
			{
			cout << " " << "" << a[i][j]<<" ";
		}
		cout << "" << endl;
	}

}

void test()
{
```

}

Specify the size of columns of 2D array

```
void processArr(int a[][10]) {
   // Do something
}
```

Pass array containing pointers

```
void processArr(int *a[10]) {
   // Do Something
}

// When callingint *array[10];
for(int i = 0; i < 10; i++)
   array[i] = new int[10];
processArr(array);
```

Pass a pointer to a pointer

```
void processArr(int **a) {
   // Do Something
}
// When calling:
int **array;
array = new int *[10];
for(int i = 0; i <10; i++)
   array[i] = new int[10];
processArr(array);
```

Algorithm Discussion

```
void test(char b[][3])
{
	int i;
	int a1[3], b1[3], c1[3] = {0};
	cout << "Connected Person"<< endl;
	for (int i = 0; i <= 2; i++) //Working to create list of Connected Ids
	{
		for (int j = 0; j <= 2; j++)
		{
			if (i==0)
			{
				if (b[i][j] == '-')
				{
					a1[j] = j;
					cout << " " << "" << a1[j];
				}
				else
					a1[j] = 0;
			}
			if (i == 1)
			{
				if (b[i][j] == '-')
				{
					b1[j] = j;
					cout << " " << "" << b1[j];
				}
				else
					b1[j] = 0;
			}
			if (i == 2)
			{
				if (b[i][j] == '-')
				{
					c1[j] = j;
					cout << " " << "" << c1[j];
				}
				else
					c1[j] = 0;
			}
		}
		cout << "" << endl;
	}
	cout << "Common Friend" << endl;
	for (int k = 0; k <=2; k++)
	{
			for (int h = 0; h <=2; h++)
			{
				if (a1[h] == b1[h])
				{
					cout << "IS" << h << endl;
				}

				cout << "Common Friend OF 0->" << h <<"Nothing" <<endl;
			}
```

```
        }
}
```

Sample Question

Question No. 2 Given this skeleton class and partial implementation; provide the implementation for the commented functions with question marks (?).

```
class Date{

   private:

 int *DateData; //day/month/year

  public:

  // default constructor

  Date() {

   DateData= new int[3];

     *(DateData+0)=0;

     *(DateData+1)=0;

    *(DateData+2)=0; }

  Date(const int a[]) {

   DateData= new int [3]   ;
for( int i=0; i < 3; i++)

  { *(DateData+i)=*(a+i);

  }  }

  Date(const Date & rhs) {

   DateData= new int [3]   ;

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

   { *(DateData+i)=
*(rhs.DateData+i); }  }

 ~ Date() {

     if(DateData != 0) {

    delete [] DateData; }  }

};  // class end
```

```
int main(){

Date D1, D2;

int a[3]= {10, 10, 1997};

cout << D1.getDay()  << D1.getMonth()
<< D1.getYear() << endl;

Date D3(a);

cout << D3.getDay()  << D3.getMonth()
<< D3.getYear() << endl;

return 0;


}
```

Question No. 2 Indicate TRUE or FALSE and explain in 2-3 lines to the argument on it.

a. **A derived class pointer can hold a base class object.**
b. **A virtual or pure virtual function can be private.**
c. **A destructor can be virtual? Give an example**