Lab 11. Exceptions

Example:

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

class exceptie1 : public exception
{
        virtual const char* what() const throw()
        {
                return "Impartire la 0!";
        }
};

class exceptie2 : public exception
{
        virtual const char* what() const throw()
        {
                return "Indexul este inafara domeniului!";
        }
};

int main()
{
        int z=-1, x = 50, y = 0;
        int arr[4] = { 0 };
        int i = 10;
        exceptie1 divide0;
        exceptie2 index_out_of_bounds;

        try
        {
                if (y==0)
                        throw divide0;
                z = x / y;
                cout << "Fara exceptie: z=" << z << endl;
        }
        catch (exception& e)
        {
                cout << "Exceptie: " << e.what() << endl;
        }

        try
        {
                if (i > 3)
                        throw index_out_of_bounds;
                arr[i] = z;
                for (i = 0; i < 4; i++)
                        cout << "arr[" << i << "] = " << arr[i] << endl;
        }
        catch (exception& e)
```

```
        {
                cout << "Exceptie: " << e.what() << endl;
        }

        return 0;
}
```

Problem 1: Build an array template (similar to the one below) and add exceptions to it.

```
class Compare

{

public:

 virtual int CompareElements(void* e1, void* e2) = 0;

};

template<class T>

class ArrayIterator

{

private:

 int Current; // mai adaugati si alte date si functii necesare
pentru iterator

public:

 ArrayIterator();

 ArrayIterator& operator ++ ();

 ArrayIterator& operator -- ();

 bool operator= (ArrayIterator<T> &);

 bool operator!=(ArrayIterator<T> &);

 T* GetElement();
```

```
};

template<class T>

class Array

{

private:

 T** List; // lista cu pointeri la obiecte de tipul T*

 int Capacity; // dimensiunea listei de pointeri

 int Size; // cate elemente sunt in lista

public:

 Array(); // Lista nu e alocata, Capacity si Size = 0

 ~Array(); // destructor

 Array(int capacity); // Lista e alocata cu 'capacity' elemente

 Array(const Array<T> &otherArray); // constructor de copiere



 T& operator[] (int index); // arunca exceptie daca index este out
of range



 const Array<T>& operator+=(const T &newElem); // adauga un
element de tipul T la sfarsitul listei si returneaza this

 const Array<T>& Insert(int index, const T &newElem); // adauga un
element pe pozitia index, retureaza this. Daca index e invalid
arunca o exceptie

 const Array<T>& Insert(int index, const Array<T> otherArray); //
adauga o lista pe pozitia index, retureaza this. Daca index e
invalid arunca o exceptie
```

```
 const Array<T>& Delete(int index); // sterge un element de pe
pozitia index, returneaza this. Daca index e invalid arunca o
exceptie



 bool operator=(const Array<T> &otherArray);



 void Sort(); // sorteaza folosind comparatia intre elementele din
T

 void Sort(int(*compare)(const T&, const T&)); // sorteaza
folosind o functie de comparatie

 void Sort(Compare *comparator); // sorteaza folosind un obiect de
comparatie



 // functii de cautare - returneaza pozitia elementului sau -1
daca nu exista

 int BinarySearch(const T& elem); // cauta un element folosind
binary search in Array

 int BinarySearch(const T& elem, int(*compare)(const T&, const
T&));//  cauta un element folosind binary search si o functie de
comparatie

 int BinarySearch(const T& elem, Compare *comparator);//  cauta un
element folosind binary search si un comparator



 int Find(const T& elem); // cauta un element in Array

 int Find(const T& elem, int(*compare)(const T&, const
T&));//  cauta un element folosind o functie de comparatie

 int Find(const T& elem, Compare *comparator);//  cauta un element
folosind un comparator
```

```
 int GetSize();
 int GetCapacity();

 ArrayIterator<T> GetBeginIterator();
 ArrayIterator<T> GetEndIterator();
};
```