Умова

Імітація черги на отримання житла. Клас “учасник черги” - прізвище, телефон, рік постановки в чергу. Неупорядкований масив “учасників черги” перетворити в стандартний контейнер queue. Побудувати функції:

-визначення довжини черги;

-додавання нового елементу в чергу (постановлення на облік нового учасника);

-вилучення учасників (по черзі потрапляння в чергу).

Передбачити виключні ситуації: дата постановки на облік більша за поточну; дата постановлення на облік не дорівнює поточній для нових учасників.

Порядок виконання роботи

Ознайомитись з основними можливостями, що надає бібліотека STL. Відповідно варіанту визначити необхідні стандартні контейнери STL, за потреби побудувати свій з поведінкою, ідентичною стандартному. Розв’язати задачу обов’язково з використанням стандартних алгоритмів (не менше двох) і операцій контейнерних класів STL. За умови доцільності, побудувати свій об’єкт-функцію. Ознайомитись з механізмом опрацювання виключних ситуацій.

У розробленій програмі передбачити обробку виключних ситуацій, означених в завданні або запропонувати свої, якщо такі у даному варіанті не конкретизовані.

Самостійно продумати і реалізувати спосіб демонстрації отриманих результатів.

Текст програми

QueueMember.h

```
#pragma once
#include <string>

using namespace std;

class QueueMember {

private:
	string name;
	string surname;
	int yearentry;

public:
	QueueMember();
	QueueMember(string name, string surname, int yearentry);
```

```
        friend ostream& operator <<(ostream & os, QueueMember& o);
        friend istream& operator >>(istream & is, QueueMember& o);

        int GetYearOfEntry();

};
```

QueueMember.cpp

```
#include "QueueMember.h"
#include <iostream>
using namespace std;

QueueMember::QueueMember() {
        name = "NoName";
        surname = "NoSurname";
        yearentry = 2000;
}

QueueMember::QueueMember(string name, string surname, int yearentry) {
        this->name = name;
        this->surname = surname;
        this->yearentry = yearentry;
}

ostream& operator <<(ostream & os, QueueMember& o) {
        os << "name : " << o.name << "\n";
        os << "surname : " << o.surname << "\n";
        os << "year_start : "<< o.yearentry << "\n";
        return os;
}


istream& operator >>(istream & is, QueueMember& o) {
        cout << "Enter Name : "; is >> o.name;
        cout << "Enter Surname : "; is >> o.surname;
        int temp;
        cout << "Enter Year"; is >> temp;
        if (temp>1500 && temp <2016) o.yearentry = temp;
        else o.yearentry = 0;
        return is;
}

int QueueMember::GetYearOfEntry() {
        return this->yearentry;
}
```

Queue.h

```
#pragma once
#define _CRT_SECURE_NO_WARNINGS
#include <cassert>
#include <iostream>
#include <iomanip>

using namespace std;

template <class T>
class Queue{
private:
	T *queuePtr;
	int begin;
	int	end;
	int size;
	int elementsCounter;
public:
	Queue(T arr[], int);
	~Queue();

	inline void Push( T&);
	inline int GetSizeOfTheQueue();
	inline T Delete();
	inline void Print();
};

template<class T>
Queue<T>::Queue(T arr[], int numberOfElements)
	: begin(0), end(0), size(10), elementsCounter(numberOfElements) {

	queuePtr = new T[size + 1];
	for (int i = 0; i < numberOfElements; i++) {
		queuePtr[end++] = arr[i];
	}

}

template<class T>
inline void Queue<T>::Push( T &newElement) {
	assert(elementsCounter < size);
	queuePtr[end++] = newElement;
	elementsCounter++;
	if (end > size)
		end -= size + 1;
}

template<class T>
inline T Queue<T>::Delete() {
	assert(elementsCounter > 0);
	T returnValue = queuePtr[begin++];
```

```
	elementsCounter--;
	if (begin > size)
		begin -= size + 1;
	return returnValue;
}

template<class T>
Queue<T>::~Queue() {
	delete[] queuePtr;
}

template<typename T>
inline void Queue<T>::Print()
{
	cout << "Queue: \n";
	if (end == 0 && begin == 0)
		cout << " Empty\n";
	else
	{
		for (int i = begin; i <end; i++)
			cout << queuePtr[i] << " ";
		cout << endl;
	}
}

template<typename T>
inline int Queue<T>::GetSizeOfTheQueue() {
	return elementsCounter;
}
```

Labwork4.cpp

```
#include <queue>
#include <iostream>
#include "QueueMember.h"
#include "Queue.h"

using namespace std;

int main() {

	QueueMember array_mem[3];
	array_mem[0] = QueueMember("Andrew", "Hawkings", 2012);
	array_mem[1] = QueueMember("Zlata", "Lilkavich", 2001);
	array_mem[2] = QueueMember("Alexander", "Zolochevsky", 1989);

	Queue<QueueMember> MyQueue(array_mem, 3);
	MyQueue.Print();

	try {
		string name;
		string surname;
```

```
		int year;
		cout << "\nEnter name : "; cin >> name;
		cout << "\nEnter surname : "; cin >> surname;
		cout << "\nEnter year : "; cin >> year;

		if (year>2016)
			throw (1);
		else if (year!=2016)
			throw ('z');

		QueueMember NewMember(name, surname, year);

		MyQueue.Push(NewMember);
		cout << "\n\nThe renewed Queue:\n ";
		MyQueue.Print();

	}
	catch (int) {cout << "More than 2016.\n";}
	catch (char) {cout << "Not 2016.\n";}
	catch (...) {cout << "Error. :)\n";}

	int n;
	cout << "\nEnter number of members to delete : "; cin >> n;
	for (int i = 0; i < n; i++)
		MyQueue.Delete();
		cout << "\n\nThe renewed Queue:\n ";
	MyQueue.Print();

	return 0;
}
```

Результати роботи програми

```
Queue:
name : Andrew
surname : Hawkings
year_start : 2012
 name : Zlata
surname : Lilkavich
year_start : 2001
 name : Alexander
surname : Zolochevsky
year_start : 1989


Enter name : Alex

Enter surname : Pissaro

Enter year : 2016
```

The renewed Queue:
 Queue:
name : Andrew
surname : Hawkings
year_start : 2012
 name : Zlata
surname : Lilkavich
year_start : 2001
 name : Alexander
surname : Zolochevsky
year_start : 1989
 name : Alex
surname : Pissaro
year_start : 2016

Enter number of members to delete : 2

The renewed Queue:
 Queue:
name : Alexander
surname : Zolochevsky
year_start : 1989
 name : Alex
surname : Pissaro
year_start : 2016