МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра ВТ

ОТЧЕТ

по лабораторной работе №6 (24 вариант)
по дисциплине «Алгоритмы и структуры данных»
Тема: использование стандартной библиотеки шаблонов

Студент гр. 7308 Замыслов Н.Ю.

Студент гр. 7308 Цебульский С.А.

Преподаватель Колинько П.Г.

Санкт-Петербург
2019

Цель работы

Получить практические навыки по работе с стандартной библиотекой шаблонов в C++.

Постановка задачи

Переделать программу, составленную при выполнении темы 5 «Последовательности», под использование контейнеров из стандартной библиотеки шаблонов. Для хранения множеств выбрать контейнер подходящего типа (*set* или *unordered_set* и т. п.) и доработать его для поддержки операций с последовательностями. Для реализации операций с контейнерами использовать возможности библиотеки алгоритмов. Программа должна реализовывать цепочку операций над множествами в соответствии с заданием по теме 3 (4) и операций с последовательностями — по теме 5. Результат каждого шага цепочки операций выводится на экран.

Реализованная программа

Для хранения множеств без повторений использовалась структура `set<int>`. Данная структура реализует красно-черное дерево с хранением только ключей, поэтому был выбран именно этот контейнер для хранения элементов множества.

Для операций над множествами использовались встроенные функции `set_union`, `set_intersection`, `set_difference`, `set_symmetric_difference`.

На экран выводятся 5 множеств. Далее последовательно результаты промежуточных операций над множествами.

```
A: 2 3
B: 1 2 3 4
C: 1 2 3 4
D: 1 2 3 4
E: 1 3 4

A or B XOR (C and D dif E)

A or B:
1 2 3 4

C and D:
1 2 3 4

C and D dif E:
2

result:
1 3 4
```

Рисунок 1 – Операции над множествами

Для реализации работы с последовательностями был разработан класс `mySeq`, который имеет поля `set<int>` для хранения элементов последовательности в виде дерева без повторений. А также поле `vector<set<int>::iterator>` для хранения итераторов на элементы дерева, задающих нужный порядок обхода дерева для вывода последовательности в исходном порядке.

Благодаря тому, что вектор итераторов при одинаковых элементах в последовательности хранит одинаковые итераторы на элементы дерева, отпала необходимость использовать шаблон `multiset<int>` для хранения множества с повторениями, что позволило сэкономить память.

На экран выводятся заданные множества в исходном порядке. Затем результаты операций erase, mul и excl над последовательностями. Перед выводом результата операции также выводится последовательность до применения операции для сравнения результата. В конце имеется вывод элементов последовательности, которые хранятся в контейнере `set<int>`, чтобы убедиться, что данная последовательность действительно хранится в дереве (причем без повторений).

```
S1: 3 1 2 1 4
S2: 1 1 3 2 1


erase S2 from 0 to 2:
S2: 1 1 3 2 1
new S2: 2 1
new Tree S2: 1 2


mul x1 S1:
S1: 3 1 2 1 4
new S1: 3 1 2 1 4 3 1 2 1 4
new Tree S1: 1 2 3 4


excl S2 from S1:
S1: 3 1 2 1 4 3 1 2 1 4
S2: 2 1
new S1: 3 1 4 3 1 4
new Tree S1: 1 3 4
```

Рисунок 2 – Операции над последовательностями

Вывод

В результате выполнения лабораторной работы были изучены некоторые контейнеры и встроенные операции над ними из стандартной библиотеки C++. Также выбранные контейнеры были доработаны для работы с последовательностями.

Благодаря удачному выбору контейнеров удалось сэкономить память при хранении последовательности с повторяющимися элементами.

Приложения

Source.cpp

```
#include <set>
#include <algorithm>
#include <iterator>
#include <conio.h>
#include <vector>
#include <ctime>
#include <iostream>


using namespace std;


const size_t SIZE = 4; // Мощность размещаемого в структурах множества
const size_t COUNT = 5; // Максимальное количество элементов в множествах


// Класс для работы с последовательностями
class mySeq {

public:

	vector<set<int>::iterator> seq;
	set<int> tree;


	// Конструктор с генерацией последовательности
	mySeq() {

		for (size_t i = 0; i < COUNT; ++i) {

			int temp = rand() % SIZE + 1;

			tree.insert(temp);

			seq.push_back(tree.find(temp));
		}
	}
```

```
// Вывод последовательности
void print() {

	for (size_t i = 0; i < seq.size(); ++i) {

		cout << *seq[i] << " ";
	}

	cout << endl;
}


// Перезапись дерева по элементам вектора
void treeRestart() {

	set<int> tempTree;
	vector<set<int>::iterator> tempSeq;
	for (size_t i = 0; i < seq.size(); ++i) {

		tempTree.insert(*seq.at(i));
		tempSeq.push_back(tempTree.find(*seq.at(i)));
	}


	swap(tempTree, tree);
	swap(tempSeq, seq);
}
```

```
	// Исключение из последовательности подпоследовательности с индекса left до right
	void erase(size_t left, size_t right) {

		if (left <= right) {

			set<int> tempSet;
			vector<set<int>::iterator> tempSeq;

			for (size_t i = 0; i < seq.size(); ++i) {

				if (!(i >= left && i <= right)) {

					tempSet.insert(*seq[i]);
					tempSeq.push_back(tempSet.find(*seq[i]));
				}
			}

			swap(tempSet, tree);
			swap(tempSeq, seq);

			treeRestart();
		}
	}


	// Добавление к последовательности в конец этой же последовательности count раз
	void mul(size_t count) {

		size_t tempSize = seq.size();

		for (size_t i = 0; i < count; ++i) {

			for (size_t j = 0; j < tempSize; ++j) {

				seq.push_back(seq[j]);
			}
		}
	}
```

```
	// Исключение из последовательности всех вхождений
подпоследовательности exclSeq
void excl(mySeq exclSeq) {

	if (seq.size() >= exclSeq.seq.size()) {

		int j = 0;

		for (int i = 0; i < seq.size(); ++i) {

			if (*seq.at(i) == *exclSeq.seq.at(j)) {
				++j;
			}
			else {
				i -= j;
				j = 0;
			}

			if (j == exclSeq.seq.size()) {

				for (int g = i; g > i - j; --g)
					seq.erase(seq.begin() + g);

				i -= j - 1;
				j = 0;

				if (seq.size() != 0 && i < seq.size())
					if (*seq.at(i) == *exclSeq.seq.at(j)) {
						++j;
					}
					else {
						i -= j;
						j = 0;
					}
			}
		}
	}

	treeRestart();
}

};
```

```
// Генерация случайного дерева
void generate(set<int> & seq) {

	for (size_t i = 0; i < COUNT; ++i) {

		seq.insert(rand() % SIZE + 1);
	}
}


// Вывод дерева
void print(set<int> & seq) {

	for (set<int>::iterator it = seq.begin(); it != seq.end();
++it) {

		cout << (*it) << " ";
	}

	cout << endl;
}
```

```
int main()
{
	srand(time(0));


	//.....................................................РЕШЕНИЕ ЦЕПОЧКИ ОПЕРАЦИЙ НАД МНОЖЕСТВАМИ
	set<int> A, B, C, D, E, temp1, temp2, temp3, result;


	// Генерация множеств
	generate(A);
	generate(B);
	generate(C);
	generate(D);
	generate(E);


	// Вывод множеств
	cout << "A: ";
	print(A);

	cout << "B: ";
	print(B);

	cout << "C: ";
	print(C);

	cout << "D: ";
	print(D);

	cout << "E: ";
	print(E);
```

```
	// Вывод промежуточных значений и результата цепочки операций
	cout << endl << "A or B XOR (C and D dif E)" << endl;
	set_union(A.begin(), A.end(), B.begin(), B.end(),
inserter(temp1, temp1.begin()));
	cout << endl << "A or B: " << endl;
	print(temp1);

	set_intersection(C.begin(), C.end(), D.begin(), D.end(),
inserter(temp2, temp2.begin()));
	cout << endl << "C and D: " << endl;
	print(temp2);

	set_difference(temp2.begin(), temp2.end(), E.begin(), E.end(),
inserter(temp3, temp3.begin()));
	cout << endl << "C and D dif E: " << endl;
	print(temp3);

	set_symmetric_difference(temp1.begin(), temp1.end(),
temp3.begin(), temp3.end(), inserter(result, result.begin()));
	cout << endl << "result: " << endl;
	print(result);
```

```
	//.....................................................ДЕМОНСТРАЦИЯ РАБОТЫ С ПОСЛЕДОВАТЕЛЬНОСТЯМИ
	mySeq S1, S2;
	size_t left, right; // Границы для операции erase
	size_t count;

	left = 0;
	right = 2;
	count = 1;


	// Вывод последовательностей
	cout << endl << endl << endl << "S1: ";
	S1.print();
	cout << "S2: ";
	S2.print();


	// Операции над последовательностями
	// ERASE
	cout << endl << endl << "erase S2 from " << left << " to " << right << ": " << endl;
	cout << "S2: ";
	S2.print();
	S2.erase(left, right);
	cout << "new S2: ";
	S2.print();
	cout << "new Tree S2: ";
	print(S2.tree);


	// MUL
	cout << endl << endl << "mul x" << count << " S1: " << endl;
	cout << "S1: ";
	S1.print();
	S1.mul(count);
	cout << "new S1: ";
	S1.print();
	cout << "new Tree S1: ";
	print(S1.tree);
```

```
        // EXCL
        cout << endl << endl << "excl S2 from S1: " << endl;
        cout << "S1: ";
        S1.print();
        cout << "S2: ";
        S2.print();
        S1.excl(S2);
        cout << "new S1: ";
        S1.print();
        cout << "new Tree S1: ";
        print(S1.tree);

        _getch();
}
```