**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра вычислительной техники**

отчет

**по лабораторной работе №6**

**по дисциплине «Алгоритмы и структуры данных»**

Тема: «Использование стандартной библиотеки шаблонов»

**Вариант №9**

|  |  |  |
| --- | --- | --- |
| Студенты гр. 6307 |  | Лазарев С.О.  Медведев Е.Р. |
| Преподаватель |  | Колинько П.Г. |

Санкт-Петербург

2018

Содержание

[ЦЕЛЬ 3](#_Toc514756635)

[ЗАДАНИЕ 3](#_Toc514756636)

[ФОРМАЛИЗАЦИЯ ЗАДАНИЯ 4](#_Toc514756637)

[НАБОР ПОДХОДЯЩИХ КОНТЕЙНЕРОВ, ФУНКЦИИ STL 4](#_Toc514756638)

[ВРЕМЕННАЯ СЛОЖНОСТЬ 5](#_Toc514756639)

[ВЫВОДЫ 7](#_Toc514756640)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 8](#_Toc514756641)

[ПРИЛОЖЕНИЕ 9](#_Toc514756642)

# 

# ЦЕЛЬ

Получить практические навыки по работе со стандартной библиотекой шаблонов.

# ЗАДАНИЕ

Составить и отладить программу для использования контейнеров из стандартной библиотеки шаблонов.

F = A ⊕ B ∩ C ∪ D ∪ E

# ФОРМАЛИЗАЦИЯ ЗАДАНИЯ

Для хранения множеств выбрать контейнер подходящего типа и доработать его для поддержки операций с последовательностями. Для реализации операций с контейнерами использовать возможности библиотеки алгоритмов. Программа должна реализовывать цепочку операций с последовательностями.

- EXCL – вторая последовательность исключается из первой, если она является ее частью.

- SUBST – вторая последовательность включается в первую с указанной позиции p.

- CONCAT – вторая последовательность подсоединяется к концу первой, образуя ее продолжение.

# НАБОР ПОДХОДЯЩИХ КОНТЕЙНЕРОВ, ФУНКЦИИ STL

Структура данных представлена в форме контейнера стандартной библиотеки шаблонов set. Данная структура данных хранит множество в виде дерева двоичного поиска с автобалансировкой (красно-черное дерево), причем контейнер set хранит множество ключей

Также для реализации функций над последовательностями был использован контейнер вектор указателей на значения элементов множества.

Из библиотеки algorithm были использованы методы set\_difference, set\_symmetri\_difference, set\_intersection для вычитания, пересечения множеств.

**Пример**:

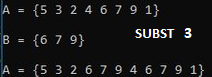


Рис 1. SUBST(3)

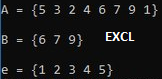


Рис 2. EXCL

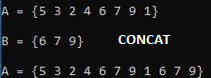


Рис 3. CONCAT

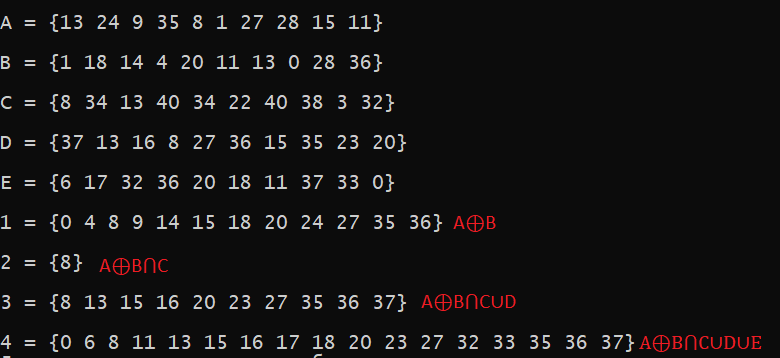


Рис 4. Множества

# ВРЕМЕННАЯ СЛОЖНОСТЬ

Временная сложность алгоритмов STL представлена в таблице 1.

Временная сложность функций программы представлена в таблице 2.

Таблица. 1. Временная сложность алгоритмов STL

|  |  |
| --- | --- |
| Функция | Временная сложность |
| vector::push\_back | O(1) |
| set::insert | O(ln n) |
| set\_intersection | O(n) |
| set\_difference | O(n) |
| set\_symmetric\_difference | O(n) |
| Swap | O(1) |
| Size | O(1) |

Таблица. 2. Временная сложность функций программы

|  |  |  |
| --- | --- | --- |
| Функция | Средняя | Худшая |
| setAnd | O(n) | O(n) |
| setDif | O(n) | O(n) |
| setSymDif | O(n) | O(n) |
| Excl | O(n) | O(n) |
| Subst | O(n) | O(n) |
| Concat | O(n) | O(n) |

# ВЫВОДЫ

В данной работе мы изучали некоторые элементы стандартной библиотеки шаблонов, в частности контейнеры set, vector, а также работу с ними. Мы научились использовать данные контейнеры, стандартные алгоритмы для работы с ними, изучили структуру контейнеров.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Алгоритмы и структуры данных. – Методические указания к лабораторным работам, практическим занятиям и курсовому проектированию, часть 2, глава 1 «Работа с иерархией объектов: наследование и полиморфизм».

# ПРИЛОЖЕНИЕ

***main.cpp***

#include<iostream>

#include<set>

#include<vector>

#include<algorithm>

#include<iterator>

#include<time.h>

using namespace std;

class mySet

{

public:

set<int> values;

vector<set<int>::iterator> pointers;

char name;

mySet(char n) :name(n) {}

void gen(int n, int mod)

{

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

{

pointers.push\_back(values.insert(rand() % (1 + mod)).first);

}

}

void setOut()

{

cout << "\n" << name << " = {";

for (auto it = values.cbegin(); it != values.cend(); ++it)

{

cout << (\*it) << ' ';

}

cout << "\b}" << endl;

}

void seqOut()

{

cout << "\n" << name << " = {";

for (auto it = pointers.cbegin(); it != pointers.cend(); ++it)

{

cout << (\*(\*it)) << ' ';

}

cout << "\b}" << endl;

}

};

string sub(string str, string del)

{

string::size\_type pos = str.find(del);

while (pos != string::npos)

{

str.erase(pos, del.size());

pos = str.find(del, pos + 1);

}

return str;

}

int setAnd(mySet A, mySet B, mySet& C) // &

{

C.values.clear();

C.pointers.clear();

set\_intersection<set<int>::const\_iterator, set<int>::const\_iterator, insert\_iterator<set<int>>>

(A.values.cbegin(), A.values.cend(), B.values.cbegin(), B.values.cend(), inserter(C.values, C.values.begin()));

for (auto it = C.values.cbegin(); it != C.values.cend(); ++it)

{

C.pointers.push\_back(it);

}

return C.pointers.size();

}

int setOr(mySet A, mySet B, mySet& C) // \*

{

C.values.clear();

C.pointers.clear();

set\_union<set<int>::const\_iterator, set<int>::const\_iterator, insert\_iterator<set<int>>>

(A.values.cbegin(), A.values.cend(), B.values.cbegin(), B.values.cend(), inserter(C.values, C.values.begin()));

for (auto it = C.values.cbegin(); it != C.values.cend(); ++it)

{

C.pointers.push\_back(it);

}

return C.pointers.size();

}

int setDif(mySet A, mySet B, mySet& C) //-

{

C.values.clear();

C.pointers.clear();

set\_difference<set<int>::const\_iterator, set<int>::const\_iterator, insert\_iterator<set<int>>>

(A.values.cbegin(), A.values.cend(), B.values.cbegin(), B.values.cend(), inserter(C.values, C.values.begin()));

for (auto it = C.values.cbegin(); it != C.values.cend(); ++it)

{

C.pointers.push\_back(it);

}

return C.pointers.size();

}

int excl(mySet a, mySet b, mySet& C)

{

string s1, s2, s3;

vector<int> temp(a.values.begin(), a.values.end()), temp2(b.values.begin(), b.values.end());

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

{

s1 += temp[i] + '0';

}

for (int i = 0; i < temp2.size(); i++)

{

s2 += temp2[i] + '0';

}

s3 = sub(s1, s2);

for (int i = 0; i < s3.size(); i++)

{

C.pointers.push\_back(C.values.insert((int)s3[i] - '0').first);

}

return C.values.size();

}

int merge(mySet A, mySet B, mySet& C)

{

vector<int> temp(A.values.begin(), A.values.end()), temp2(B.values.begin(), B.values.end());

temp.insert(temp.end(), temp2.begin(), temp2.end());

sort(temp.begin(), temp.end());

string s;

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

{

s += temp[i] + '0';

}

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

{

C.pointers.push\_back(C.values.insert((int)s[i] - '0').first);

}

return C.values.size();

}

int setSymDif(mySet A, mySet B, mySet& C) // xor

{

C.values.clear();

C.pointers.clear();

set\_symmetric\_difference<set<int>::const\_iterator, set<int>::const\_iterator, insert\_iterator<set<int>>>

(A.values.cbegin(), A.values.cend(), B.values.cbegin(), B.values.cend(), inserter(C.values, C.values.begin()));

for (auto it = C.values.cbegin(); it != C.values.cend(); ++it)

{

C.pointers.push\_back(it);

}

return C.pointers.size();

}

int subst(mySet& A, mySet B, int p)

{

mySet T('A');

int i = 0;

auto itA = A.pointers.cbegin();

for (; itA != A.pointers.cend() && i < p; ++itA, ++i)

{

T.pointers.push\_back(T.values.insert((\*(\*itA))).first);

}

for (auto it = B.pointers.cbegin(); it != B.pointers.cend(); ++it)

{

T.pointers.push\_back(T.values.insert((\*(\*it))).first);

}

for (; itA != A.pointers.cend(); ++itA)

{

T.pointers.push\_back(T.values.insert((\*(\*itA))).first);

}

swap(A, T);

return A.pointers.size();

}

int concat(mySet& A, mySet B)

{

for (auto it = B.pointers.cbegin(); it != B.pointers.cend(); ++it)

{

A.pointers.push\_back(A.values.insert((\*(\*it))).first);

}

return A.pointers.size();

}

int main()

{

srand(time(nullptr));

mySet A('A'), B('B'), C('C'), D('D'), E('E'), BC('1'), BCD('2'), BCDE('3'), xo('4'), exc('e');

mySet mer('m');

/\*A.pointers.push\_back(A.values.insert(5).first);

A.pointers.push\_back(A.values.insert(3).first);

A.pointers.push\_back(A.values.insert(2).first);

A.pointers.push\_back(A.values.insert(4).first);

A.pointers.push\_back(A.values.insert(6).first);

A.pointers.push\_back(A.values.insert(7).first);

A.pointers.push\_back(A.values.insert(9).first);

A.pointers.push\_back(A.values.insert(1).first);

B.pointers.push\_back(B.values.insert(1).first);

B.pointers.push\_back(B.values.insert(2).first);

B.pointers.push\_back(B.values.insert(3).first);\*/

A.gen(10, 40);

B.gen(10, 40);

C.gen(10, 40);

D.gen(10, 40);

E.gen(10, 40);

A.seqOut();

B.seqOut();

C.seqOut();

D.seqOut();

E.seqOut();

setSymDif(A, B, BC);

setAnd(BC, C, BCD);

setOr(BCD, D, BCDE);

setOr(BCDE, E, xo);

BC.seqOut();

BCD.seqOut();

BCDE.seqOut();

xo.seqOut();

//excl(A, B,exc);

//exc.seqOut();

//C.seqOut();

//setAnd(A, B, C);

//setDif(A, B, C);

//setSymDif(A, B, C);

//concat(A, B);

//A.seqOut();

//subst(A, B, 3);

//A.seqOut();

//merge(A, B, mer);

//mer.seqOut();

//setOr(A, B, C);

//C.seqOut();

system("pause");

return 0;

}