**Новосибирский государственный технический университет**

Лабораторная работа № 8 по дисциплине программирование

Тема: Библиотека шаблонных классов STL

|  |  |
| --- | --- |
| Группа: АВТ-907  Студент: Смоленский И.В. | Преподаватель: Булатов А. Д. |

Новосибирск, 2020

**Задание**:

Вариант 14.

Цель работы: Ознакомиться с шаблонными классами библиотеки STL. Изучить применение этих классов и их методов на практике.

Задания к лабораторной работе

Для встроенного типа и класса из лабораторной работы №1 провести временной анализ заданных шаблонных классов на основных операциях: добавление, удаление, поиск, сортировка. Использовать итераторы для работы с контейнерами. Для получения времени выполнения операции засекать системное время начала и окончания операции и автоматически генерировать большое количество данных.

**Вариант 14**

* стек;
* словарь с дубликатами.

Краткие теоретические сведения:

Краткие теоретические сведения

Контейнерные классы — это классы, предназначенные для хранения данных, ор­ганизованных определенным образом. Примерами контейнеров могут служить массивы, линейные списки или стеки. Для каждого типа контейнера определены методы для работы с его элементами, не зависящие от конкретного типа данных, которые хранятся в контейнере, поэтому один и тот же вид контейнера можно использовать для хранения данных различных типов. Эта возможность реализо­вана с помощью шаблонов классов, поэтому часть библиотеки C++, в которую входят контейнерные классы, а также алгоритмы и итераторы, о которых будет рассказано в следующих разделах, называют ***стандартной библиотекой шаблонов***(STL — Standard Template Library).

STL содержит контейнеры, реализующие основные структуры данных, исполь­зуемые при написании программ — *векторы, двусторонние очереди, списки и их разновидности, словари и множества.* Контейнеры можно разделить на два типа; ***последовательные*** и ***ассоциативные***.

***Последовательные контейнеры***обеспечивают хранение конечного количества од­нотипных величин в виде непрерывной последовательности. К ним относятся ***векторы (vector), двусторонние очереди (deque) и списки (list)***, а также так на­зываемые адаптеры, то есть варианты, контейнеров — ***стеки (stack), очереди (queue) и очереди с приоритетами (priority\_queue).***

***Ассоциативные контейнеры***обеспечивают быстрый доступ к данным по ключу. Эти контейнеры построены на основе сбалансированных деревьев. Существует пять типов ассоциативных контейнеров; ***словари (тар), словари с дубликатами (multimap), множества (set), множества с дубликатами (multiset) и битовые мно­жества (bitset).***

Контейнерные классы обеспечивают стандартизованный интерфейс при их ис­пользовании. Смысл одноименных операций для различных контейнеров одина­ков, основные операции применимы ко всем типам контейнеров. Стандарт опре­деляет только интерфейс контейнеров, поэтому разные реализации могут сильно отличаться по эффективности.

# При помощи итераторов можно просматривать контейнеры, не заботясь о факти­ческих типах данных, используемых для доступа к элементам. *Итератор* является аналогом указателя на элемент. Он используется для про­смотра контейнера в прямом или обратном направлении. Все, что требуется от итератора — уметь ссылаться на элемент контейнера и реализовывать операцию перехода к его следующему элементу.

В программе осуществлена реализация очереди типа int и типа Fraction. Также были реализованы словари типов <int, int> и <int, Fraction>.

Для каждой очереди и словаря были созданы 10000 объектов, для очередей не были осуществлены функции поиска и сортировки, так как стандартные функции библиотеки STL не подходят для односторонней очереди. Для словарей не были осуществлены сортировки, потому что в словарях происходит автоматическая сортировка при добавлении нового элемента.

Реализация 8 лабораторной работы:

void Lab8\_Object\_Test()

{

int a, b;

char\* SS = new char[20];

const char\* SSS = "Happiness " ;

int start, end;

stack <Bankomat> myStack;

cout << " Start BaNkOmAt stack " << endl;

start = clock();

strcpy(SS, SSS);

for (int g = 0; g < 10000; g++)

{

a = rand() % 10000;

b = rand() % 5000;

Bankomat \*Afs = new Bankomat(SS, a, b);

myStack.push(\*Afs);

}

end = clock() - start;

cout << " Time for creating : " << end << endl;

start = clock();

for (int i = 1; i < 3; i++)

myStack.pop();

end = clock() - start;

cout << " Time for delete : " << end << endl;

// Map

char\* SS = new char[20];

const char\* SSS = "Happiness ";

int start, end;

multimap<int,Bankomat> myMultiMap;

cout << " Start bankomat multimap " << endl;

start = clock();

strcpy(SS, SSS);

for (int i = 0; i < 10000; i++)

{

a = rand() % 10000;

b = rand() % 5000;

Bankomat\* Afs = new Bankomat(SS, a, b);

myMultiMap.insert(make\_pair(i, Bankomat::add(i,Afs)));

}

end = clock() - start;

cout << " Time for creating : " << end << endl;

map<int, Bankomat> ::iterator it2;

start = clock();

it2 = myMultiMap.find(10001);

end = clock() - start;

cout << "Searching time: " << end << endl;

start = clock();

for (int i = 1; i < 10000; i++) myMultiMap.erase(i);

end = clock() - start;

cout << "Deleting time: " << end << endl;

}

void Lab\_8()

{

stack <int> myIntStack;

int start, end;

start = clock();

cout << " Start int stack " << endl;

for (int i = 0; i < 10000; i++)

myIntStack.push(i);

end = clock() - start;

cout << " Time for creating : " << end << endl;

start = clock();

for (int i = 1; i < 10000; i++)

myIntStack.pop();

end = clock() - start;

cout << " Time for delete : " << end << endl;

multimap<int, int> myIntMap;

start = clock();

for (int i = 0; i < 10000; i++) myIntMap.insert(make\_pair(i, 10000 - i)); //добавление 10000 элементов

end = clock() - start;

std::cout << "Creating time: " << end << endl;

map<int, int> ::iterator it;

start = clock();

it = myIntMap.find(10001);

end = clock() - start;

std::cout << "Searching time: " << end << endl;

start = clock();

for (int i = 1; i < 10000; i++) myIntMap.erase(i);

end = clock() - start;

std::cout << "Deleting time: " << end << endl;

}

Вывод:

Стандартные классы библиотеки STL упрощают работу с различными структурами. Они имеют встроенные стандартные функции, что упрощает реализацию решения поставленной задачи. Также из рассчитанного количества тактов, полученного функцией clock(), можно сделать вывод что все стандартные функции работают чрезвычайно быстро и эффективно.