

Лабораторная работа №13

Стандартные обобщенные алгоритмы библиотеки STL.

1. Цель задания:

- 1) Создание консольного приложения, состоящего из нескольких файлов в системе программирования Visual Studio.
- 2) Использование стандартных обобщенных алгоритмов из библиотеки STL в ОО программе.

2. Теоретические сведения

Каждый алгоритм выражается шаблоном функции или набором шаблонов функций. Таким образом, алгоритм может работать с очень разными контейнерами, содержащими значения разнообразных типов. Алгоритмы, которые возвращают итератор, как правило, для сообщения о неудаче используют конец входной последовательности. Алгоритмы не выполняют проверки диапазона на их входе и выходе. Когда алгоритм возвращает итератор, это будет итератор того же типа, что и был на входе. Алгоритмы в STL реализуют большинство распространенных универсальных операций с контейнерами, такие как просмотр, сортировка, поиск, вставка и удаление элементов.

Алгоритмы определены в заголовочном файле `<algorithm.h>`.

2.1. Функциональные объекты и предикаты

Функциональным объектом называется класс, в котором определена операция вызова функции. Чаще всего эти объекты используются в качестве параметров стандартных алгоритмов для задания пользовательских критериев сравнения объектов или способов их обработки.

В тех алгоритмах, где в качестве параметра можно использовать функциональный объект, можно использовать и указатель на функцию. При этом применение функционального объекта может оказаться более эффективным, поскольку операцию () можно определить как встроенную.

Стандартная библиотека предоставляет множество функциональных объектов, необходимых для ее эффективного использования и расширения. Они описаны в заголовочном файле `<functional>`.

Среди этих объектов можно выделить объекты, возвращающие значения типа `bool`. Такие объекты называются предикатами. Предикатом называется также и обычная функция, возвращающая `bool`.

В стандартной библиотеке определены шаблоны функциональных объектов для всех арифметических операций, определенных в языке C++.

Имя	Результат
<code>plus</code>	$x + y$
<code>minus</code>	$x - y$
<code>multiply</code>	$x * y$
<code>divide</code>	x / y
<code>modulus</code>	$x \% y$
<code>negate</code>	$-x$

Рассмотрим шаблон объекта `plus` (остальные объекты описаны аналогичным образом):

```
//базовый класс, который вводит имена для типов аргументов
//шаблон бинарной функции
```

```

template <class Arg1, class Arg2, class Result>
struct binary__function
{
    typedef Arg1 first_argument_type;
    typedef Arg2 second_argument_type;
    typedef Result result_type;
};

//параметризированный класс - наследник
template <class T>
struct plus : binary_function <T, T, T>
{
    T operator()(const T& x, const T& y) const
    {
        return x + y;
    }
};

```

В стандартной библиотеке определены шаблоны функциональных объектов для операций сравнения и логических операций, определенных в языке C++. Они возвращают значение типа `bool`, то есть являются предикатами.

Имя	Результат
<code>equal_to</code>	<code>x==y</code>
<code>not_equal_to</code>	<code>x!= y</code>
<code>greater</code>	<code>x > y</code>
<code>less</code>	<code>x < y</code>
<code>greater_equal</code>	<code>x >= y</code>
<code>less_equal</code>	<code>x <= y</code>
<code>logical_and</code>	<code>x && y</code>
<code>logical_or</code>	<code>x y</code>
<code>logical_not</code>	<code>! x</code>

Рассмотрим шаблон объекта `equal_to` (остальные объекты описаны аналогичным образом):

```

template <class T>
struct equal_to : binary_function <T, T, bool>
{
    bool operator ()(const T& x, const T& y) const
    {
        return x == y;
    }
};

```

Программист может описать собственные предикаты для определения критериев сравнения объектов. Это необходимо, когда контейнер состоит из элементов пользовательского типа.

```

class Person
{
    string name;
    int age;
    . . . .
};

//сравнение объектов типа Person по полю age
struct Person_less_age:
public binary_function<Person, Person, bool>

```

```

{
    bool operator() (Person p1, Person p2)
    {
        return p1.get_age() < p2.get_age();
    }
};

```

2.2. Адаптеры функций

Адаптером функции называют функцию, которая получает в качестве аргумента функцию и конструирует из нее другую функцию. На месте функции может быть также функциональный объект.

Стандартная библиотека содержит описание нескольких типов адаптеров:

- связыватели для использования функционального объекта с двумя аргументами как объекта с одним аргументом;
- отрицатели для инверсии значения предиката;
- адаптеры указателей на функцию;
- адаптеры методов для использования методов в алгоритмах.

Отрицатели `not1` и `not2` применяются для получения противоположного унарного и бинарного предиката соответственно. Для того чтобы получить инверсию предиката `less<int>()`, нужно записать выражение `not2(less<int>())`. Оно эквивалентно `greater_equal<int>`.

Отрицатели применяются для инвертирования предикатов, заданных пользователем, т.к. для стандартных предикатов библиотека содержит соответствующие им противоположные объекты.

С помощью бинарных предикатов можно сравнивать два различных объекта. Часто требуется сравнить объект не с другим объектом, а с константой. Чтобы использовать для этого тот же самый предикат, требуется связать один из двух его аргументов с константой. Для этого используются связыватели `bind2nd` и `bind1st`, позволяющие связать с конкретным значением соответственно второй и первый аргумент бинарной функции.

Связыватели реализованы в стандартной библиотеке как шаблоны функций, принимающих первым параметром функциональный объект `f` с двумя аргументами, а вторым — привязываемое значение `value`. Результатом вызова функции является функциональный объект, созданный из входного объекта `f` путем «подстановки» `value` в его первый или второй аргумент.

Для описания типа возвращаемого функционального объекта в библиотеке описаны шаблоны классов `binder2nd` и `binder1st`:

```

template <class Op, class T>
binder2nd<Op> bind2nd(const Op& op, const T& x);

```

```

template <class Op, class T>
binder1st<Op> bind1st(const Op& op, const T& x);

```

где `Op` — тип функционального объекта, `T` — тип привязываемого значения.

Пример использования связывателя в стандартном алгоритме `count_if()`:

```

int k=count_if(m, m + 8, bind2nd(less<int>(), 40));

```

Здесь вычисляется количество элементов в одномерном массиве `m` из 8 элементов меньших 40.

Для того чтобы применять связыватели к обычным указателям на функции, требуются специальные преобразователи, или адаптеры. Стандартная библиотека определяет два функциональных объекта — указатель на унарную функцию

pointer_to_unary_function и указатель на бинарную функцию pointer_to_binary_function, а также две функции-адаптера ptrfun с одним и двумя аргументами, которые преобразуют переданный им в качестве параметра указатель на функцию в функциональный объект.

Пример применения адаптера функции:

```
#include <iostream>
#include <functional>
#include <algorithm>
using namespace std;
//структура с двумя полями
struct A
{
    int x, y;
};
// логическая функция для сравнения объектов типа A по полю x
bool lss(A a1, A a2){return a1.x < a2.x;}
void main()
{
    A ma[5] = {{2, 4}, {3, 1}, {2, 2}, {1, 2}, {1, 2}};
    A elem = {3, 0};
    cout <<count_if(ma, ma + 5. bind2nd(ptr_fun(lss), elem));
}
```

Данный фрагмент программы вычисляет количество элементов структуры ma, удовлетворяющих условию, заданному третьим параметром. Этим параметром является функциональный объект, созданный связывателем bind2nd из функционального объекта, полученного из функции lss с помощью адаптера ptr_fun, и переменной, подставляемой на место второго параметра функции. В результате будет вычислено количество элементов структуры A, поле x которых меньше 3.

При хранении в контейнерах объектов пользовательских классов данных часто возникает задача применить ко всем элементам контейнера один и тот же метод класса. Для просмотра элементов контейнера в библиотеке есть алгоритм for_each. В него можно передать указатель на функцию, которую требуется вызвать для каждого просматриваемого элемента контейнера, например:

```
#include <iostream>
#include <algorithm>
using namespace std;
void show(int a){ cout << a<<endl;}
void main()
{
    int m[4] = {3, 5, 9, 6}:
    for_each(m, m + 4, show);
}
```

Если вместо обычного массива использовать контейнер, содержащий объекты некоторого класса, то записать в аналогичной программе на месте функции show() вызов метода класса не удастся, поскольку метод требуется вызывать с указанием конкретного объекта класса.

Адаптеры методов позволяют использовать методы классов в качестве аргументов стандартных алгоритмов.

Имя	Тип объекта	Действие
mem_fun	mem_fun_t	Вызывает безаргументный метод через указатель
mem_fun	const_mem_fun_t	Вызывает безаргументный константный метод через

		указатель
mem_fun	mem_fun1_t	Вызывает унарный метод через указатель
mem_fun_ref	mem_fun_ref_t	Вызывает безаргументный метод через ссылку
mem_fun_ref	const_mem_fun_ref_t	Вызывает безаргументный константный метод через ссылку
mem_fun_ref	mem_fun1_ref_t	Вызывает унарный метод через ссылку
mem_fun_ref	const_mem_fun1_ref_t	Вызывает унарный константный метод через ссылку
mem_fun_ref	const_mem_fun1_t	Вызывает унарный константный метод через указатель

```

class Person()
{
    string name;
    int age;
    public()
        . . . .
    bool f(){ . . . . } //свойство объекта
        . . . .
};

//вектор из объектов типа Person
vector <Person> v(10);
. . . .
//находит количество элементов вектора, обладающих свойством f
int k=count_if(v.begin(), v.end(), mem_fun_ref(&person::f));

```

2.3. Алгоритмы

Алгоритмы STL предназначены для работы с контейнерами и другими последовательностями. Каждый алгоритм реализован в виде шаблона или набора шаблонов функции, поэтому может работать с различными видами последовательностей и данными разнообразных типов. Для настройки алгоритма на конкретные требования пользователя применяются функциональные объекты.

Объявления стандартных алгоритмов находятся в заголовочном файле <algorithm>, стандартных функциональных объектов — в файле <functional>.

Все алгоритмы STL можно разделить на четыре категории:

- немодифицирующие операции с последовательностями;
- модифицирующие операции с последовательностями;
- алгоритмы, связанные с сортировкой;
- алгоритмы работы с множествами и пирамидами;

В качестве параметров алгоритму передаются итераторы, определяющие начало и конец обрабатываемой последовательности. Вид итераторов определяет типы контейнеров, для которых может использоваться данный алгоритм.

2.3.1. Немодифицирующие операции с последовательностями

Алгоритм	Выполняемая функция
<code>for_each()</code>	выполняет операции для каждого элемента последовательности
<code>find()</code>	находит первое вхождение значения в последовательность
<code>find_if()</code>	находит первое соответствие предикату в последовательности
<code>count()</code>	подсчитывает количество вхождений значения в последовательность
<code>count_if()</code>	подсчитывает количество выполнений предиката в последовательности
<code>search()</code>	находит первое вхождение последовательности как подпоследовательности
<code>search_n()</code>	находит n-е вхождение значения в последовательность

2.3.2. Модифицирующие операции с последовательностями

Алгоритм	Выполняемая функция
<code>copy()</code>	копирует последовательность, начиная с первого элемента
<code>swap()</code>	меняет местами два элемента
<code>replace()</code>	меняет местами два элемента заменяет элементы с указанным значением
<code>replace_if()</code>	заменяет элементы при выполнении предиката
<code>replace_copy()</code>	копирует последовательность, заменяя элементы с указанным значением
<code>replace_copy_if()</code>	копирует последовательность, заменяя элементы при выполнении предиката
<code>fill()</code>	заменяет все элементы данным значением
<code>remove()</code>	удаляет элементы с данным значением
<code>remove_if()</code>	удаляет элементы при выполнении предиката
<code>remove_copy()</code>	копирует последовательность, удаляя элементы с указанным значением
<code>remove_copy_if()</code>	копирует последовательность, удаляя элементы при выполнении предиката
<code>reverse()</code>	меняет порядок следования элементов на обратный
<code>transform()</code>	выполняет заданную операцию над каждым элементом последовательности
<code>unique()</code>	удаляет равные соседние элементы

<code>unique_copy()</code>	копирует последовательность, удаляя равные соседние элементы
----------------------------	--

2.3.3. Алгоритмы, связанные с сортировкой

Алгоритм	Выполняемая функция
<code>sort()</code>	сортирует последовательность с хорошей средней эффективностью
<code>partial_sort()</code>	сортирует часть последовательности
<code>stable_sort()</code>	сортирует последовательность, сохраняя порядок следования равных элементов
<code>lower_bound()</code>	находит первое вхождение значения в отсортированной последовательности
<code>upper_bound()</code>	находит первый элемент, больший чем заданное значение
<code>binary_search()</code>	определяет, есть ли данный элемент в отсортированной последовательности
<code>merge()</code>	сливает две отсортированные последовательности
<code>min()</code>	меньшее из двух
<code>max()</code>	большее из двух
<code>min_element()</code>	наименьшее значение в последовательности
<code>max_element()</code>	наибольшее значение в последовательности

2.3.4. Алгоритмы работы с множествами и пирамидами

Алгоритм	Выполняемая функция
<code>includes()</code>	проверка на вхождение одного множества в другое
<code>set_union()</code>	объединение множеств
<code>set_intersection()</code>	пересечение множеств
<code>set_difference()</code>	разность множеств
<code>make_heap()</code>	преобразование последовательности с произвольным доступом в пирамиду
<code>pop_heap()</code>	извлечение элемента из пирамиды
<code>push_heap()</code>	добавление элемента в пирамиду
<code>sort_heap()</code>	сортировка пирамиды

3. Постановка задачи

Задача 1.

1. Создать последовательный контейнер.

2. Заполнить его элементами пользовательского типа (тип указан в варианте). Для пользовательского типа перегрузить необходимые операции.
3. Заменить элементы в соответствии с заданием (использовать алгоритмы `replace_if()`, `replace_copy()`, `replace_copy_if()`, `fill()`).
4. Удалить элементы в соответствии с заданием (использовать алгоритмы `remove()`, `remove_if()`, `remove_copy_if()`, `remove_copy()`).
5. Отсортировать контейнер по убыванию и по возрастанию ключевого поля (использовать алгоритм `sort()`).
6. Найти в контейнере заданный элемент (использовать алгоритмы `find()`, `find_if()`, `count()`, `count_if()`).
7. Выполнить задание варианта для полученного контейнера (использовать алгоритм `for_each()`).
8. Для выполнения всех заданий использовать стандартные алгоритмы библиотеки STL.

Задача 2.

1. Создать адаптер контейнера.
2. Заполнить его элементами пользовательского типа (тип указан в варианте). Для пользовательского типа перегрузить необходимые операции.
3. Заменить элементы в соответствии с заданием (использовать алгоритмы `replace_if()`, `replace_copy()`, `replace_copy_if()`, `fill()`).
4. Удалить элементы в соответствии с заданием (использовать алгоритмы `remove()`, `remove_if()`, `remove_copy_if()`, `remove_copy()`).
5. Отсортировать контейнер по убыванию и по возрастанию ключевого поля (использовать алгоритм `sort()`).
6. Найти в контейнере элемент с заданным ключевым полем (использовать алгоритмы `find()`, `find_if()`, `count()`, `count_if()`).
7. Выполнить задание варианта для полученного контейнера (использовать алгоритм `for_each()`).
8. Для выполнения всех заданий использовать стандартные алгоритмы библиотеки STL.

Задача 3

1. Создать ассоциативный контейнер.
2. Заполнить его элементами пользовательского типа (тип указан в варианте). Для пользовательского типа перегрузить необходимые операции.
3. Заменить элементы в соответствии с заданием (использовать алгоритмы `replace_if()`, `replace_copy()`, `replace_copy_if()`, `fill()`).
4. Удалить элементы в соответствии с заданием (использовать алгоритмы `remove()`, `remove_if()`, `remove_copy_if()`, `remove_copy()`).
5. Отсортировать контейнер по убыванию и по возрастанию ключевого поля (использовать алгоритм `sort()`).
6. Найти в контейнере элемент с заданным ключевым полем (использовать алгоритмы `find()`, `find_if()`, `count()`, `count_if()`).
7. Выполнить задание варианта для полученного контейнера (использовать алгоритм `for_each()`).
8. Для выполнения всех заданий использовать стандартные алгоритмы библиотеки STL.

4. Ход работы

Задача 1

1. Контейнер – вектор (vector);
 2. тип элементов - Time ;
 3. Заменить элементы большие среднего арифметического на минимальное значение контейнера.
 4. Отсортировать контейнер по убыванию и по возрастанию ключевого поля.
 5. Найти в контейнере элемент с заданным ключевым полем.
 6. Удалить минимальный элемент из контейнера.
 7. Каждый элемент разделить на максимальное значение контейнера.
1. Создать пустой проект. Для этого требуется
 - 1.1. Запустить MS Visual Studio:
 - 1.2. Выбрать команду File/New/Project
 - 1.3. В окне New Project выбрать Win Console 32 Application, в поле Name указать имя проекта (Lab13), в поле Location указать место положения проекта (личную папку), нажать кнопку Ok.
 - 1.4. В следующем окне выбрать кнопку Next.
 - 1.5. В следующем окне выбрать кнопку Next.
 - 1.6. В диалоговом окне Additional Settings установить флажок Empty (Пустой проект) и нажать кнопку Finish.
 - 1.7. В результате выполненных действий получим пустой проект.
 2. Добавляем в него класс Time из лабораторной работы 11. Для этого нужно вызвать контекстное меню проекта и выбрать пункт Добавить/Существующий элемент (Add/Existing Item). В диалоговом окне выбрать нужные файлы. При использовании файла Time.h в директиве #include нужно будет указывать полный путь к этому файлу, т. к. он будет расположен в папке отличной от папки текущего проекта.
 3. Добавим в проект файл Lab13_main.cpp, содержащий основную программу. Для этого нужно:
 - 3.1. Вызвать контекстное меню проекта в панели Обозреватель решений (Solution Explorer), выбрать в нем пункт меню Add/ New Item.
 - 3.2. В диалоговом окне Add New Item – Lab13 выбрать Категорию Code, шаблон – C++File (.cpp), задать имя файла Lab13_main. В результате выполненных действий получим пустой файл Lab13_main.cpp, в котором будет редактироваться текст программы.
 - 3.3. Ввести следующий текст программы:

```
#include "D:\РАБОТА\ЛабыС++\ПВИ МВД\лабы 2009_2010\lab11\time.h"
#include "vector"
#include <iostream>
using namespace std;
typedef vector<Time> TVector;
//формирование вектора
TVector make_vector(int n)
{
    Time a;
    TVector v;
    for(int i=0;i<n;i++)
    {
        cin>>a;
        v.push_back(a);
    }
    return v;
}
//печать вектора
void print_vector(TVector v)
{
    for(int i=0;i<v.size();i++)
```

```

        cout<<v[i]<<endl;
    cout<<endl;
}
void main()
{
    int n;
    cout<<"N?";
    cin>>n;
    TVector v;
    v=make_vector(n);
    print_vector(v);
}

```

4. Запустить программу на выполнение и протестировать ее работу.

5. Подключить библиотеку для работы с алгоритмами STL

```
#include <algorithm>
```

6. Добавим глобальную переменную Time s для сравнения с заданным значением перед функцией main():

```

.....
typedef vector<Time> TVector;
Time s;

```

7. Добавим предикат для сравнения заданного значения с объектом типа Time перед функцией main()

```

struct Greater_s //больше, чем s
{
    bool operator() (Time t)
    {
        if (t>s) return true; else return false;
    }
};

```

8. Добавим функцию для вычисления среднего арифметического вектора (см. лабораторную работу №11):

```

Time srednee(TVector v)
{
    Time s=v[0];
    //перебор вектора
    for(int i=1;i<v.size();i++)
        s=s+v[i];
    int n=v.size();//количество элементов в векторе
    return s/n;
}

```

8. Добавим в функцию main() операторы для выполнения задания 3:

```

void main()
{
    . . . . .
    TVector::iterator i;
    //поставили итератор i на максимальный элемент
    i=max_element(v.begin(),v.end());
    cout<<"max="<<*i<<endl;
    Time m=*i;
    s=srednee(v);//нашли среднее арифметическое вектора
    cout<<"sred="<<s<<endl;
    //замена с использованием предиката
    replace_if(v.begin(),v.end(), Greater_s(),m);
    cout<<"ЗАМЕНА"<<endl;
    print_vector(v);
}

```

9. Запустить программу на выполнение и протестировать ее работу.

10. Добавим перед функцией main() предикат для изменения порядка сортировки:

```
struct Comp_less // для сортировки по убыванию
```

```

{
    public:
    bool operator() (Time t1, Time t2)
    {

        if(t1>t2) return true;
        else return false;

    }
};

```

11. Добавим в функцию main() операторы для выполнения задания 4:

```

void main()
{
    . . . . .
    //по убыванию
    cout<<"Sortirovka po ubivaniu:"<<endl;
    sort(v.begin(), v.end(), Comp_less());
    print_vector(v);
    //по возрастанию
    cout<<"Sortirovka po vozrasaniu:"<<endl;
    sort(v.begin(), v.end());
    print_vector(v);
}

```

12. Запустить программу на выполнение и протестировать ее работу.

13. Добавим перед функцией main() предикат для поиска заданного значения:

```

struct Equal_s
{
    bool operator() (Time t)
    {
        return t==s;
    }
};

```

14. Добавим в функцию main() операторы для выполнения задания 5:

```

void main()
{
    . . . . .
    cout<<"POISK"<<endl;
    cin>>s;
    //поиск элементов, удовлетворяющих условию предиката
    i=find_if(v.begin(), v.end(), Equal_s());
    if(i!=v.end())//если нет конца вектора
        cout<<*(i)<<endl;
    else
        cout<<"Not such element!"<<endl;
}

```

15. Запустить программу на выполнение и протестировать ее работу.

16. Добавим в функцию main() операторы для выполнения задания 6:

```

void main()
{
    . . . . .
    cout<<"UDALENIE"<<endl;
    i=min_element(v.begin(), v.end());
    s=*i;
    //переместили элементы совпадающие с min в конец вектора
    i=remove_if(v.begin(), v.end(), Equal_s());
    //удалили элементы, начиная с i и до конца вектора
    v.erase(i, v.end());
    print_vector(v);
}

```

17. Запустить программу на выполнение и протестировать ее работу.

18. Добавим функцию для выполнения задания 7:

```
void del (Time& t)
{
    t=t/s;
}
```

19. Добавим в функцию main() операторы для выполнения задания 7:

```
void main()
{
    . . . . .
    cout<<"DELENIIE"<<endl;
    i=max_element(v.begin(),v.end());
    s=*i;
    //для каждого элемента вектора вызывается функция del
    for_each(v.begin(),v.end(),del);
    print_vector(v);
}
```

20. Запустить программу на выполнение и протестировать ее работу.

5. Варианты

№	Задание		
1	Задача 1 1. Контейнер - вектор 2. Тип элементов - Time(см. лабораторную работу №3). Задача 2 Адаптер контейнера - стек. Задача 3 Ассоциативный контейнер - множество		
	Задание 3	Задание 4	Задание 5
	Заменить максимальный элемент на заданное значение	Найти минимальный элемент и удалить его из контейнера	К каждому элементу добавить среднее арифметическое контейнера
2	Задача 1 1. Контейнер - список 2. Тип элементов Time (см. лабораторную работу №3). Задача 2 Адаптер контейнера - очередь. Задача 3 Ассоциативный контейнер – множество с дубликатами		
	Задание 3	Задание 4	Задание 5
	Найти минимальный элемент и добавить его в конец контейнера	Найти элемент с заданным ключом и удалить его из контейнера	К каждому элементу добавить сумму минимального и максимального элементов контейнера
3	Задача 1 1. Контейнер - двунаправленная очередь 2. Тип элементов Time (см. лабораторную работу №3). Задача 2 Адаптер контейнера – очередь с приоритетами. Задача 3 Ассоциативный контейнер - словарь		
	Задание 3	Задание 4	Задание 5
	Найти элемент с заданным ключом и добавить его на заданную позицию	Найти элемент с заданным ключом и удалить его из	Найти разницу между максимальным и минимальным элементами контейнера и

	контейнера	контейнера	вычесть ее из каждого элемента контейнера
4	Задача 1 1. Контейнер - двунаправленная очередь 2. Тип элементов Time (см. лабораторную работу №3). Задача 2 Адаптер контейнера - очередь. Задача 3 Ассоциативный контейнер – словарь с дубликатами		
	Задание 3	Задание 4	Задание 5
	Найти максимальный элемент и добавить его в конец контейнера	Найти элемент с заданным ключом и удалить его из контейнера	К каждому элементу добавить среднее арифметическое элементов контейнера
5	Задача 1 1. Контейнер - список 2. Тип элементов Time (см. лабораторную работу №3). Задача 2 Адаптер контейнера - вектор. Задача 3 Ассоциативный контейнер - множество		
	Задание 3	Задание 4	Задание 5
	Найти минимальный элемент и добавить его на заданную позицию контейнера	Найти элементы большие среднего арифметического и удалить их из контейнера	Каждый элемент домножить на максимальный элемент контейнера
6	Задача 1 1. Контейнер - вектор 2. Тип элементов Money (см. лабораторную работу №3). Задача 2 Адаптер контейнера - стек. Задача 3 Ассоциативный контейнер – множество с дубликатами.		
	Задание 3	Задание 4	Задание 5
	Найти максимальный элемент и добавить его в начало контейнера	Найти минимальный элемент и удалить его из контейнера	К каждому элементу добавить среднее арифметическое контейнера
7	Задача 1 1. Контейнер - вектор 2. Тип элементов Money (см. лабораторную работу №3). Задача 2 Адаптер контейнера - очередь. Задача 3 Ассоциативный контейнер - словарь		
	Задание 3	Задание 4	Задание 5
	Найти минимальный элемент и добавить его в конец контейнера	Найти элемент с заданным ключом и удалить его из контейнера	К каждому элементу добавить сумму минимального и максимального элементов контейнера

8	Задача 1 1. Контейнер - список 2. Тип элементов Money (см. лабораторную работу №3). Задача 2 Адаптер контейнера – очередь с приоритетами. Задача 3 Ассоциативный контейнер – словарь с дубликатами		
	Задание 3	Задание 4	Задание 5
	Найти элемент с заданным ключом и добавить его на заданную позицию контейнера	Найти элемент с заданным ключом и удалить его из контейнера	Найти разницу между максимальным и минимальным элементами контейнера и вычесть ее из каждого элемента контейнера
9	Задача 1 1. Контейнер - двунаправленная очередь 2. Тип элементов Money (см. лабораторную работу №3). Задача 2 Параметризованный класс – Вектор (см. лабораторную работу №7) Задача 3 Ассоциативный контейнер - множество		
	Задание 3	Задание 4	Задание 5
	Найти максимальный элемент и добавить его в конец контейнера	Найти элемент с заданным ключом и удалить его из контейнера	К каждому элементу добавить среднее арифметическое элементов контейнера
10	Задача 1 1. Контейнер - вектор 2. Тип элементов Money (см. лабораторную работу №3). Задача 2 Адаптер контейнера – очередь с приоритетами. Задача 3 Ассоциативный контейнер – множество с дубликатами		
	Задание 3	Задание 4	Задание 5
	Найти минимальный элемент и добавить его на заданную позицию контейнера	Найти элементы большие среднего арифметического и удалить их из контейнера	Каждый элемент домножить на максимальный элемент контейнера
11	Задача 1 1. Контейнер - вектор 2. Тип элементов Money (см. лабораторную работу №3). Задача 2 Адаптер контейнера - очередь. Задача 3 Ассоциативный контейнер - словарь		
	Задание 3	Задание 4	Задание 5
	Найти среднее арифметическое и добавить его в начало контейнера	Найти элемент с заданным ключом и удалить их из контейнера	Из каждого элемента вычесть минимальный элемент контейнера
12	Задача 1 1. Контейнер - список		

	Тип элементов Pair (см. лабораторную работу №3). Задача 2 Адаптер контейнера – очередь с приоритетами. Задача 3 Ассоциативный контейнер словарь с дубликатами		
	Задание 3	Задание 4	Задание 5
	Найти среднее арифметическое и добавить его на заданную позицию контейнера	Найти элементы ключами из заданного диапазона и удалить их из контейнера	Из каждого элемента вычесть среднее арифметическое контейнера.
13	Задача 1 1. Контейнер - двунаправленная очередь 2. Тип элементов Pair (см. лабораторную работу №3). Задача 2 Адаптер контейнера – стек. Задача 3 Ассоциативный контейнер - множество		
	Задание 3	Задание 4	Задание 5
	Найти максимальный элемент и добавить его в конец контейнера	Найти элементы ключами из заданного диапазона и удалить их из контейнера	К каждому элементу добавить среднее арифметическое контейнера.
14	Задача 1 1. Контейнер - вектор 2. Тип элементов Pair (см. лабораторную работу №3). Задача 2 Адаптер контейнера – очередь. Задача 3 Ассоциативный контейнер – множество с дубликатами.		
	Задание 3	Задание 4	Задание 5
	Найти минимальный элемент и добавить его на заданную позицию контейнера	Найти меньше среднего арифметического и удалить их из контейнера	Каждый элемент разделить на максимальный элемент контейнера.
15	Задача 1 1. Контейнер - список 2. Тип элементов Pair (см. лабораторную работу №3). Задача 2 Адаптер контейнера – очередь с приоритетами. Задача 3 Ассоциативный контейнер - словарь		
	Задание 3	Задание 4	Задание 5
	Найти среднее арифметическое и добавить его в конец контейнера	Найти элементы ключами из заданного диапазона и удалить их из контейнера	К каждому элементу добавить сумму минимального и максимального элементов контейнера.