ЛАБОРАТОРНАЯ РАБОТА №1
STL КОНТЕЙНЕРЫ И ИТЕРАТОРЫ

1 Цель работы

- Использование последовательных контейнеров библиотеки STL.
- Использовать для работы с типом пользователя глобальные функции и методы класса.
- Повторить применение шаблонов классов.

2 Теоретические сведения и ссылки

2.1. Основные концепции STL

http://www.cplusplus.com/reference/stl/

STL – Standard Template Library, стандартная библиотека шаблонов состоит из двух основных частей: набора **контейнерных классов** и набора **обобщенных алгоритмов**.

Контейнеры — это объекты, которые содержат коллекцию других однотипных об6ъектов. Реализованы в виде шаблонов, что дает гибкость в их применении как встроенных, так и пользовательских типов. Контейнер управляет памятью для хранения коллекции и предоставляет методы для доступа к элементам, как напрямую, так и через итераторы.

Контейнеры повторяют наиболее часто используемые программные структуры: динамический массив (vector), очередь (queue), стек (stack), очередь с приоритетами или куча (priority queue), список (list),, деревья или множество (set), ассоциативный массив (map)…
Стек, очередь и очередь с приоритетами реализованы как контейнер адаптер, или как класс который обеспечивает интерфейс для решения определенной задачи. Реализованы на последовательных контейнерных классах таких как список или дек (double ended queue). Доступ к внутреннему контейнеру возможен только через методы класса адаптера, независимо от используемого адаптером контейнера.

http://www.cplusplus.com/reference/algorithm/

Алгоритмы — набор специально разработанных функций для применения к диапазону коллекции элементов.
Часть или вся последовательность элементов может быть доступна через итераторы или указатели. Например массив или контейнерный класс (STL containers) с заданным типом. Можно отметить, что алгоритмы работают через итераторы напрямую с объектом контейнера не влияя на сам контейнер.

2.2. Контейнеры

Контейнеры STL можно разделить на два типа: последовательные и ассоциативные (рис. 1).

Рис. 1. Контейнерные классы

Последовательные контейнеры обеспечивают хранение конечного количества однотипных объектов в виде непрерывной последовательности. К базовым последовательным контейнерам относятся:

- векторы (**vector**),
- списки (**list**)
- и двусторонние очереди (**deque**).

Есть еще специализированные контейнеры (или адаптеры контейнеров), реализованные на основе базовых:

- стеки (**stack**),
- очереди (**queue**)
- и очереди с приоритетами (**priority_queue**).

Для использования контейнера в программе необходимо включить в нее соответствующий заголовочный файл. Тип объектов, сохраняемых в контейнере, задается с помощью аргумента шаблона, например:

```
#include <vector>
#include <list>
#include “person.h”
.....
vector<int> v;
list<person> l;
```

Ассоциативные контейнеры обеспечивают быстрый доступ к данным по ключу. Эти контейнеры построены на основе сбалансированных деревьев. Существует пять типов ассоциативных контейнеров:

- словари (**map**),
- словари с дубликатами (**multimap**),
- множества (**set**),
- множества с дубликатами (**multiset**)
- битовые множества (**bitset**).

2.3. Итераторы

Рассмотрим, как можно реализовать шаблон функции для поиска элементов в массиве, который хранит объекты типа Data:

```
template <class Data>
Data* Find(Data*mas, int n, const Data& key)
{
    for(int i=0;i<n;i++)
       if (*(mas + i) == key)
          return mas + i;
    return 0;
}
```

Функция возвращает адрес найденного элемента или 0, если элемент с заданным значением не найден.

Эту функцию можно использовать для поиска элементов в массиве любого типа, но использовать ее для списка нельзя, поэтому авторы STL ввели понятие итератора. Итератор более общее понятие, чем указатель. Тип `iterator` определен для всех контейнерных классов STL, однако, реализация его в разных классах разная.

К основным операциям, выполняемым с любыми итераторами, относятся:

- Разыменование итератора: если p — итератор, то *p — значение объекта, на который он ссылается.
- Присваивание одного итератора другому.
- Сравнение итераторов на равенство и неравенство (== и !=).
- Перемещение его по всем элементам контейнера с помощью префиксного (++p) или постфиксного (p++) инкремента.

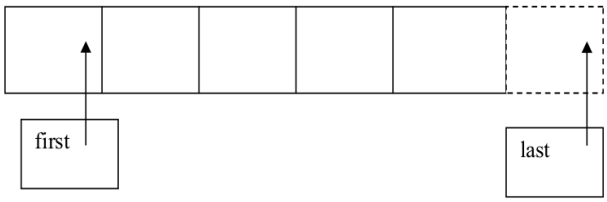
Так как реализация итератора специфична для каждого класса, то при объявлении объектов типа итератор всегда указывается область видимости в форме `имя_шаблона ::` , например:

```
vector<int>::iterator iterl;
List<person>::iterator iter2;
```

Организация циклов просмотра элементов контейнеров тоже имеет некоторую специфику. Так, если i — некоторый итератор, то вместо привычной формы

`for (i =0; i < n; ++i)` используется следующая: **`for (i = first; i != last; ++i)`**,

где **`first`** - значение итератора, указывающее на первый элемент в контейнере, а **`last`** — значение итератора, указывающее на воображаемый элемент, который следует за последним элементом контейнера.



Операция сравнения < заменена на операцию ! =, т. к. операции < и > для итераторов в общем случае не поддерживаются.

Для всех контейнерных классов определены унифицированные методы **`begin()`** и **`end()`**, возвращающие адреса **`first`** и **`last`** соответственно.

В STL существуют следующие типы итераторов:

- входные,
- выходные,
- прямые,
- двунаправленные итераторы,
- итераторы произвольного доступа.

2.4. Общие свойства контейнеров

Таблица 1. Унифицированные типы, определенные в STL

Поле	Пояснение
size_type	Тип индексов, счетчиков элементов и т. д.
iterator	Итератор
const_iterator	Константный итератор (значения элементов изменять запрещено)
reference	Ссылка на элемент
const_reference	Константная ссылка на элемент (значение элементаизменять запрещено)
key_type	Тип ключа (для ассоциативных контейнеров)
key compare	Тип критерия сравнения (для ассоциативных контейнеров)

В табл. 2 представлены некоторые общие для всех контейнеров операции.
Таблица 2. Операции и методы, общие для всех контейнеров

Операция или метод	Пояснение
Операции равенства **(==)** и неравенства **(!=)**	Возвращают значение true или false
Операция присваивания **(=)**	Копирует один контейнер в другой
clear	Удаляет все элементы
insert	Добавляет один элемент или диапазон элементов
erase	Удаляет один элемент или диапазон элементов
size_type size() const	Возвращает число элементов
size_type max_size() const	Возвращает максимально допустимый размер контейнера
bool empty() const	Возвращает true, если контейнер пуст
iterator begin()	Возвращают итератор на начало контейнера (итерации будут производиться в прямом направлении)
iterator end()	Возвращают итератор на конец контейнера (итерации в прямом направлении будут закончены)
reverse_iterator begin()	Возвращают реверсивный итератор на конец контейнера (итерации будут производиться в обратном направлении)
reverse_iterator end()	Возвращают реверсивный итератор на начало контейнера (итерации в обратном направлении будут закончены)

2.5. Использование последовательных контейнеров

К основным последовательным контейнерам относятся вектор (**`vector`**), список (**`list`**) и двусторонняя очередь (**`deque`**).

Чтобы использовать последовательный контейнер, нужно включить в программу соответствующий заголовочный файл:

```
#include <vector>
#include <list>
#include <deque>
using namespace std;
```

Контейнер **вектор** является аналогом обычного массива, за исключением того, что он автоматически выделяет и освобождает память по мере необходимости. Контейнер эффективно обрабатывает произвольную выборку элементов с помощью операции индексации `[]` или метода `at`. Однако **вставка элемента в любую позицию, кроме конца вектора, неэффективна**. Для этого потребуется сдвинуть все последующие элементы путем копирования их значений. По этой же причине **неэффективным является удаление любого элемента, кроме последнего**.

Контейнер **список** организует хранение объектов в виде двусвязного списка. Каждый элемент списка содержит три поля: значение элемента, указатель на предшествующий и указатель на последующий элементы списка. Вставка и удаление работают эффективно для любой позиции элемента в списке. Однако список **не поддерживает произвольного доступа** к своим элементам: например, для выборки n-го элемента нужно последовательно выбрать предыдущие n-1 элементов.

Контейнер **двусторонняя очередь** во многом аналогичен вектору, элементы хранятся в непрерывной области памяти. Но в отличие от вектора двусторонняя очередь **эффективно поддерживает вставку и удаление первого элемента (так же, как и последнего)**.

Существует пять способов определить объект для последовательного контейнера.

1. Создать пустой контейнер:

```
vector<int> vecl;
list<double> listl;
```

2. Создать контейнер заданного размера и инициализировать его элементы значениями по умолчанию:

```
vector<int> vecl(100);
list<double> listl(20);
```

3. Создать контейнер заданного размера и инициализировать его элементы указанным значением:

```
vector<int> vecl(100, 0);
deque<float> decl(300, 0.0);
```

4. Создать контейнер и инициализировать его элементы значениями диапазона (**first**, **last**) элементов другого контейнера:

```
int arr[7] = (15, 2, 19, -3, 28, 6, 8);
vector<int> vl(arr, arr + 7);
list<int> lst(vl.beg() + 2, vl.end());
```

5. Создать контейнер и инициализировать его элементы значениями элементов другого однотипного контейнера:

```
vector<int> vl;
// добавить в vl элементы
vector<int> v2(vl);
```

Таблица 3. Методы, которые поддерживают последовательные контейнеры

Vector	Deque	List	Пояснение
push_back()	push_back(T&key)	push_back(T&key)	добавление в конец
pop_back()	pop_back()	pop_back()	удаление из конца
insert	push_front(T&key)	push_front(T&key)	добавление в начало
erase	pop_front()	pop_front()	удаление из начала
	insert	insert	вставка в произвольное место
	erase	erase	удаление из произвольного места
[] или at	[] или at		доступ к произвольному элементу
swap		swap	обмен векторов
clear()		clear()	очистить вектор
		splice	сцепка списков

Метод **insert** имеет следующие реализации:

- **`iterator insert(iterator pos, const T&key);`** - вставляет элемент **`key`** в позицию, на которую указывает итератор **`pos`**, возвращает итератор на вставленный элемент
- **`void insert(iterator pos, size_type n, const T&key);`** - вставляет **n** элементов **`key`** начиная с позиции, на которую указывает итератор **`pos`**
- **`template <class InputIter>`**
  `void insert(iterator pos, InputIter first, InputIter last);` - вставляет элементы диапазона **`first..last`**, начиная с позиции, на которую указывает итератор **`pos`**.

Пример использования метода **`insert():`**

```
void main()
{
  /*создать вектор из 5 элементов,
    проинициализировать элементы
    нулями*/
  vector <int>v1(5,0);
  int m[5]={1,2,3,4,5};//массив из 5 элементов
  //вставить элемент со значением 100 в начало вектора
  v1.insert(v1.begin(),100);
  /*вставить два элемента со значением 200
    после первого элемента вектора*/
  v1.insert(v1.begin()+1,2,200);
  //вставить элементы из массива m после третьего элемента
  v1.insert(v1.begin()+3,m,m+5);
  //вставить элемент 100 в конец вектора
  v1.insert(v1.end(),100);
  //вывести вектор на печать
  for(int i=0;i<v1.size();i++)
     cout<<v1[i]<<" ";
}
```

Результат работы программы будет следующим:

```
100 200 200 1 2 3 4 5 0 0 0 0 0 100
```

Метод **erase** имеет следующие реализации:

- **iterator erase(iterator pos);** - удаляет элемент, на который указывает итератор pos
- **iterator erase(iterator first,iterator last);** - удаляет диапазон элементов

Пример использования метода **erase()**:

```
void main()
{
   vector <int>v1;//создать пустой вектор
   int m[5]={1,2,3,4,5};
   int n,a;
   cout<<"\nn?";
   cin>>n;
   for(int i=0;i<n;i++){
      cout<<"?";
      cin>>a;
      //добавить в конец вектора элемент со значением a
      v1.push_back(a);
   }
   //вывод вектора
   for( i=0;i<v1.size();i++)
      cout<<v1[i]<<" ";
   cout<<"\n";
   /*удалить элемент из начала вектора и итератор
     поставить на начало вектора*/
   vector<int>::iterator iv=v1.erase(v1.begin());
   cout<<(*iv)<<"\n";//вывод первого элемента
   //вывод вектора
   for( i=0;i<v1.size();i++)
      cout<<v1[i]<<" ";
}
```

В процессе работы над программами, использующими контейнеры, часто приходится выводить на экран их текущее содержимое. Приведем шаблон функции, решающей эту задачу для любого типа контейнера:

```
//функция для печати последовательного контейнера
template<class T>
void print(char*String,T&C)
{  T::iterator p=C.begin();
   cout<<String<<endl;
   if(C.empty())
      cout<<"\nEmpty!!!\n";
   else
      for(;p!=C.end();p++)
         cout<<*p<<" ";
   cout<<"\n";
}
```

2.6. Адаптеры контейнеров

Специализированные последовательные контейнеры — стек, очередь и очередь с приоритетами — не являются самостоятельными контейнерными классами, а реализованы на основе рассмотренных выше классов, поэтому они называются адаптерами контейнеров.
По умолчанию для стека прототипом является класс **deque**.

Объявление **stack<int> s** создает стек на базе двусторонней очереди (по умолчанию). Если по каким-то причинам нас это не устраивает, и мы хотим создать стек на базе списка, то объявление будет выглядеть следующим образом:

```
stack<int, list<int> > s;
```

Смысл такой реализации заключается в том, что специализированный класс просто переопределяет интерфейс класса-прототипа, ограничивая его только теми методами, которые нужны новому классу. Стек не позволяет выполнить произвольный доступ к своим элементам, а также не дает возможности пошагового перемещения, т. е. итераторы в стеке не поддерживаются.

Методы класса stack:

- **push ()** - добавление в конец;
- **pop ()** - удаление из конца;
- **top ()** - получение текущего элемента стека;
- **empty()** - проверка пустой стек или нет;
- **size ()** - получение размера стека.

Шаблонный класс **queue** (заголовочный файл **<queue>**) является адаптером, который может быть реализован на основе двусторонней очереди (реализация по умолчанию) или списка. Класс **vector** в качестве класса-прототипа не подходит, поскольку в нем нет выборки из начала контейнера. Очередь использует для проталкивания данных один конец, а для выталкивания — другой.

Методы класса **queue**:

- **push ()** – добавление в конец очереди;
- **pop ()** – удаление из начала очереди;
- **front()** – получение первого элемента очереди;
- **back()** - получение последнего элемента очереди;
- **empty()** – проверка пустая очередь или нет;
- **size()** – получение размера очереди.

Шаблонный класс **priority_queue** (заголовочный файл **<queue>**)
поддерживает такие же операции, как и класс queue, но реализация класса возможна либо на основе вектора (реализация по умолчанию), либо на основе списка. Очередь с приоритетами отличается от обычной очереди тем, что для извлечения выбирается максимальный элемент из хранимых в контейнере. Поэтому после каждого изменения состояния очереди максимальный элемент из оставшихся сдвигается в начало контейнера .

Методы класса **priority_queue:**

- **push ()** – добавление в конец очереди;
- **pop ()** – удаление из начала очереди;
- **front ()** – получение первого элемента очереди;
- **back()** - получение последнего элемента очереди;
- **empty ()** – проверка пустая очередь или нет;
- **size()** – получение размера очереди.

Рассмотрим пример использования очереди с приоритетами:

```
#include <queue>
void main()
{  priority_queue <int> P;//очередь с приоритетами

   //добавить элементы
   P.push(17); P.push(5); P.push(400); P.push(2500); P.push(1);
   //пока очередь не пустая
   while (!P.empty()){
      cout<<P.top()<<' ';     //вывести первый элемент
      P.pop();                //удалить элемент из начала
   }
}
```

Результат работы программы: `2500 400 17 5 1`

3 Постановка задачи

Лабораторная работа состоит из пяти задач. Ниже представлены задачи, то что нужно выполнить в этой лабораторной работе. По каждой из задач в функции main показать использование методов либо глобальных функций.

Задача 1.

1. Создать **последовательный контейнер**.
2. Заполнить его элементами стандартного типа (тип указан в варианте).
3. Добавить элементы в соответствии с заданием
4. Удалить элементы в соответствии с заданием.
5. Выполнить задание варианта для полученного контейнера.
6. Выполнение всех заданий оформить в виде **глобальных функций**.

Задача 2.

1. Создать **последовательный контейнер**.
2. Заполнить его элементами пользовательского типа (тип указан в варианте). Для пользовательского типа перегрузить необходимые операции.
3. Добавить элементы в соответствии с заданием
4. Удалить элементы в соответствии с заданием.
5. Выполнить задание варианта для полученного контейнера.
6. Выполнение всех заданий оформить в виде **глобальных функций**.

Задача 3

1. Создать **параметризированный класс**, используя в качестве контейнера последовательный контейнер.
2. Заполнить его элементами.
3. Добавить элементы в соответствии с заданием
4. Удалить элементы в соответствии с заданием.
5. Выполнить задание варианта для полученного контейнера.
6. Выполнение всех заданий оформить в виде **методов параметризированного класса**.

Задача 4

1. Создать **адаптер контейнера**.
2. Заполнить его элементами пользовательского типа (тип указан в варианте). Для пользовательского типа перегрузить необходимые операции.
3. Добавить элементы в соответствии с заданием
4. Удалить элементы в соответствии с заданием.
5. Выполнить задание варианта для полученного контейнера.
6. Выполнение всех заданий оформить в виде **глобальных функций**.

Задача 5

1. Создать **параметризированный класс**, используя в качестве контейнера адаптер контейнера.
2. Заполнить его элементами.
3. Добавить элементы в соответствии с заданием
4. Удалить элементы в соответствии с заданием.
5. Выполнить задание варианта для полученного контейнера.
6. Выполнение всех заданий оформить в виде **методов параметризированного класса**.

4 Варианты заданий и исходных данных

Вариант	номер задачи	тип контейнера	тип элементов	Что нужно сделать к каждой задаче варианта
1	1	*вектор*	*int*	Пункт 3 Задача X(X = 1...5) Найти максимальный элемент и добавить его в начало контейнера
	2	*вектор*	*АТД — time*	
	3	*параметризованный класс (шаблон) вектор*	*Int, АТД — time*	Пункт 4 Задача X(X = 1...5) Найти минимальный элемент и удалить его из контейнера
	4	*Адаптер контейнера — стек*	*Int, АТД — time*	
	5	*Параметризированный класс – Вектор Адаптер контейнера - стек.*	*Int, АТД — time*	Пункт 5 Задача X(X = 1...5) К каждому элементу добавить среднее арифметическое контейнера
2	1	*список*	*float*	Пункт 3 Задача X(X = 1...5) Найти минимальный элемент и добавить его в конец контейнера
	2	*список*	*АТД — time*	
	3	*параметризованный класс (шаблон) очередь*	*float, АТД — time*	Пункт 4 Задача X(X = 1...5) Найти элемент с заданным ключом и удалить его из контейнера
	4	*Адаптер контейнера — очередь*	*float, АТД — time*	
	5	*Параметризированный класс – Вектор Адаптер контейнера — очередь.*	*float, АТД — time*	Пункт 5 Задача X(X = 1...5) К каждому элементу добавить сумму минимального и максимального элементов контейнера
3	1	*двунаправленная очередь*	*double*	Пункт 3 Задача X(X = 1...5) Найти элемент с заданным значением и добавить его на заданную позицию контейнера
	2	*двунаправленная очередь*	*АТД — time*	
	3	*параметризованный класс (шаблон) вектор*	*double, АТД — time*	Пункт 4 Задача X(X = 1...5) Найти элемент с заданным значением и удалить его из контейнера
	4	*Адаптер контейнера — очередь с приоритетами*	*double, АТД — time*	
	5	*Параметризированный класс – Вектор Адаптер контейнера — очередь с приоритетами.*	*double, АТД — time*	Пункт 5 Задача X(X = 1...5) Найти разницу между максимальным и минимальным элементами контейнера и вычесть ее из каждого элемента контейнера
4	1	*двунаправленная очередь*	*int*	Пункт 3 Задача X(X = 1...5) Найти максимальный элемент и добавить его в конец контейнера
	2	*двунаправленная очередь*	*АТД — time*	
	3	*параметризованный класс (шаблон) очередь*	*Int, АТД — time*	Пункт 4 Задача X(X = 1...5) Найти элемент с заданным ключом и удалить его из контейнера
	4	*Адаптер контейнера — очередь*	*Int, АТД — time*	
	5	*Параметризированный класс – Вектор Адаптер контейнера — очередь.*	*Int, АТД — time*	Пункт 5 Задача X(X = 1...5) К каждому элементу добавить среднее арифметическое элементов контейнера
5	1	*список*	*float*	Пункт 3 Задача X(X = 1...5) Найти минимальный элемент и добавить его на заданную позицию контейнера
	2	*список*	*АТД — time*	
	3	*параметризованный класс (шаблон) вектор*	*float, АТД — time*	Пункт 4 Задача X(X = 1...5) Найти элементы большие среднего арифметического и удалить их из контейнера
	4	*Адаптер контейнера — вектор*	*float, АТД — time*	
	5	*Параметризированный класс – Вектор Адаптер контейнера — стек.*	*float, АТД — time*	Пункт 5 Задача X(X = 1...5) Каждый элемент домножить на максимальный элемент контейнера
6	1	*вектор*	*double*	Пункт 3 Задача X(X = 1...5) Найти максимальный элемент и добавить его в начало контейнера
	2	*вектор*	*АТД — Money*	
	3	*параметризованный класс (шаблон) Множество*	*double, АТД — Money*	Пункт 4 Задача X(X = 1...5) Найти минимальный элемент и удалить его из контейнера
	4	*Адаптер контейнера — стек*	*double, АТД — Money*	
	5	*Параметризированный класс – Вектор Адаптер контейнера — стек.*	*double, АТД — Money*	Пункт 5 Задача X(X = 1...5) К каждому элементу добавить среднее арифметическое контейнера

Вариант	номер задачи	тип контейнера	тип элементов	Что нужно сделать к каждой задаче варианта
7	1	*вектор*	*float*	Пункт 3 Задача X(X = 1...5) Найти минимальный элемент и добавить его в конец контейнера
	2	*вектор*	*АТД — Money*	
	3	*параметризованный класс (шаблон) вектор*	*float, АТД — Money*	Пункт 4 Задача X(X = 1...5) Найти элемент с заданным ключом и удалить его из контейнера
	4	*Адаптер контейнера — очередь*	*float, АТД — Money*	
	5	*Параметризированный класс – Вектор Адаптер контейнера — очередь.*	*float, АТД — Money*	Пункт 5 Задача X(X = 1...5) К каждому элементу добавить сумму минимального и максимального элементов контейнера
8	1	*список*	*double*	Пункт 3 Задача X(X = 1...5) Найти элемент с заданным значением и добавить его на заданную позицию контейнера
	2	*список*	*АТД — Money*	
	3	*параметризованный класс (шаблон) вектор*	*double, АТД — Money*	Пункт 4 Задача X(X = 1...5) Найти элемент с заданным ключом и удалить его из контейнера
	4	*Адаптер контейнера — очередь с приоритетами*	*double, АТД — Money*	
	5	*Параметризированный класс – Вектор Адаптер контейнера — очередь с приоритетами.*	*double, АТД — Money*	Пункт 5 Задача X(X = 1...5) Найти разницу между максимальным и минимальным элементами контейнера и вычесть ее из каждого элемента контейнера
9	1	*двунаправленная очередь*	*int*	Пункт 3 Задача X(X = 1...5) Найти максимальный элемент и добавить его в конец контейнера
	2	*двунаправленная очередь*	*АТД — Money*	
	3	*параметризованный класс (шаблон) Вектор*	*int, АТД — Money*	Пункт 4 Задача X(X = 1...5) Найти элемент с заданным ключом и удалить его из контейнера
	4	*Адаптер контейнера — стек*	*int, АТД — Money*	
	5	*Параметризированный класс – Вектор Адаптер контейнера — стек.*	*int, АТД — Money*	Пункт 5 Задача X(X = 1...5) К каждому элементу добавить среднее арифметическое элементов контейнера
10	1	*вектор*	*float*	Пункт 3 Задача X(X = 1...5) Найти минимальный элемент и добавить его на заданную позицию контейнера
	2	*вектор*	*АТД — Money*	
	3	*параметризованный класс (шаблон) Вектор*	*float, АТД — Money*	Пункт 4 Задача X(X = 1...5) Найти элементы большие среднего арифметического и удалить их из контейнера
	4	*Адаптер контейнера — очередь с приоритетами*	*float, АТД — Money*	
	5	*Параметризированный класс – Вектор Адаптер контейнера — очередь с приоритетами.*	*float, АТД — Money*	Пункт 5 Задача X(X = 1...5) Каждый элемент домножить на максимальный элемент контейнера
11	1	*вектор*	*float*	Пункт 3 Задача X(X = 1...5) Найти среднее арифметическое и добавить его в начало контейнера
	2	*вектор*	*АТД — Money*	
	3	*параметризованный класс (шаблон) Список*	*float, АТД — Money*	Пункт 4 Задача X(X = 1...5) Найти элемент с заданным значением и удалить их из контейнера
	4	*Адаптер контейнера — очередь*	*float, АТД — Money*	
	5	*Параметризированный класс – Список Адаптер контейнера — очередь.*	*float, АТД — Money*	Пункт 5 Задача X(X = 1...5) Из каждого элемента вычесть минимальный элемент контейнера
12	1	*список*	*int*	Пункт 3 Задача X(X = 1...5) Найти среднее арифметическое и добавить его на заданную позицию контейнера
	2	*список*	*АТД — Point*	
	3	*параметризованный класс (шаблон) Список*	*int, АТД — Point*	Пункт 4 Задача X(X = 1...5) Найти элементы с ключами из заданного диапазона и удалить их из контейнера
	4	*Адаптер контейнера — очередь с приоритетами*	*int, АТД — Point*	
	5	*Параметризированный класс – Список Адаптер контейнера — очередь с приоритетами.*	*int, АТД — Point*	Пункт 5 Задача X(X = 1...5) Из каждого элемента вычесть среднее арифметическое контейнера.

Вариант	номер задачи	тип контейнера	тип элементов	Что нужно сделать к каждой задаче варианта
13	1	*двунаправленная очередь*	*double*	Пункт 3 Задача X(X = 1...5) Найти максимальный элемент и добавить его в конец контейнера
	2	*двунаправленная очередь*	*АТД — Point*	
	3	*параметризованный класс (шаблон) Список*	*double, АТД — Point*	Пункт 4 Задача X(X = 1...5) Найти элементы с ключами из заданного диапазона и удалить их из контейнера
	4	*Адаптер контейнера — стек*	*double, АТД — Point*	
	5	*Параметризированный класс – Список Адаптер контейнера — стек.*	*double, АТД — Point*	Пункт 5 Задача X(X = 1...5) К каждому элементу добавить среднее арифметическое контейнера
14	1	*вектор*	*float*	Пункт 3 Задача X(X = 1...5) Найти минимальный элемент и добавить его на заданную позицию контейнере
	2	*вектор*	*АТД — Point*	
	3	*параметризованный класс (шаблон) список*	*float, АТД — Point*	Пункт 4 Задача X(X = 1...5) Найти меньше среднего арифметического и удалить их из контейнера
	4	*Адаптер контейнера — очередь*	*float, АТД — Point*	
	5	*Параметризированный класс – Список Адаптер контейнера — очередь.*	*float, АТД — Point*	Пункт 5 Задача X(X = 1...5) Каждый элемент разделить на максимальный элемент контейнера.
15	1	*список*	*double*	Пункт 3 Задача X(X = 1...5) Найти среднее арифметическое и добавить его в конец контейнера
	2	*список*	*АТД — Point*	
	3	*параметризованный класс (шаблон) Список*	*double, АТД — Point*	Пункт 4 Задача X(X = 1...5) Найти элементы со значениями из заданного диапазона и удалить их из контейнера
	4	*Адаптер контейнера — очередь с приоритетами.*	*double, АТД — Point*	
	5	*Параметризированный класс – Список Адаптер контейнера — очередь с приоритетами.*	*double, АТД — Point*	Пункт 5 Задача X(X = 1...5) К каждому элементу добавить сумму минимального и максимального элементов контейнера.

5 Контрольные вопросы

1. Из каких частей состоит библиотека STL?
2. Какие типы контейнеров существуют в STL?
3. Что нужно сделать для использования контейнера STL в своей программе?
4. Что представляет собой итератор?
5. Какие операции можно выполнять над итераторами?
6. Каким образом можно организовать цикл для перебора контейнера с использованием итератора?
7. Какие типы итераторов существуют?
8. Перечислить операции и методы общие для всех контейнеров.
9. Какие операции являются эффективными для контейнера vector? Почему?
10. Какие операции являются эффективными для контейнера list? Почему?
11. Какие операции являются эффективными для контейнера deque? Почему?
12. Перечислить методы, которые поддерживает последовательный контейнер vector.
13. Перечислить методы, которые поддерживает последовательный контейнер list.
14. Перечислить методы, которые поддерживает последовательный контейнер deque.
15. Задан контейнер vector. Как удалить из него элементы со 2 по 5?
16. Задан контейнер vector. Как удалить из него последний элемент?
17. Задан контейнер list. Как удалить из него элементы со 2 по 5?
18. Задан контейнер list. Как удалить из него последний элемент?
19. Задан контейнер deque. Как удалить из него элементы со 2 по 5?
20. Задан контейнер deque. Как удалить из него последний элемент?
21. Написать функцию для печати последовательного контейнера с использованием итератора.
22. Что представляют собой адаптеры контейнеров?

23. Чем отличаются друг от друга объявления stack<int> s и stack<int, list<int> > s?
24. Перечислить методы, которые поддерживает контейнер stack.
25. Перечислить методы, которые поддерживает контейнер queue.
26. Чем отличаются друг от друга контейнеры queue и priority_queue?
27. Задан контейнер stack. Как удалить из него элемент с заданным номером?
28. Задан контейнер queue. Как удалить из него элемент с заданным номером?
29. Написать функцию для печати контейнера stack с использованием итератора.
30. Написать функцию для печати контейнера queue с использованием итератора.

6 Пример:

Задача 1

1. Контейнер – вектор;
2. тип элементов - int;
3. Найти среднее арифметическое вектора и добавить его в вектор под номером k.
4. Удалить максимальный элемент из вектора.
5. Каждый элемент разделить на минимальное значение вектора.

1. Создать пустой проект.
2. Добавить в проект файл Lab1_main.cpp, содержащий основную программу:

```
#include <iostream>
#include <vector>
#include <cstdlib>
using namespace std;
typedef vector<int> Vec; //определяем тип для работы с вектором
//функция для формирования вектора
Vec make_vector(int n) {
	Vec v; //пустой вектор
	for (int i = 0; i < n; i++) {
		int a = rand() % 100 - 50;
		v.push_back(a); //добавляем a в конец вектора
	}
	return v; //возвращаем вектор как результа работы функции
}
//функция для печати вектора
void print_vector(Vec v) {
	for (int i = 0; i < v.size(); i++)
		cout << v[i] << " ";
	cout << endl;
}
```

```
//основная функция
int main() {
	try {
		vector<int> v; //вектор
		vector<int>::iterator vi=v.begin();//итератор
		int n;
		cout << "N?";
		cin >> n;
		v = make_vector(n); //формирование вектора
		print_vector(v); //печать вектора
	} catch (int) //блок обработки ошибок
	{
		cout << "error!";
	}
	return 0;
}
```

3. Запустить программу на выполнение и протестировать ее работу.

4. Добавить функции для выполнения задания 3.

```
//вычисление среднего арифметического
int srednee(Vec v) {
	int s = 0;
	//перебор вектора
	for (int i = 0; i < v.size(); i++)
		s += v[i];
	int n = v.size(); //количество элементов в векторе
	return s / n;
}

//добавление на место pos элемент el
void add_vector(Vec &v, int el, int pos) {
	v.insert(v.begin() + pos, el);
}
```

5. Добавить в функцию main() вызов функций для выполнения задания 3.

```
//основная функция
int main() {
	try {
```

```
		vector<int> v;
		vector<int>::iterator vi = v.begin();
		//формирование вектора
		int n;
		cout << "N?";
		cin >> n;
		v = make_vector(n);
		print_vector(v);
		//вычисление среднего
		int el = srednee(v);
		//позиция для вставки
		cout << "pos?";
		int pos;
		cin >> pos;
		//генерируем ошибку, если позиция для вставки больше размера
вектора
		if (pos > v.size())
			throw 1;
		//вызов функции для добавления
		add_vector(v, el, pos);
		//печать вектора
		print_vector(v);
	} catch (int) //блок обработки ошибок
	{
		cout << "error!";
	}
	return 0;
}
```

6. Запустить программу на выполнение и протестировать ее работу.

7. Добавить функции для выполнения задания 4.

```
//поиск максимального элемента
int max(Vec v) {
	int m = v[0], //минимальный элемент
			n = 0; //номер минимального элемента
	for (int i = 0; i < v.size(); i++) //перебор вектора
		if (m < v[i]) {
			m = v[i]; //максимальный элемент
```

```
                        n = i; //номер максимального
                }
        return n;
}
//удалить элемент из позиции pos
void del_vector(Vec &v, int pos) {
        v.erase(v.begin() + pos);
}
```

7. Добавить в функцию main() вызов функций для выполнения задания 4.

```
//основная функция
int main() {
        try {
                …
                int n_max=max(v);//найти номер максимального
                del_vector(v,n_max);//удалить элемент с этим номером
                print_vector(v);
        } catch (int) //блок обработки ошибок
        {
                cout << "error!";
        }
        return 0;
}
```

9. Запустить программу на выполнение и протестировать ее работу.

10. Добавить функции для выполнения задания 5.

```
//поиск минимального элемента
int min(Vec v) {
        int m = v[0], //минимальный элемент
                        n = 0; //номер минимального элемента
        for (int i = 0; i < v.size(); i++) //перебор вектора
                if (m > v[i]) {
                        m = v[i]; //минимальный элемент
                        n = i; //номер минимального
                }
        return n;
}
void delenie(Vec &v) {
        int m = min(v);
```

```
        int min = v[m];
        for (int i = 0; i < v.size(); i++)
                v[i] = v[i] / min;
}
```

11. Добавить в функцию main() вызов функций для выполнения задания 5.

```
//основная функция
int main() {
        try {
                …
                //каждый элемент разделить на минимальное значение вектора
                delenie(v);//
                print_vector(v);
        } catch (int) //блок обработки ошибок
        {
                cout << "error!";
        }
        return 0;
}
```

12. Запустить программу на выполнение и протестировать ее работу.

Задача 2.

1. Контейнер - вектор.
2. Тип элементов Time.
3. Найти среднее арифметическое вектора и добавить его в вектор под номером k.
4. Удалить максимальный элемент из вектора.
5. Каждый элемент разделить на минимальное значение вектора.
6. Выполнение всех заданий оформить в виде глобальных функций.

1. Добавить новый проект в существующее решение.

2. Добавить в него класс Time. В результате будут сформированы 2 файла: Time.h, Time.cpp. В файл Time.h ввести описание класса Time:

```
#include <iostream>
using namespace std;
class Time {
        int min, sec;
public:
        Time() {
                min = 0;
                sec = 0;
```

```
        }
        Time(int m, int s) {
                min = m;
                sec = s;
        }
        Time(const Time&t) {
                min = t.min;
                sec = t.sec;
        }
        ~Time() {
        }
        int get_min() {
                return min;
        }
        int get_sec() {
                return sec;
        }
        void set_min(int m) {
                min = m;
        }
        void set_sec(int s) {
                sec = s;
        }
        //перегруженные операции
        Time&operator=(const Time&);
        //глобальные функции ввода-вывода
        friend istream& operator>>(istream&in, Time&t);
        friend ostream& operator<<(ostream&out, const Time&t);
};
```

3. В файл Time.cpp ввести определения глобальных функций и операции присваивания:

```
//перегрузка операции присваивания
Time&Time::operator=(const Time&t)
{
//проверка на самоприсваивание
        if(&t==this) return *this;
        min=t.min;
        sec=t.sec;
```

```
        return *this;
}
//перегрузка глобальной функции-операции ввода
istream&operator>>(istream&in, Time&t) {
        cout << "min?";
        in >> t.min;
        cout << "sec?";
        in >> t.sec;
        return in;
}
//перегрузка глобальной функции-операции вывода
ostream&operator<<(ostream&out, const Time&t) {
        return (out << t.min << " : " << t.sec);
}
```

4. Проанализировать какие операции требуется перегрузить для класса Time, чтобы можно было решить поставленные выше задачи для данных типа Time. Для поиска максимального и минимального значений требуется перегрузить операции > и <, т. к. необходимо сравнивать значения типа Time друг с другом. Для поиска среднего арифметического необходимо выполнить деление на целое число и сложение переменных типа Time друг с другом. Для деления элементов вектора на минимальное значение необходимо перегрузить операцию деления на переменную типа Time.

5. Перегрузить указанные операции. Для этого добавить в файл Time.h прототипы методов:

```
#include <iostream>
using namespace std;
class Time {
        int min, sec;
public:
        Time() {
                min = 0;
                sec = 0;
        }
        Time(int m, int s) {
                min = m;
                sec = s;
        }
        Time(const Time&t) {
                min = t.min;
                sec = t.sec;
        }
        ~Time() {
```

```
        }
        int get_min() {
                return min;
        }
        int get_sec() {
                return sec;
        }
        void set_min(int m) {
                min = m;
        }
        void set_sec(int s) {
                sec = s;
        }
        //перегруженные операции
        Time&operator=(const Time&);
        Time operator+(const Time&);
        Time operator/(const Time&);
        Time operator/(const int&);
        bool operator >(const Time&);
        bool operator <(const Time&);
        //глобальные функции ввода-вывода
        friend istream& operator>>(istream&in, Time&t);
        friend ostream& operator<<(ostream&out, const Time&t);
};
```

6. В файл Time.cpp добавить определения методов:

```
bool Time::operator <(const Time &t) {
        if (min < t.min)
                return true;
        if (min == t.min && sec < t.sec)
                return true;
        return false;
}
bool Time::operator >(const Time &t) {
        if (min > t.min)
                return true;
        if (min == t.min && sec > t.sec)
                return true;
        return false;
```

```
}
Time Time::operator+(const Time&t) {
	int temp1 = min * 60 + sec;
	int temp2 = t.min * 60 + t.sec;
	Time p;
	p.min = (temp1 + temp2) / 60;
	p.sec = (temp1 + temp2) % 60;
	return p;
}
//перегрузка бинарной операции деления
Time Time::operator/(const Time&t) {
	int temp1 = min * 60 + sec;
	int temp2 = t.min * 60 + t.sec;
	Time p;
	p.min = (temp1 / temp2) / 60;
	p.sec = (temp1 / temp2) % 60;
	return p;
}
Time Time::operator/(const int&t) {
	int temp1 = min * 60 + sec;
	Time p;
	p.min = (temp1 / t) / 60;
	p.sec = (temp1 / t) % 60;
	return p;
}
```

7. Добавить в проект файл zadacha2_main. Ввести следующий текст программы:

```
#include <iostream>
#include <vector>
#include <cstdlib>
#include "Time.h"
using namespace std;
typedef vector<Time> Vec; //определяем тип для работы с вектором
//функция для формирования вектора
Vec make_vector(int n) {
	Vec v; //пустой вектор
	for (int i = 0; i < n; i++) {
		int min = abs( rand() % 60);
```

```
            int sec = abs(rand() % 60);
            v.push_back(Time(min, sec)); //добавляем а в конец вектора
        }
        return v; //возвращаем вектор как результа работы функции
}
//функция для печати вектора
void print_vector(Vec v) {
        for (int i = 0; i < v.size(); i++)
            cout << v[i] << " ";
        cout << endl;
}
//основная функция
int main()
{
        try
        {
            vector<Time> v; //вектор
            vector<Time>::iterator vi=v.begin();//итератор
            int n;
            cout<<"N?"; cin>>n;
            v=make_vector(n);//формирование вектора
            print_vector(v);//печать вектора
        }
        catch(int) //блок обработки ошибок
        {
            cout<<"error!";
        }
        return 0;
}
```

8. Запустить программу на выполнение и протестировать ее работу.

9. Добавить функции для выполнения задания 3.

```
//вычисление среднего арифметического
Time srednee(Vec v) {
        Time s;
        //перебор вектора
        for (int i = 0; i < v.size(); i++)
            s = s + v[i];
        int n = v.size(); //количество элементов в векторе
```

```
	return s/n;
}

void add_vector(Vec &v, Time el, int pos) {
	//добавляем на место pos элемент el
	v.insert(v.begin() + pos, el);
}
```

10. Добавить в функцию main() вызов функций для выполнения задания 3.

```
//основная функция
int main()
{
	try {
		vector<Time> v;
		vector<Time>::iterator vi = v.begin();
		//формирование вектора
		int n;
		cout << "N?";
		cin >> n;
		v = make_vector(n);
		print_vector(v);
		//вычисление среднего
		Time el = srednee(v);
		//позиция для вставки
		cout << "pos?";
		int pos;
		cin >> pos;
		//генерируем ошибку, если позиция для вставки больше размера
вектора
		if (pos > v.size())
			throw 1;
		//вызов функции для добавления
		add_vector(v, el, pos);
		//печать вектора
		print_vector(v);
	} catch (int) //блок обработки ошибок
	{
		cout << "error!";
	}
```

```
	return 0;
}
```

11. Запустить программу на выполнение и протестировать ее работу.

12. Добавить функции для выполнения задания 4.

```
//поиск максимального элемента
int max(Vec v) {
	Time m = v[0]; //минимальный элемент
	int     n = 0; //номер минимального элемента
	for (int i = 0; i < v.size(); i++) //перебор вектора
		if (m < v[i]) {
			m = v[i]; //максимальный элемент
			n = i; //номер максимального
		}
	return n;
}

//удалить элемент из позиции pos
void del_vector(Vec &v, int pos) {
	v.erase(v.begin() + pos);
}
```

13. Добавить в функцию main() вызов функций для выполнения задания 4.

```
//основная функция
int main()
{
	try {
		...
		int n_max=max(v);//найти номер максимального
		del_vector(v,n_max);//удалить элемент с этим номером
		print_vector(v);
	} catch (int) //блок обработки ошибок
	{
		cout << "error!";
	}

	return 0;
}
```

14. Запустить программу на выполнение и протестировать ее работу.

15. Добавить функции для выполнения задания 5.

```
//поиск минимального элемента
int min(Vec v) {
	Time m = v[0]; //минимальный элемент
	int n = 0; //номер минимального элемента
	for (int i = 0; i < v.size(); i++) //перебор вектора
		if (m > v[i]) {
			m = v[i]; //минимальный элемент
			n = i; //номер минимального
		}
	return n;
}
void delenie(Vec &v) {
	int m = min(v);
	Time min = v[m];
	for (int i = 0; i < v.size(); i++){
		v[i]=v[i]/min;
	}
}
```

16. Добавим в функцию main() вызов функций для выполнения задания 4.

```
//основная функция
int main()
{
	try {
	...
		delenie(v);//делене на минимальный элемент
		print_vector(v);
	} catch (int) //блок обработки ошибок
	{
		cout << "error!";
	}

	return 0;
}
```

17. Запустить программу на выполнение и протестировать ее работу.

Задача 3

1. Контейнер - вектор.
2. Тип элементов Time .
3. Найти среднее арифметическое вектора и добавить его в вектор под номером k.
4. Удалить максимальный элемент из вектора.
5. Каждый элемент разделить на минимальное значение вектора.
6. Выполнение всех заданий оформить в виде методов параметризированного класса Вектор.

1. Добавить новый проект в существующее решение.

2. Добавить в проект класс Time из задачи 2.

3. Добавить в проект параметризированный класс Vector.

4. Определить шаблон Vector с минимальной функциональностью следующим образом:

```
#pragma once
#include <vector>
#include <iostream>
using namespace std;
//шаблон класса
template<class T>
class Vector {
	vector<T> v; //последовательный контейнер для хранения элементов вектора
	int len;
public:
	Vector(void); //конструктор без параметра
	Vector(int n); //конструктор с параметром
	void Print(); //печать
	~Vector(void); //деструктор
};
```

5. Добавить в этот же файл определение параметризированных функций:

```
//конструктор без параметра
template<class T>
Vector<T>::Vector() {
	len = 0;
}
//деструктор
template<class T>
Vector<T>::~Vector(void) {
}
//конструктор с параметром
template<class T>
```

```
Vector<T>::Vector(int n) {
	T a;
	for (int i = 0; i < n; i++) {
		cin >> a;
		v.push_back(a);
	}
	len = v.size();
}
//печать
template<class T>
void Vector<T>::Print() {
	for(int i=0;i<v.size();i++)
	cout << v[i] << " ";
	cout << endl;
}
```

6. Добавить файл zadacha3_main. Ввести следующий текст программы:

```
//полный путь к файлу Time.h
#include "Time.h"
#include "Vector.h"
#include <iostream>
using namespace std;
int main() {
	Vector<Time> vec(5); //создать вектор из 5 элементов
	vec.Print(); //печать вектора
	return 0;
}
```

7. Запустить программу на выполнение и протестировать ее работу.

8. Добавить в описание вектора функции для выполнения задания 2:

```
//шаблон класса
template<class T>
class Vector {
	vector<T> v; //последовательный контейнер для хранения элементов вектора
	int len;
public:
	...
	T Srednee();//вычисление среднего арифметического
	void Add(T el, int pos);//добавление элемента el на позицию pos
```

```
};
```

9. Добавить в файл Vector.h определение функции:

```
//вычислить среднее арифметическое
template<class T>
T Vector<T>::Srednee() {
	T s = v[0];
	for (int i = 1; i < v.size(); i++)
		s = s + v[i];
	int n = v.size();
	return s / n;
}
//добавление элемента
template<class T>
void Vector<T>::Add(T el, int pos) {
	v.insert(v.begin() + pos, el);
}
```

10. Добавить в файл zadacha3_main вызов функций:

```
int main() {
	Vector<Time> vec(5); //создать вектор из 5 элементов
	vec.Print(); //печать вектора
	Time s=vec.Srednee();//среднее ариметическое
	cout<<"Srednee="<<s<<endl;
	cout<<"pos?";
	int p;
	cin>>p;//ввести позицию для добавления
	vec.Add(s,p);//добавить элемент в вектор
	vec.Print();//печать вектора
	return 0;
}
```

11. Запустить программу на выполнение и протестировать ее работу.

12. Добавить в описание вектора функции для выполнения задания 3:

```
//шаблон класса
template<class T>
class Vector {
	vector<T> v; //последовательный контейнер для хранения элементов вектора
	int len;
```

```
public:
	...
	int Max();//найти номер максимальнго элемента
	void Del(int pos);//удалить элемент из позиции pos
};
```

13. Добавить в файл Vector.h определение функций:

```
template<class T>
int Vector<T>::Max() {
	T m = v[0];
	int n = 0;
	for (int i = 1; i < v.size(); i++)
		if (v[i] > m) {
			m = v[i];
			n = i;
		}
	return n;
}
//удаление элемента из позиции pos
template<class T>
void Vector<T>::Del(int pos) {
	v.erase(v.begin() + pos);
}
```

14. Добавить в файл zadacha2_main вызов функций:

```
int main() {
	...
	p=vec.Max();//найти максимальный элемент
	vec.Del(p);//удаление
	vec.Print();//печать
	return 0;
}
```

15. Запустить программу на выполнение и протестировать ее работу.

16. Добавить в описание вектора функции для выполнения задания 4:

```
//шаблон класса
template<class T>
class Vector {
```

```
	vector<T> v; //последовательный контейнер для хранения элементов вектора
	int len;
public:
	...
	int Min();//найти номер минимальнго элемента
	void Delenie();//деление на минимальный
};
```

17. Добавить в файл Vector.h определение функций:

```
//поиск минимального элемента
template<class T>
int Vector<T>::Min() {
	T m = v[0];
	int n = 0;
	for (int i = 1; i < v.size(); i++)
		if (v[i] < m) {
			m = v[i];
			n = i;
		}
	return n;
}
//деление всех элементов вектора на минимальный элемент
template<class T>
void Vector<T>::Delenie() {
	int m = Min();
	T min = v[m];
	for (int i = 0; i < v.size(); i++)
		v[i] = v[i] / min;
}
```

18. Добавить в файл zadacha3_main вызов функций:

```
int main() {
	...
	vec.Delenie();//деление
	vec.Print();//печать
	return 0;
}
```

19. Запустить программу на выполнение и протестировать ее работу.

Задача 4.

1. Адаптер контейнера - стек.
2. Тип элементов Time.
3. Найти среднее арифметическое стека и добавить его в стек под номером k.
4. Удалить максимальный элемент из стека.
5. Каждый элемент стека разделить на минимальное значение стека.
6. Выполнение всех заданий оформить в виде глобальных функций.

1. Добавить новый проект в существующее решение.

2. Добавить в проект класс Time из задачи 2.

3. Добавить в проект файл zadacha4_main.

4. Подключить библиотеки с помощью директив препроцессора:

```
#include <iostream>
#include <stack>
#include <vector>
#include "Time.h"
using namespace std;
```

5. В программе будем использовать контейнер типа vector в качестве вспомогательного контейнера, поэтому определим два типа:

```
typedef stack<Time>St;//стек
typedef vector<Time>Vec;//вектор
```

6. Определим функцию для формирования стека:

```
St make_stack(int n) {
	St s;
	Time t;
	for (int i = 0; i < n; i++) {
		cin >> t; //ввод переменной
		s.push(t); //добавление ее в стек
	}
	return s; //вернуть стек как результат функции
}
```

7. При работе со стеком у нас есть доступ только к вершине стека. Чтобы получить доступ к следующему элементу, элемент из вершины нужно удалить. Потому перед просмотром стека его надо сохранить во вспомогательном контейнере типа vector (vector выбирается, т. к. с ним достаточно просто работать). Для работы с вектором и стеком напишем вспомогательные функции:

```
//копирует стек в вектор
Vec copy_stack_to_vector(St s) {
	Vec v;
	while (!s.empty()) //пока стек не пустой
```

```
	{
		//добавить в вектор элемент из вершиы стека
		v.push_back(s.top());
		s.pop();
	}
	return v; //вернуть вектор как результат функции
}
//копирует вектор в стек
St copy_vector_to_stack(Vec v) {
	St s;
	for (int i = 0; i < v.size(); i++) {
		s.push(v[i]); //добавить в стек элемент вектора
	}
	return s; //вернуть стек как результат функции
}

//выводит стек
void print_stack(St s) {
	Vec v = copy_stack_to_vector(s);
	while (!s.empty()) //пока стек не пустой
		{
			cout << s.top() << endl; //вывод элемента в вершине стека
			s.pop(); //удаляем элемент из вершины
		}
	//копируем вектор в стек
	s = copy_vector_to_stack(v); //скопировать вектор в стек
	cout<<endl;
}
```

8. Добавить функцию main()

```
int main()
{
	Time t;
	St s;
	int n;
	cout<<"n?";
	cin>>n;
	s=make_stack(n); //создать стек
```

```
	print_stack(s);//печать стека
	return 0;
}
```

9. Запустить программу на выполнение и протестировать ее работу.

10. Добавить функции для выполнения задания 3.

```
//вычисление среднего значения
Time Srednee(St s) {
	Vec v = copy_stack_to_vector(s); //копируем стек в вектор
	int n = 1;
	Time sum = s.top(); //начальное значениедля суммы
	s.pop(); //удалить первый элемент из вектора
	while (!s.empty()) //пока стек не пустой
	{
		sum = sum + s.top(); //добавить в сумму элемент из вершины стека
		s.pop(); //удалить элемент
		n++;
	}
	s = copy_vector_to_stack(v); //скопировать вектор в стек
	return sum / n; //вернуть среднее арифметическое
}
//добавление элемента в стек
void Add_to_stack(St &s, Time el, int pos) {
	Vec v;
	Time t;
	int i = 1; //номер элемента в стеке
	while (!s.empty()) //пока стек не пустой
	{
		t = s.top(); //получить элемент из вершины
		//если номер равен номеру позиции, на котоую добвляем элемент
		if (i == pos)
			v.push_back(el); //добавить новый элемент в вектор
		v.push_back(t); //добавить элемент из стека в вектор
		s.pop(); //удалить элемент из стека
		i++;
	}
	s = copy_vector_to_stack(v); //копируем вектор в стек
}
```

11. Добавить в функцию main() операторы для вызова функций, выполняющих задание 3:

```
int main()
{
	Time t;
	St s;
	int n;
	cout<<"n?";
	cin>>n;
	s=make_stack(n);//создать стек
	print_stack(s);//печать стека
	t=Srednee(s);//вычисляем среднее
	cout<<"Srednee="<<Srednee(s)<<endl;
	cout<<"Add srednee:"<<endl;
	cout<<"pos?";
	int pos;
	cin>>pos;//вводим позицию для добавления
	Add_to_stack(s,t,pos);//добавление элемента
	print_stack(s);//печать стека
	return 0;
}
```

12. Запустить программу на выполнение и протестировать ее работу.

13. Добавить функции для выполнения задания 4.

```
//поиск максимального элемента в стеке
Time Max(St s) {
	Time m = s.top(); //переменной m присваиваем значение из вершины стека
	Vec v = copy_stack_to_vector(s); //копируем стек в вектор
	while (!s.empty()) //пока стк не пустой
	{
		if (s.top() > m)
			m = s.top(); //срвниваем m и элемнт в вершине стека
		s.pop(); //удаляем элемент из стека
	}
	s = copy_vector_to_stack(v); //копиуем вектор в стек
	return m; //возвращаем m
}
//Удалить максимальный элемент из стека
void Delete_from_stack(St&s) {
```

```
	Time m = Max(s); //находим максимальный элемент
	Vec v;
	Time t;
	while (!s.empty()) //пока стек не пустой
	{
		t = s.top(); //получаем элемент из вершины стека
		//если он не равен максимальному, заносим элемент в вектор
		if (t != m)
			v.push_back(t);
		s.pop(); //удаляем элемент из стека
	}
	s = copy_vector_to_stack(v); //копируем вектор в стек
}
```

14. Добавить перегрзку операции != в класс Time

```
bool Time::operator !=(const Time &t) {
	if (min > t.min||min < t.min)
			return true;
		if (min == t.min && (sec > t.sec || sec < t.sec))
			return true;
	return false;
}
```

Добавить объявление перегрузки в Time.h

```
bool Time::operator !=(const Time &t)ж
```

15. Добавить в функцию main() операторы для вызова функций, выполняющих задание 4:

```
int main()
{
	...
	cout<<"Delete Max:"<<endl;
	Delete_from_stack(s);
	print_stack(s);
	return 0;
}
```

16. Запустить программу на выполнение и протестировать ее работу.

17. Добавим функции для выполнения задания 5.

```
//поиск минимального элемента
Time Min(St s) {
```

```
	Time m = s.top();
	Vec v = copy_stack_to_vector(s);
	while (!s.empty()) {
		if (s.top() < m)
			m = s.top();
		s.pop();
	}
	s = copy_vector_to_stack(v);
	return m;
}
//деление на минимальный
void Delenie(St&s) {
	Time m = Min(s); //находим минимальный элемент
	Vec v;
	Time t;
	while (!s.empty()) //пока стек не пустой
	{
		t = s.top();//получаем элемент из вершины
		v.push_back(t/m);//делим t на минимальный и добавляем в вектор
		s.pop(); //удаляем элемент из вершины
	}
	s = copy_vector_to_stack(v); //копируем вектор в стек
}
```

18. Добавить в функцию main() операторы для вызова функций, выполняющих задание 5:

```
int main()
{
	...
	cout<<"Delenie:"<<endl;
	Delenie(s);
	print_stack(s);
	return 0;
}
```

19. Запустить программу на выполнение и протестировать ее работу.

Задача 5

1. Адаптер контейнера - стек.
2. Тип элементов Time.

3. Найти среднее арифметическое стека и добавить его в стек под номером k.

4. Удалить максимальный элемент из стека.

5. Каждый элемент стека разделить на минимальное значение стека.

6. Выполнение всех заданий оформить в виде методов параметризированного класса.

1. Добавим новый проект в существующее решение.

2. Добавить в проект класс Time из задачи 2.

3. Добавить в проект параметризированный класс Vector.

4. Подключить библиотеки с помощью директив препроцессора:

```
#include <iostream>
#include <stack>
#include <vector>
```

5. Определить шаблон Vector с минимальной функциональностью следующим образом:

```
template<class T>
class Vector {
	stack<T> s; //контейнер
	int len; //размер контейнера
public:
	Vector(); //конструктор без параметров
	Vector(int n); //конструктор с параметрами
	Vector(const Vector<T>&); //конструктор копирования
	void Print();
};
```

6. Добавим в этот же файл шаблоны глобальных функций для работы с вектором и стеком:

```
//копирование стека в вектор
template<class T>
vector<T> copy_stack_to_vector(stack<T> s) {
	vector<T> v;
	while (!s.empty()) {
		v.push_back(s.top());
		s.pop();
	}
	return v;
}
//копирование вектора в стек
template<class T>
stack<T> copy_vector_to_stack(vector<T> v) {
	stack<T> s;
```

```
	for (int i = 0; i < v.size(); i++) {
		s.push(v[i]);
	}
	return s;
}
```

7. Добавить в этот же файл определение параметризированных функций:

```
//конструктор без параметров
template<class T>
Vector<T>::Vector() {
	len = 0;
}
//конструктор с параметром
template<class T>
Vector<T>::Vector(int n) {
	T a;
	for (int i = 0; i < n; i++) {
		cin >> a;
		s.push(a); //добавить в стек элемент a
	}
	len = s.size();
}
//конструктор копирования
template<class T>
Vector<T>::Vector(const Vector<T> &Vec) {
	len = Vec.len;
	//копируем значения стека Vec.s в вектор v
	Vector v = copy_stack_to_vector(Vec.s);
	//копируем вектор v в стек s
	s = copy_vector_to_stack(v);
}
//печать
template<class T>
void Vector<T>::Print() {
	//копируем стек в вектор
	vector<T> v = copy_stack_to_vector(s);
	while (!s.empty()) //пока стек не пустой
	{
```

```
            cout << s.top() << endl; //вывод элемента в вершине стека
            s.pop(); //удаляем элемент из вершины
        }
        //копируем вектор в стек
        s = copy_vector_to_stack(v);
}
```

8. Добавить в проект файл zadacha5_main.

9. Подключить библиотеки с помощью директив препроцессора:

```
#include <iostream>
#include <stack>
#include <vector>
#include "Time.h"
#include "Vector.h"
using namespace std;
```

10. Добавить функцию main()

```
int main() {
        Vector<Time> v(3);
        v.Print();
        return 0;
}
```

11. Запустить программу на выполнение и протестировать ее работу.

12. Добавить в класс Vector функции для выполнения задания 3:

```
template<class T>
class Vector {
        stack<T> s; //контейнер
        int len; //размер контейнера
public:
        ...
        T Srednee();
        void Add(T el, int pos);
};
```

13. Добавим в файл Vector.h параметризированные функции для выполнения задания 3:

```
//среднее арифметическое
template <class T>
T Vector<T>::Srednee()
```

```
{
	//копируем стек в вектор
	vector<T> v=copy_stack_to_vector(s);
	int n=1;//количество элементов в стеке
	T sum=s.top();//начальное значение для суммы
	s.pop();//удаляем элемент из вершины стека
	while(!s.empty())//пока стек не пустой
	{
		sum=sum+s.top(); //добавляем в сумму элемент из вершины стека
		s.pop();//удаляем элемент из вершины стека
		n++;//увеличиваем количество элементов
	}
	//копируем вектор в стек
	s=copy_vector_to_stack(v);
	return sum/n;
}
//добавление элемента el в стек на позицию pos
template<class T>
void Vector<T>::Add(T el, int pos) {
	vector<T> v; //вспомогательный вектор
	T t;
	int i = 1;
	while (!s.empty()) //пока стек не пустой
	{
		t = s.top(); //получить элемент из вершины стека
		//если номер элемента равен pos добавляем в вектор новый элемент
		if (i == pos)
			v.push_back(el);
		v.push_back(t); //добавляем t в вектор
		s.pop(); //удаляем элемент из вершины стека
		i++;
	}
	s = copy_vector_to_stack(v); //копируем вектор в стек
}
```

14. Добавить в функцию main() операторы для вызова функций, выполняющих задание 3:

```
int main() {
	Vector<Time>v(3);
	v.Print();
```

```
        Time t=v.Srednee();
        cout<<"\nSrednee="<<t<<endl;
        cout<<"Add srednee"<<endl;
        cout<<"pos?";
        int pos;
        cin>>pos;
        v.Add(t,pos);
        v.Print();
        return 0;
}
```

15. Запустить программу на выполнение и протестировать ее работу.

16. Добавим в класс Vector функции для выполнения задания 4:

```
template<class T>
class Vector {
        stack<T> s; //контейнер
        int len; //размер контейнера
public:
        ...
        T Max();
        void Del();
};
```

17. Добавим в файл Vector.h параметризированные функции для выполнения задания 4:

```
//поиск максимального элемента
template<class T>
T Vector<T>::Max() {
        T m = s.top(); //m присвоить значение из вершины стека
        //в вектор скопировать элементы стека
        vector<T> v = copy_stack_to_vector(s);
        while (!s.empty()) //пока стек не пустой
        {
                //сравниваем m и элемент в вершине стека
                if (s.top() > m)
                        m = s.top();
                s.pop(); //удаляем элемент из вершины стека
        }
        s = copy_vector_to_stack(v); //копируем вектор в стек
```

```
        return m;
}
//удаление элемента из вектора
template<class T>
void Vector<T>::Del() {
        T m = Max(); //поиск максимального
        vector<T> v;
        T t;
        while (!s.empty()) //пока стек не пустой
        {
                t = s.top(); //получить элемент из вершины стека
                //если t не равен максимальному, то добавить его в вектор
                if (t != m)
                        v.push_back(t);
                s.pop(); //удалить элемент из стека
        }
        //копируем вектор в стек
        s=copy_vector_to_stack(v);
}
```

18. Добавить в функцию main() операторы для вызова функций, выполняющих задание 4:

```
int main() {
        ...
        cout<<"Max="<<v.Max()<<endl;
        cout<<"Delete Max:"<<endl;
        v.Del();
        v.Print();
        return 0;
}
```

19. Запустить программу на выполнение и протестировать ее работу.

20. Добавить в класс Vector функции для выполнения задания 5:

```
class Vector {
        stack<T> s; //контейнер
        int len; //размер контейнера
public:
        ...
        T Min();
```

```
        void Delenie();
};
```

21. Добавить в файл Vector.h параметризированные функции для выполнения задания 5:

```
//поиск минимального элемента
template<class T>
T Vector<T>::Min() {
        T m = s.top();
        vector<T> v = copy_stack_to_vector(s);
        while (!s.empty()) {
                if (s.top() < m)
                        m = s.top();
                s.pop();
        }
        s = copy_vector_to_stack(v);
        return m;
}
//деление всех элементов на минимальный
template<class T>
void Vector<T>::Delenie() {
        T m = Min();
        vector<T> v;
        T t;
        while (!s.empty()) {
                t = s.top();
                v.push_back(t / m);
                s.pop();
        }
        s = copy_vector_to_stack(v);
}
```

22. Добавить в функцию main() операторы для вызова функций, выполняющих задание 5:

```
int main() {
        ...
        cout<<"Delenie na min"<<endl;
        cout<<"Min="<<v.Min()<<endl;
        v.Delenie();
        v.Print();
```

```
        return 0;
}
```

23. Запустить программу на выполнение и протестировать ее работу.