Министерство науки и высшего образования Российской Федерации Федеральное государственное автономное образовательное учреждение высшего образования

«Пермский национальный исследовательский политехнический университет»

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы» направление подготовки: 09.03.04 – «Программная инженерия»

Лабораторная работа №11.

" Последовательные контейнеры библиотеки STL. "

15 вариант

Выполнил студент гр. РИС-24-2б

Иванова Елена Олеговна

Проверил:

Доц. Каф. ИТАС

Ольга Андреевна Полякова

(оценка) (подпись)

(дата)

г. Пермь, 2025

Цель:

1) Создание консольного приложения, состоящего из нескольких файлов в системе программирования Visual Studio.

2) Использование последовательных контейнеров библиотеки STL в ОО программе.

Постановка задачи:

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:



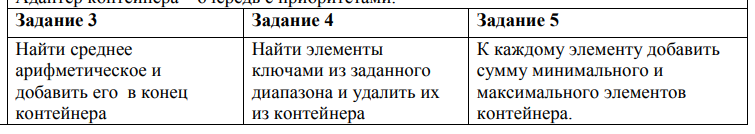
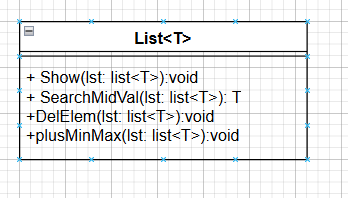
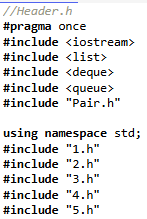
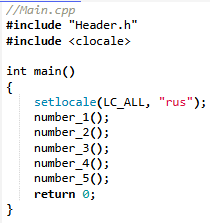


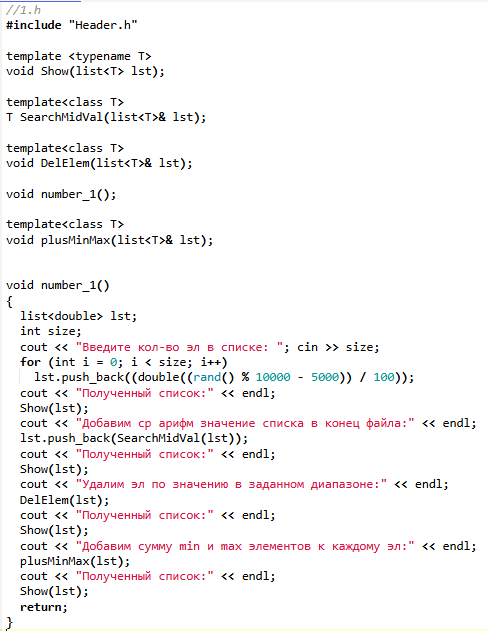
Диаграмма:

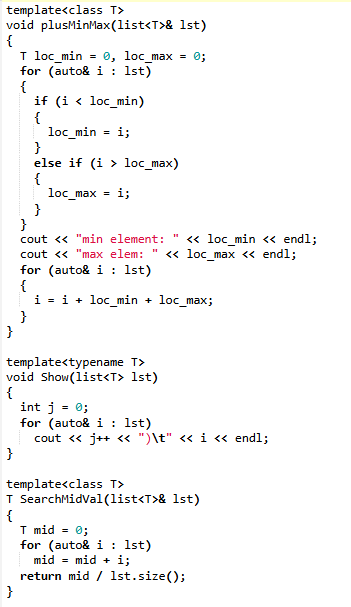


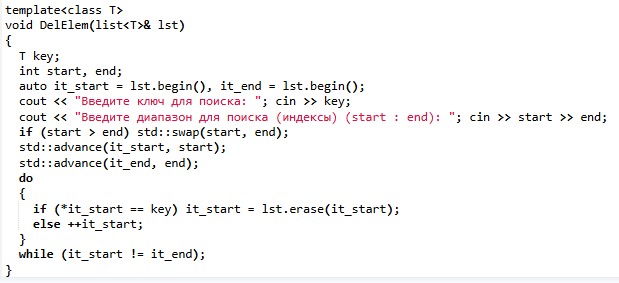
Код:



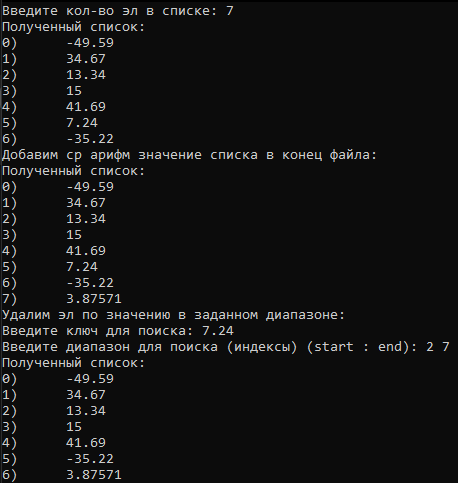


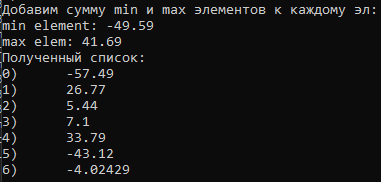






Вывод:





Постановка задачи:

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:



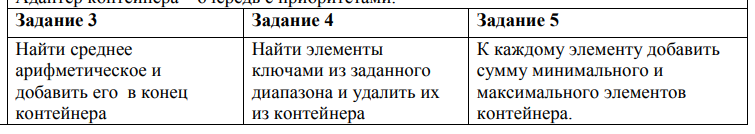
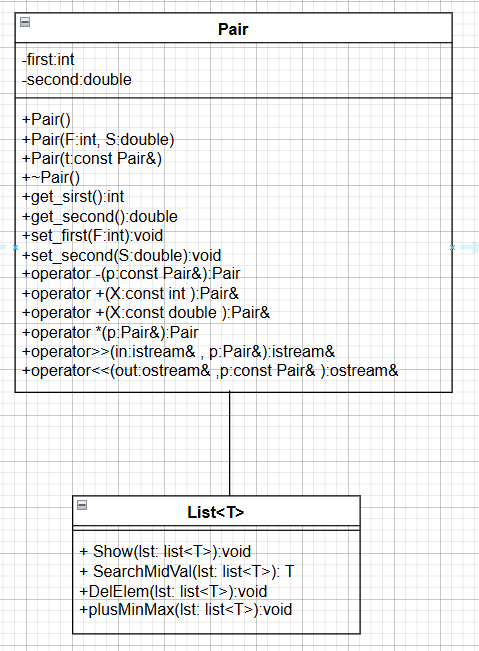
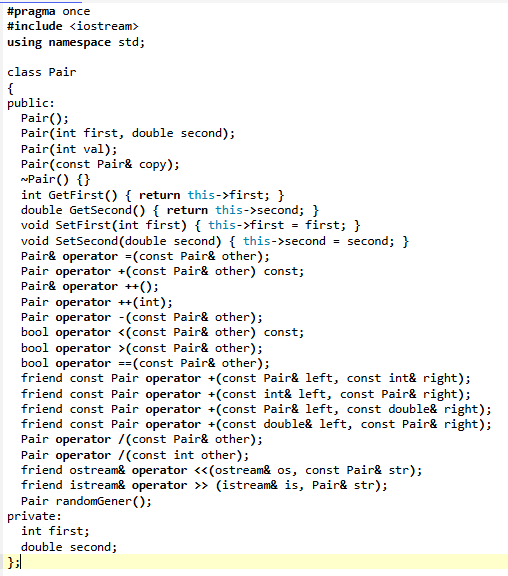
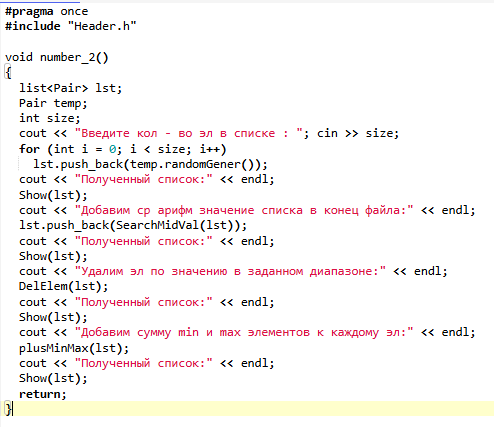


Диаграмма:

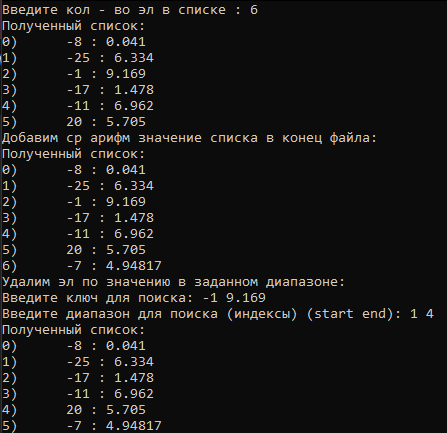


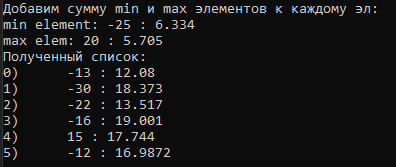
Код:





Вывод:





Постановка задачи:

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:



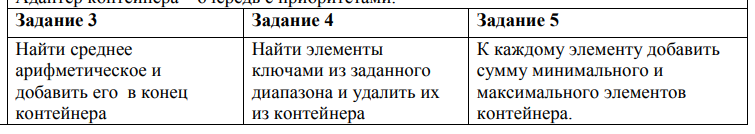
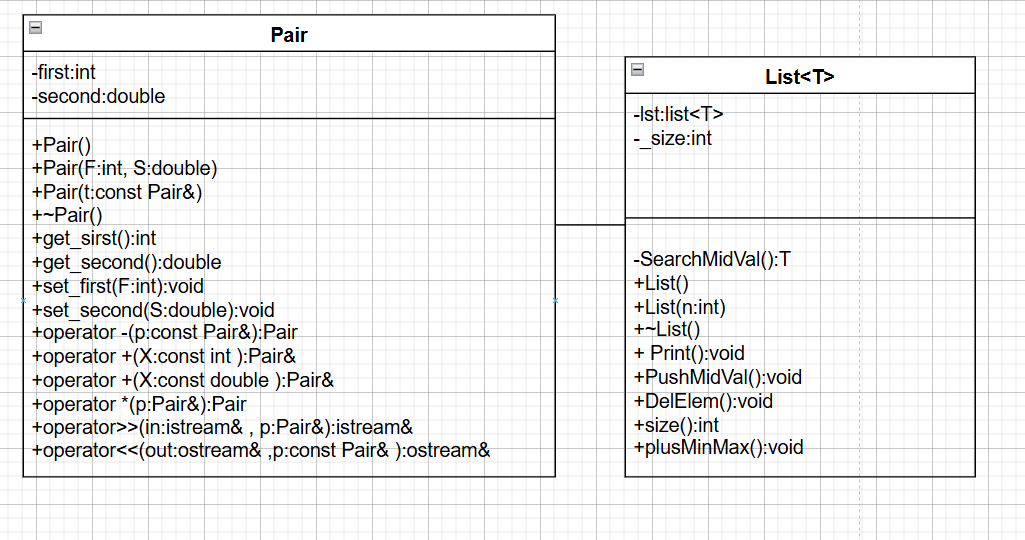
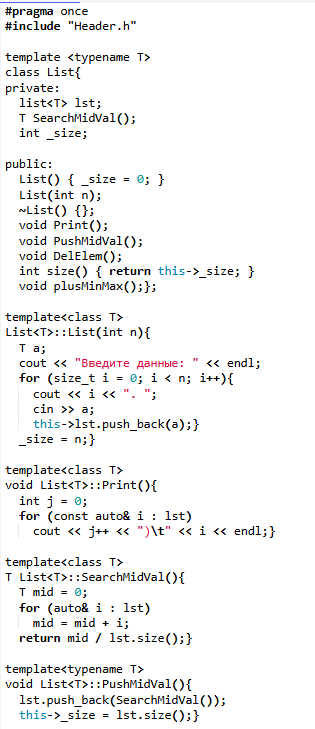
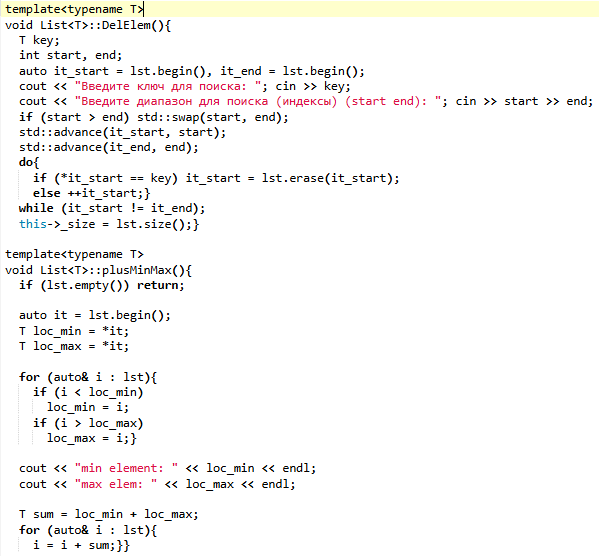


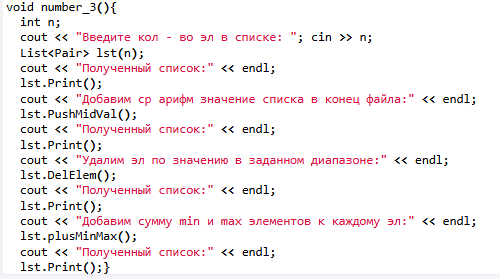
Диаграмма:



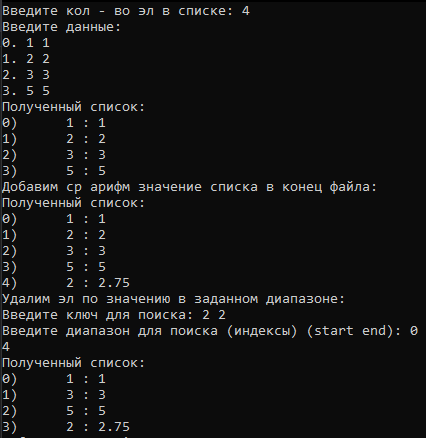
Код:

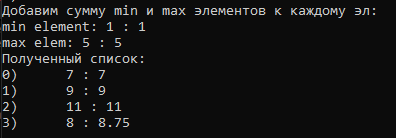






Вывод:





Постановка задачи:

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:



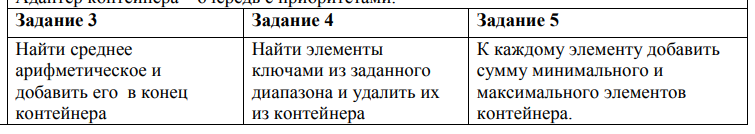
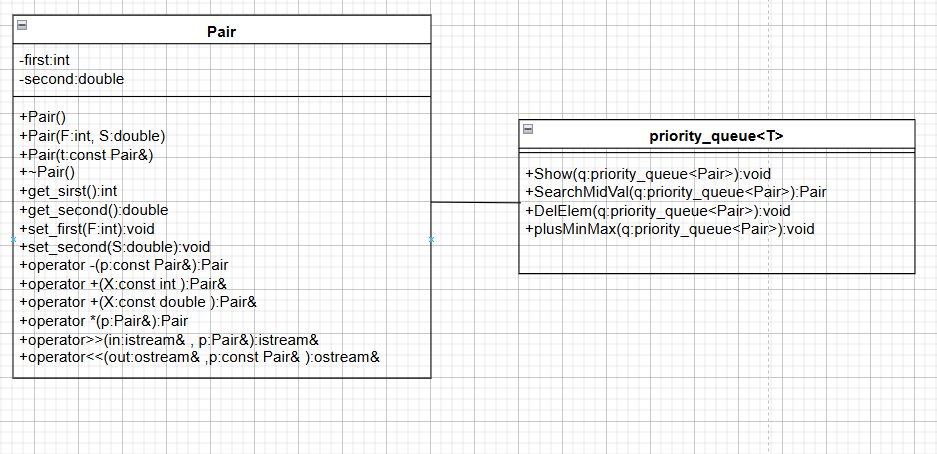
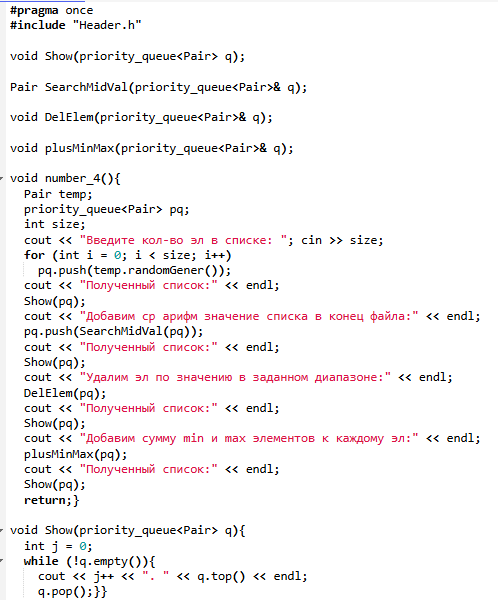
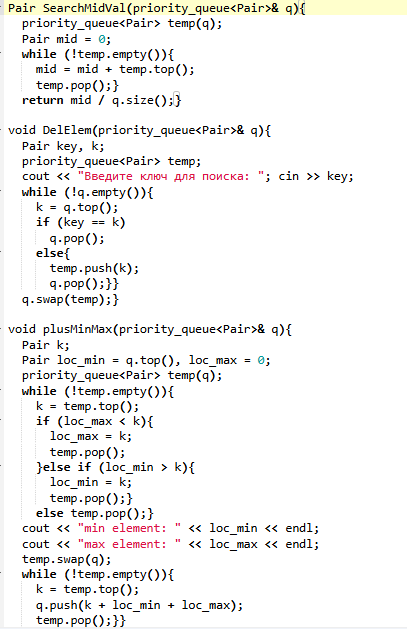


Диаграмма:

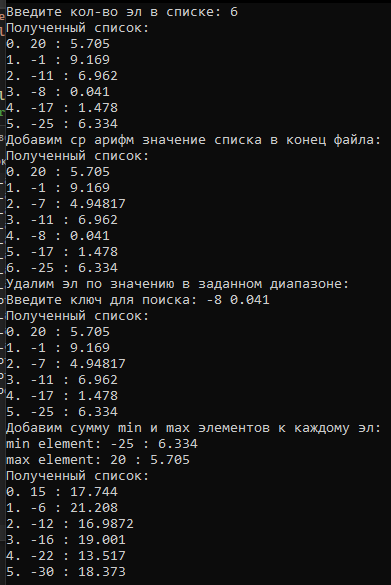


Код:





Вывод:



Постановка задачи:

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

Задача:



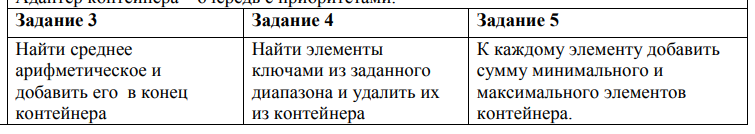
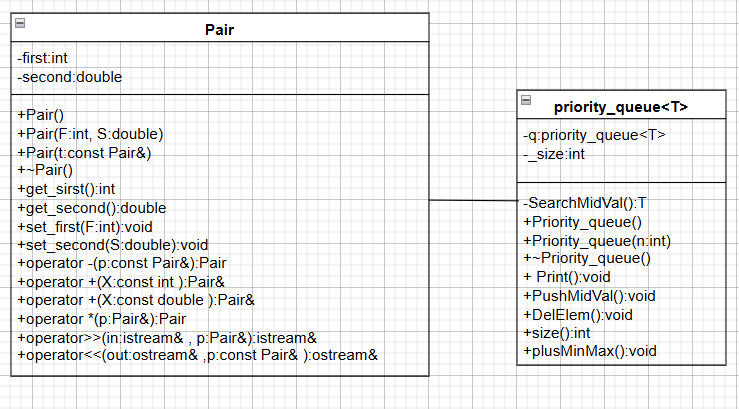
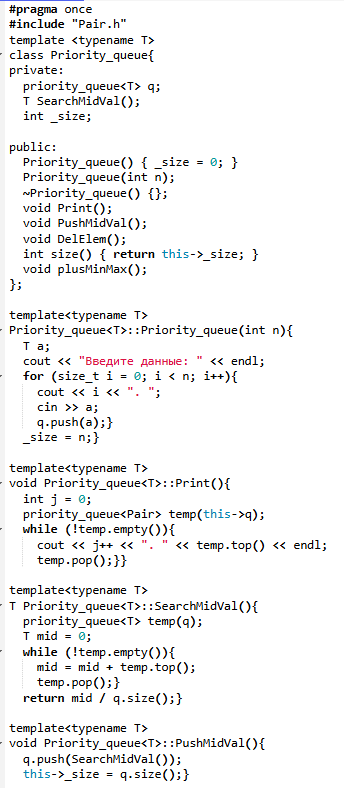
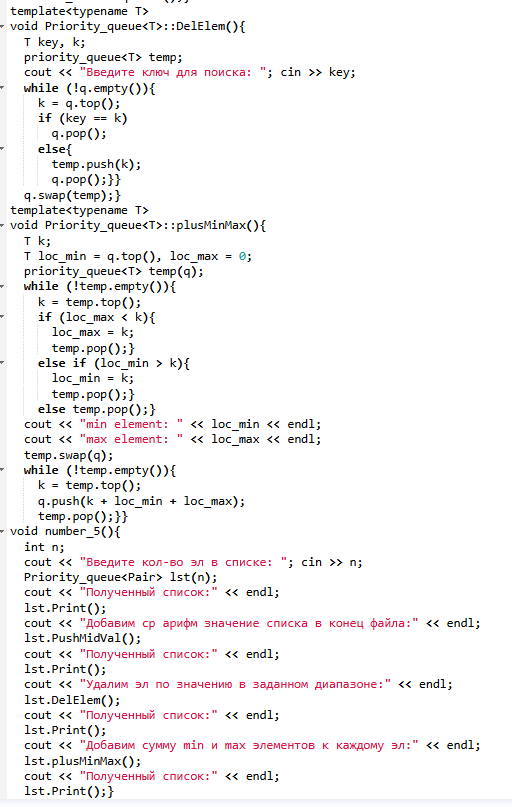


Диаграмма:

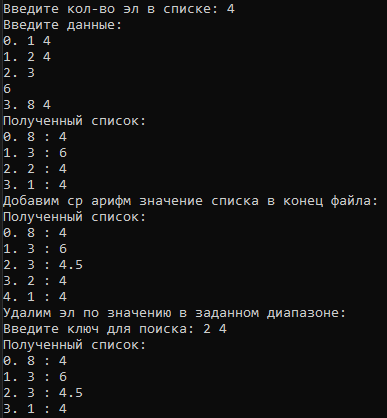


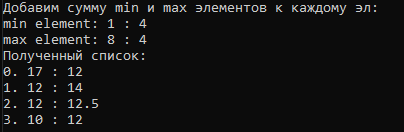
Код:



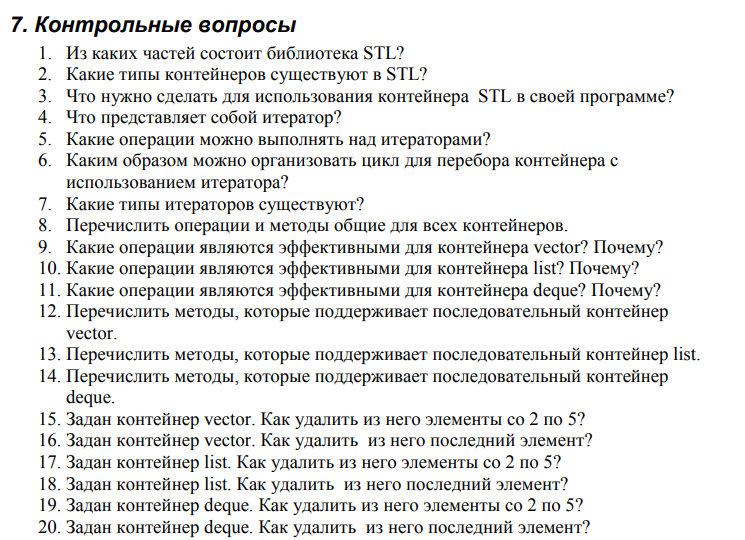


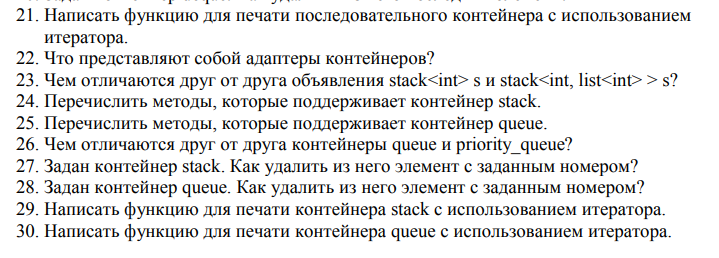
Вывод:





Контрольные вопросы:

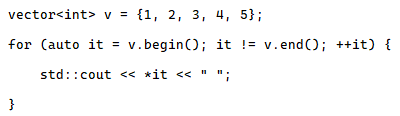




1. Библиотека STL (Standard Template Library) включает три части: контейнеры, алгоритмы и итераторы.

* Последовательные контейнеры: vector, deque, list, forward\_list, array
* Ассоциативные контейнеры: set, multiset, map, multimap
* Неупорядоченные ассоциативные контейнеры: unordered\_set, unordered\_multiset, unordered\_map, unordered\_multimap
* Адаптеры контейнеров: stack, queue, priority\_queue

1. Для использования контейнеров STL в своей программе необходимо подключить заголовочный файл, соответствующий нужному контейнеру. Например, для использования вектора (vector) нужно подключить заголовочный файл <vector> с помощью директивы #include <vector>. Также, необходимо использовать пространство имен std.
2. Итератор - это объект, который позволяет последовательно перебирать элементы контейнера STL (и не только STL). Он представляет собой абстракцию, которая инкапсулирует конкретную реализацию обхода элементов контейнера. Итераторы могут быть использованы для доступа к элементам контейнера, изменения их значений и удаления. Они могут также перемещаться по контейнеру вперед и назад. Кроме того, алгоритмы STL используют итераторы для работы с контейнерами, что позволяет создавать универсальные алгоритмы, работающие с любыми контейнерами STL.
3. Итераторы поддерживают несколько базовых операций:
   1. Операция разыменования (\*) - получение значения, на которое указывает итератор.
   2. Операция инкремента (++) - перемещение итератора на следующий элемент в контейнере.
   3. Операция декремента (--) - перемещение итератора на предыдущий элемент в контейнере (не поддерживается всеми итераторами).
   4. Операции сравнения (==, !=, <, >, <=, >=) - сравнение итераторов между собой.
   5. Операция присваивания (=) - присваивание одного итератора другому.
4. Для перебора контейнера с использованием итератора можно организовать цикл, используя два итератора - начальный и конечный. Начальный итератор указывает на первый элемент контейнера, а конечный итератор указывает на элемент, следующий за последним элементом контейнера. Таким образом, цикл перебирает все элементы контейнера, начиная с первого и заканчивая последним (не включая его).



1. В STL существует пять основных типов итераторов:
   1. Input iterator - итератор ввода, который позволяет перебирать элементы контейнера только в одном направлении и только для чтения.
   2. Output iterator - итератор вывода, который позволяет записывать значения в контейнер только в одном направлении.
   3. Forward iterator - итератор прямого доступа, который позволяет перебирать элементы контейнера только в одном направлении и изменять значения элементов.
   4. Bidirectional iterator - итератор двунаправленного доступа, который поддерживает все операции Forward iterator и добавляет возможность перемещения итератора назад.
   5. Random access iterator - итератор произвольного доступа, который поддерживает все операции Bidirectional iterator и добавляет возможность перемещения итератора на произвольное число шагов вперед и назад.
2. А
   1. begin() - метод, возвращающий итератор на первый элемент контейнера.
   2. end() - метод, возвращающий итератор на элемент, следующий за последним элементом контейнера.
   3. size() - метод, возвращающий количество элементов в контейнере.
   4. empty() - метод, возвращающий true, если контейнер пустой, и false в противном случае.
   5. clear() - метод, удаляющий все элементы из контейнера.
   6. insert() - метод, вставляющий новый элемент в контейнер.
   7. erase() - метод, удаляющий элемент из контейнера.
   8. find() - метод, возвращающий итератор на элемент, найденный по ключу или значению.
3. Контейнер vector реализован в виде динамического массива, поэтому для него эффективными операциями являются:
   1. Доступ к произвольному элементу с помощью оператора [] - O(1) по времени, так как элементы хранятся последовательно в памяти и доступ к любому элементу происходит за константное время.
   2. Вставка нового элемента в конец контейнера методом push\_back() - O(1) по амортизированному времени, так как вектор хранит элементы в памяти последовательно и может использовать выделенную под него память для хранения новых элементов. Если места для новых элементов недостаточно, то выделится новая область памяти и произойдет копирование всех элементов в нее, что займет O(N) времени, где N - текущее количество элементов в контейнере.
   3. Удаление последнего элемента из контейнера методом pop\_back() - O(1) по времени, так как это просто удаление элемента из конца массива.

Эти операции эффективны для контейнера vector, так как позволяют работать с ним за константное время.

1. Контейнер list реализован в виде двусвязного списка, поэтому для него эффективными операциями являются:
   1. Вставка нового элемента в произвольное место контейнера методом insert() - O(1) по времени, так как при вставке элемента в список не требуется перемещать другие элементы, достаточно просто обновить ссылки на следующий и предыдущий элементы.
   2. Удаление элемента из произвольного места контейнера методом erase() - O(1) по времени, так как при удалении элемента из списка не требуется перемещать другие элементы, достаточно просто обновить ссылки на следующий и предыдущий элементы.
   3. Обход элементов контейнера в прямом и обратном направлении с помощью итераторов - O(N) по времени, где N - количество элементов в списке. Обход списка в прямом направлении выполняется за линейное время, так как каждый элемент списка имеет ссылку на следующий элемент, и можно легко перейти от одного элемента к другому.

Эти операции эффективны для контейнера list, так как позволяют работать с ним за константное время

1. Контейнер deque (double-ended queue) представляет собой двустороннюю очередь, которая состоит из блоков фиксированного размера. Для него эффективными операциями являются:
   1. Вставка и удаление элементов с начала и конца очереди с помощью методов push\_front(), pop\_front(), push\_back() и pop\_back() - O(1) по времени, так как эти операции выполняются непосредственно над блоками, без необходимости переноса элементов из одного блока в другой.
   2. Доступ к произвольному элементу с помощью оператора [] и метода at() - O(1) по времени, так как элементы хранятся в блоках, и к каждому элементу можно получить доступ за константное время.
   3. Итерация с помощью итераторов - O(N) по времени, где N - количество элементов в deque. Итерация выполняется за линейное время, так как элементы хранятся последовательно в блоках.

Эти операции эффективны для контейнера deque, так как позволяют выполнять основные операции за константное время

1. Методы vector.

* push\_back() - добавляет элемент в конец вектора
* pop\_back() - удаляет элемент с конца вектора
* size() - возвращает количество элементов в векторе
* capacity() - возвращает количество элементов, которые могут быть сохранены в векторе без перевыделения памяти
* resize() - изменяет размер вектора, при необходимости добавляя новые элементы или удаляя существующие
* reserve() - изменяет емкость вектора, выделяя достаточно памяти для хранения указанного количества элементов
* clear() - удаляет все элементы из вектора
* empty() - проверяет, является ли вектор пустым
* operator[] - обращение к элементам вектора по индексу
* at() - обращение к элементам вектора по индексу с проверкой на выход за пределы
* front() - возвращает первый элемент вектора
* back() - возвращает последний элемент вектора
* data() - возвращает указатель на начало вектора для чтения и записи
* begin() - возвращает итератор, указывающий на первый элемент вектора
* end() - возвращает итератор, указывающий на элемент, следующий за последним элементом вектора.

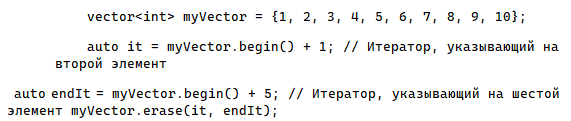
1. Методы list.

* push\_front() - добавляет элемент в начало списка
* push\_back() - добавляет элемент в конец списка
* pop\_front() - удаляет первый элемент списка
* pop\_back() - удаляет последний элемент списка
* size() - возвращает количество элементов в списке
* empty() - проверяет, является ли список пустым
* clear() - удаляет все элементы из списка
* front() - возвращает первый элемент списка
* back() - возвращает последний элемент списка
* insert() - добавляет элементы в список по указанной позиции
* erase() - удаляет элементы из списка по указанной позиции или диапазону
* remove() - удаляет все элементы списка, равные заданному значению
* reverse() - изменяет порядок элементов в списке на обратный
* sort() - сортирует элементы списка по возрастанию (по умолчанию) или по заданному компаратору
* merge() - объединяет два отсортированных списка в один отсортированный список

1. Методы deque.

* push\_front() - добавляет элемент в начало очереди
* push\_back() - добавляет элемент в конец очереди
* pop\_front() - удаляет первый элемент очереди
* pop\_back() - удаляет последний элемент очереди
* size() - возвращает количество элементов в очереди
* empty() - проверяет, является ли очередь пустой
* clear() - удаляет все элементы из очереди
* front() - возвращает первый элемент очереди
* back() - возвращает последний элемент очереди
* at() - возвращает элемент очереди по заданному индексу
* insert() - добавляет элементы в очередь по указанной позиции
* erase() - удаляет элементы из очереди по указанной позиции или диапазону
* resize() - изменяет размер очереди
* swap() - обменивает содержимое двух очередей

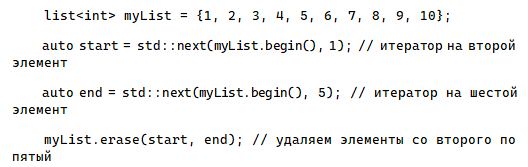
1. .



1. .



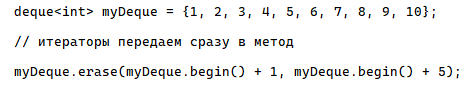
1. .



1. .



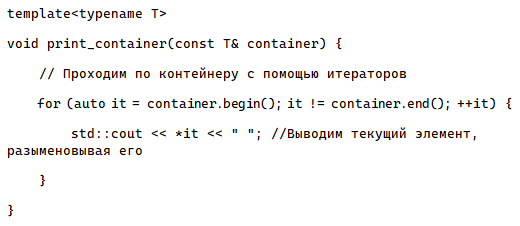
1. .



1. .



1. .



1. Адаптеры контейнеров - это шаблонные классы, которые позволяют изменять поведение существующих контейнеров STL без изменения их основной структуры. Они обычно представляют собой обертки над существующими контейнерами, добавляющие новые функциональные возможности или изменяющие существующие.
2. Объявление stack<int> s создает стек, который использует vector<int> в качестве контейнера-адаптера, тогда как объявление stack<int, list<int>> s создает стек, который использует list<int> в качестве контейнера-адаптера. Таким образом, различие между ними заключается в типе контейнера, используемого для хранения элементов стека.
3. Методы контейнера stack.

* Push(): добавляет элемент на вершину стека.
* Pop(): удаляет элемент с вершины стека.
* Top(): возвращает элемент, находящийся на вершине стека.
* Empty(): возвращает true, если стек пуст, и false в противном случае.
* Size(): возвращает количество элементов в стеке.

1. Перечислите методы, которые поддерживает контейнер queue.

* Push(): добавляет элемент в конец очереди.
* Pop(): удаляет элемент из начала очереди.
* Front(): возвращает элемент, находящийся в начале очереди.
* Back(): возвращает элемент, находящийся в конце очереди.
* Empty(): возвращает true, если очередь пуста, и false в противном случае.
* Size(): возвращает количество элементов в очереди.

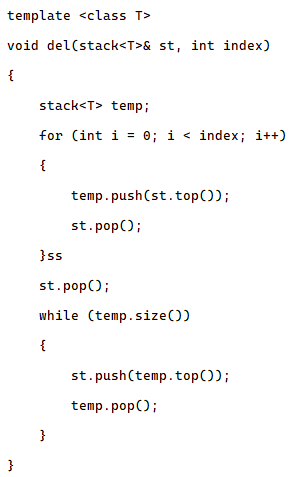
1. Контейнеры queue и priority\_queue относятся к адаптерам контейнеров и представляют собой структуры данных для хранения элементов в определенном порядке. Основное отличие между ними заключается в том, как элементы извлекаются из контейнера:

queue - используется для хранения элементов в порядке, в котором они были добавлены, и извлекает элементы по принципу "первым пришел - первым вышел".

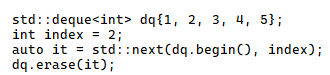
priority\_queue - используется для хранения элементов в отсортированном порядке, и извлекает элементы с наибольшим приоритетом (в соответствии с заданной функцией сравнения).

Таким образом, queue поддерживает методы push(), pop() и front(), а priority\_queue - методы push(), pop() и top().

1. .



1. .



1. Это невозможно, так как у контейнера stack даже итераторов нет.

Вот пример как можно посмотреть все, что лежит в stack, но так делать тоже не надо, потому что это ломает весь смысл работы с данным контейнером, гораздо логичнее будет взять любой другой контейнер, например отлично подойдет однонаправленный список.

