Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«**Пермский национальный исследовательский политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**ОТЧЕТ**

Дисциплина: «Основы алгоритмизации и программирования»

Тема: «Последовательные контейнеры библиотеки STL»

Семестр 2

Выполнил работу:

Студент группы ИВТ-22-2б

Цыбуцынин Фёдор Александрович

Проверил:

Доцент кафедры ИТАС

Полякова Ольга Андреевна

Г. Пермь-2023

**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 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 параметризованный 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.

**Описание класса**

1. Найти минимальный элемент и добавить его в конец контейнера;

2. Найти элементы с заданным ключом и удалить их из контейнера;

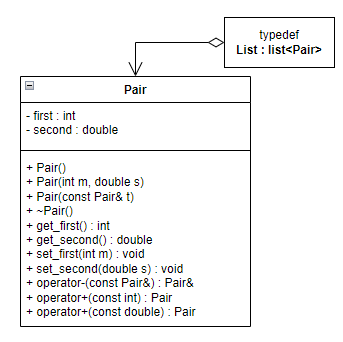
3. К каждому элементу добавить сумму минимального и максимального элементов контейнера.

**UML - Диаграмма**

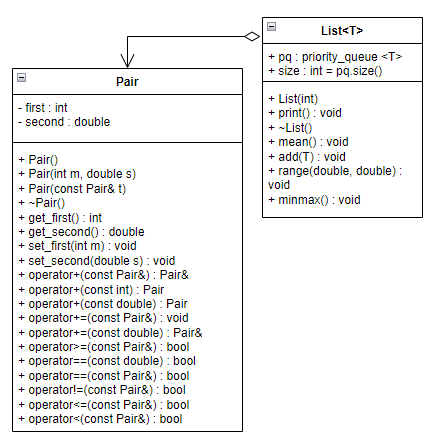
**Задача 1**



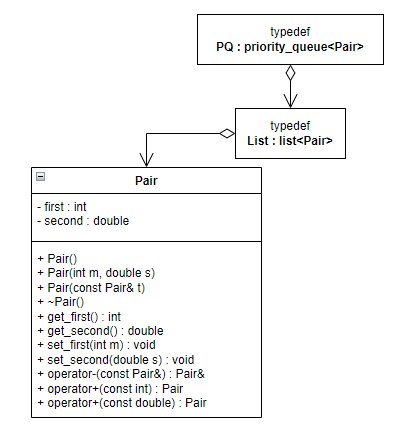
**Задача 2**



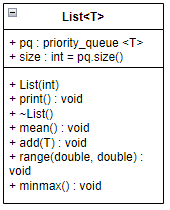
**Задача 3**



**Задача 4**



**Задача 5**



**Контрольные вопросы**

1. Из каких частей состоит библиотека STL?

STL – Standard Template Library, стандартная библиотека шаблонов

состоит из двух основных частей: набора контейнерных классов и набора обобщенных алгоритмов. Контейнеры — это объекты, содержащие другие однотипные объекты. Обобщенные алгоритмы реализуют большое количество процедур, применимых к контейнерам: поиск, сортировку, слияние и т. п.

1. Какие типы контейнеров существуют в STL?

Последовательные контейнеры (векторы (vector), списки (list) и двусторонние очереди (deque)), ассоциативные контейнеры (словари (mар), словари с дубликатами (multimap), множества (set), множества с дубликатами (multiset) и битовые множества (bitset)), есть еще специализированные контейнеры (или адаптеры контейнеров), реализованные на основе базовых — стеки (stack), очереди (queue) и очереди с приоритетами (priority\_queue).

1. Что нужно сделать для использования контейнера STL в своей программе?

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>

1. Что представляет собой итератор?

Итератор — интерфейс, предоставляющий доступ к элементам коллекции и навигацию по ним.

1. Какие операции можно выполнять над итераторами?

- Разыменование итератора: если р — итератор, то \*р — 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 (++р) или постфиксного (р++) инкремента.

1. Каким образом можно организовать цикл для перебора контейнера с использованием итератора?

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 (iterator it = v.begin(); it != v.end(), ++it)

1. Какие типы итераторов существуют?

В STL существуют следующие типы итераторов:

- входные,

- выходные,

- прямые,

- двунаправленные итераторы,

- итераторы произвольного доступа.

1. Перечислить 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 empty0 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 |

1. Какие операции являются эффективными для контейнера vector? Почему?

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. Происходит это потому, что массив - это последовательно занятая память, так что доступ к любому элементу происходит быстро.

1. Какие операции являются эффективными для контейнера list? Почему?

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, поскольку не требуется заново перевыделять память, достаточно переобозначить связи с предыдущим и последующим элементом.

1. Какие операции являются эффективными для контейнера deque? Почему?

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). Дек является сочетанием вектора и списка, поэтому при добавлении элементов в начало создается новый массив, конец которого по принципа списка указывает на изначальный массив. Аналогично с добавлением элементов в конец.

1. Перечислить методы, которые поддерживает последовательный контейнер vector.

push\_back(), pop\_back(), insert, erase, [], at, swap, clear()

1. Перечислить методы, которые поддерживает последовательный контейнер list.

push\_back(), pop\_back(), push\_front(), pop\_back(), insert(), erase, swap, clear()

1. Перечислить методы, которые поддерживает последовательный контейнер deque.

push\_back(), push\_front(), pop\_back, pop\_front, insert, erase, [] ,at

1. Задан контейнер vector. Как удалить из него элементы со 2 по 5?

for (int i = 2; i <= 5; ++i) v.erase(v.begin() + i);

1. Задан контейнер vector. Как удалить из него последний элемент?

vect.pop\_back()

1. Задан контейнер list. Как удалить из него элементы со 2 по 5?

list<int>::iterator beg = l.begin();

advance(beg, 2);

list<int>::iterator end = l.begin();

advance(end, 6);

l.erase(beg, end);

1. Задан контейнер list. Как удалить из него последний элемент?

list.pop\_back()

1. Задан контейнер deque. Как удалить из него элементы со 2 по 5?

auto beg = q.begin();

advance(beg, 2);

auto end = q.begin();

advance(end, 6);

q.erase(beg, end);

1. Задан контейнер deque. Как удалить из него последний элемент?

deque.pop\_back()

1. Написать функцию для печати последовательного контейнера с использованием итератора.

void print(deque<int>q)

{

for (deque<int>::iterator elem = q.begin();

{

elem != q.end(); ++elem)

cout << \*elem << " ";

}

}

1. Что представляют собой адаптеры контейнеров?

Специализированные последовательные контейнеры — стек, очередь и очередь с приоритетами — не являются самостоятельными контейнерными классами, а реализованы на основе последовательных контейнеров, поэтому они называются адаптерами контейнеров.

1. Чем отличаются друг от друга объявления stack<int> s и stack<int, list<int> > s?

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;

1. Перечислить методы, которые поддерживает контейнер stack.

- push () - добавление в конец;

- pop () - удаление из конца;

- top () - получение текущего элемента стека;

- empty() - проверка пустой стек или нет;

- size () – получение размера стека.

1. Перечислить методы, которые поддерживает контейнер queue.

- push () - добавление в конец;

- pop () - удаление из конца;

- front () - получение первого элемента очереди;

- back() - получение последнего элемента очереди

- empty() - проверка пустой стек или нет;

- size () – получение размера стека.

1. Чем отличаются друг от друга контейнеры queue и priority\_queue?

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.

1. Задан контейнер stack. Как удалить из него элемент с заданным номером?

stack<int> copy;

int counter = 0;

while (counter != 2)

{

copy.push(s.top());

s.pop();

++counter;

}

while (counter != 6)

{

s.pop();

++counter;

}

while (!copy.empty())

{

s.push(copy.top());

copy.pop();

}

1. Задан контейнер queue. Как удалить из него элемент с заданным номером?

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

{

push(qwe, qwe->head->key);

pop(qwe);

}

pop(qwe);

for (int i = 0; i < (qwe->size) - (num + 2 \* k); i++)

{

push(qwe, qwe->head->key);

pop(qwe);

}

1. Написать функцию для печати контейнера stack с использованием итератора.

void print(stack<int> s)

{

int size = s.size();

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

{

cout << s.top() << " ";

s.pop();

}

}

1. Написать функцию для печати контейнера queue с использованием итератора.

void print(queue<int> s)

{

int size = s.size();

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

{

cout << s.front() << " ";

s.pop();

}

}