Министерство высшего образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

ПЕРМСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ (ПНИПУ)

Электротехнический факультет

Дисциплина: основы алгоритмизации и программирования, 2 семестр

ОТЧЁТ

Тема: «Лабораторная работа №11»

Выполнил

Студент РИС-24-1б

Конькова С. С.

Проверил

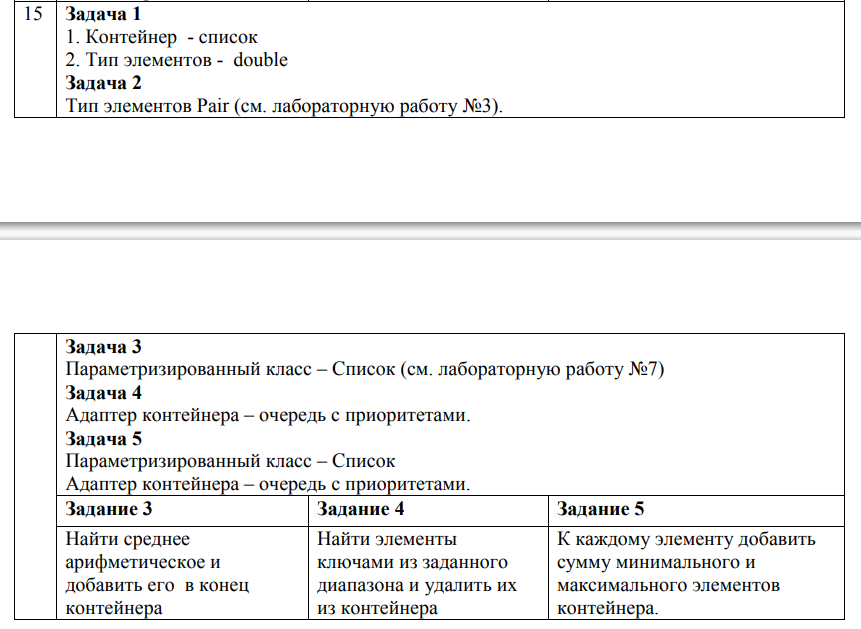
Доцент кафедры ИТАС

Полякова О. А.

Пермь 2025

Изображение выглядит как текст, снимок экрана, Шрифт, число

Контент, сгенерированный ИИ, может содержать ошибки.



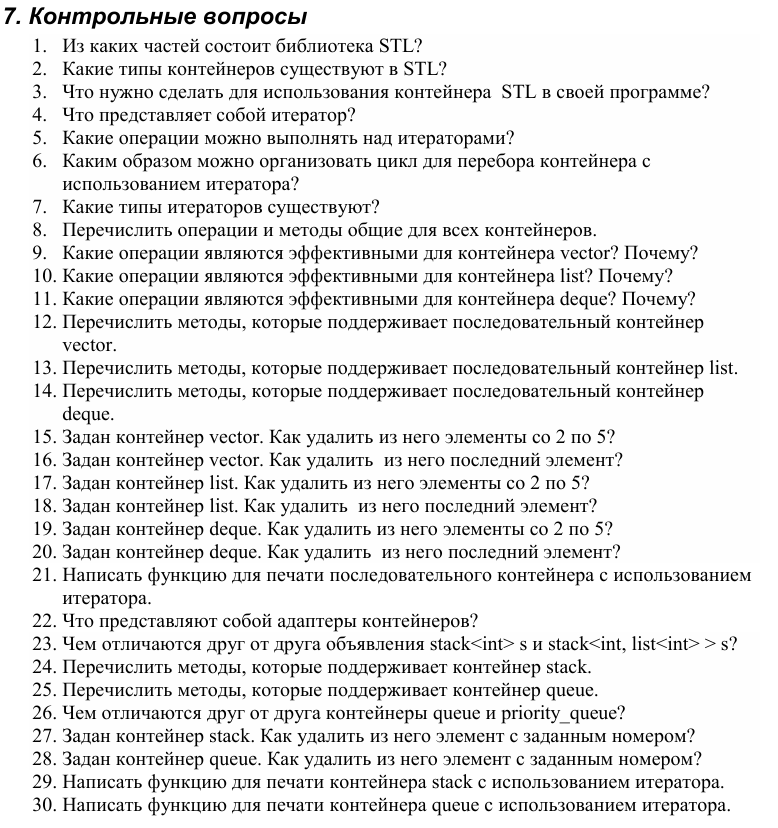
Анализ

* ex\_1() — работа со стандартным списком list<double>:
  1. Создаёт список из чисел от 1 до 8.
  2. Выводит список.
  3. Находит среднее арифметическое всех элементов, добавляет его в конец списка.
  4. Запрашивает у пользователя диапазон и значение "ключ", удаляет элементы из указанного диапазона, если они равны ключу.
  5. К каждому элементу добавляет сумму максимального и минимального значений.
* ex\_2() — работа со списком list<Pair>:
  1. Добавляет три объекта Pair (1,1.5), (-2,2.7), (3,-3.2).
  2. Выводит список.
  3. Вычисляет среднее арифметическое Pair, добавляет его в конец.
  4. Запрашивает диапазон и ключ Pair, удаляет по аналогии как в ex\_1.
  5. К каждому элементу прибавляет сумму максимального и минимального элемента.
* ex\_3() — работа с LinkedList<Pair> (самописный класс):
  1. То же самое, что в ex\_2, только использует свой собственный класс списка на указателях.
  2. Список выводится через рекурсивную функцию show().
* ex\_4() — работа с priority\_queue<Pair>:
  1. Добавляет несколько Pair.
  2. Выводит все элементы очереди.
  3. Находит среднее арифметическое и добавляет в очередь.
  4. Удаляет элементы по диапазону и ключу.
  5. Добавляет сумму минимального и максимального Pair ко всем элементам.
* ex\_5() — это повторный вызов ex\_3()

Результат работы

Изображение выглядит как текст, снимок экрана, документ, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.



1. набора контейнерных классов и набора обобщенных алгоритмов.

2. последовательные, ассоциативные, контейнеры-адаптеры и псевдоконтейнеры.

3. необходимо включить в нее соответствующий заголовочный файл. Тип объектов, сохраняемых в контейнере, задается с помощью аргумента шаблона

4. это обобщение концепции указателей: они ссылаются на элементы контейнера.

5. Их можно инкрементировать (++), как обычные указатели, для последовательного продвижения по контейнеру, а также разыменовывать для получения или изменения значения элемента (\*).

6. for (i = first; i != last; ++i),

7.

• входные

• выходные

• прямые

• двунаправленные итераторы

• итераторы произвольного доступа.

8. size\_type, iterator, const\_iterator, reference, const\_reference, key\_type, key compare

9. индексации []

10. не поддерживает произвольного доступа

11. эффективно поддерживает вставку и удаление первого элемента (так же, как и последнего)

12. push\_back(), pop\_back(), insert, erase.

13. push\_back(T&key), pop\_back(), push\_front(T&key), pop\_front(), insert, erase, swap

14. push\_back(T&key), pop\_back(), push\_front(T&key), pop\_front(), insert.

15. erase(begin, end): удаляет элементы из диапазона, на начало и конец которого указывают итераторы begin и end. Возвращает итератор на элемент, следующий после последнего удаленного, или на конец контейнера, если удален последний элемент

16. Функцией pop\_back()

17. Erase(2,5)

18. pop\_back()

19.Erase(2,5)

20. pop\_back()

21.

For(auti i : st){

Cout<<i<<endl;}

22. Они используют какой-нибудь другой контейнер (например, дек или вектор) для хранения своих элементов, но при этом предоставляют свой набор функций для работы с ними

23. Что 1 случай — это объявление stack. А 2 — это объявление стака внутри которого есть list

24. Empty, pop, push, size, top

25.Перечислить методы, которые поддерживает контейнер queue.

Back, empty, front, pop, push, size

26. priority\_queue всегда является самым большим или имеет высший приоритет

27. Pop\_Back ()

28. Erase (value, pos)

29.

while (stack != empty) {

cout << stack.top();

i++;

}

30.

while (queue != empty) {

cout << queue.front();

i++;

}