B. Задания к лабораторным работам

Алексей Мартынов

7 сентября 2015 г.

Версия 1.0

Содержание

1 Векторы

Необходимо выполнить *все* задания.

1. Напишите алгоритм сортировки (любой простейший) коллекции целых чисел так, чтобы:
   (a) сортировка вектора проводилась с использованием оператора **operator** [];
   (b) сортировка вектора проводилась с использованием метода vector :: at().
   (c) сортировка односвязного списка осуществлялась при помощи итераторов (см. **??**).
2. Прочитайте во встроенный массив C содержимое текстового файла, скопируйте данные в вектор одной строкой кода (без циклов и алгоритмов STL).
3. Напишите программу, сохраняющую в векторе числа, полученные из стандартного ввода (окончанием ввода является число 0). Удалите все элементы, которые делятся на 2 (не используя стандартные алгоритмы STL), если последнее число 1. Если последнее число 2, добавьте после каждого числа, которое делится на 3, три единицы.
4. Напишите функцию **void** fillRandom(**double** * array, **int** size), заполняющую массив случайными значениями в интервале от -1.0 до $+1.0$. Заполните с помощью заданной функции вектора размером 5, 10, 25, 50, 100 и отсортируйте его содержимое (с помощью любого разработанного ранее алгоритма, модифицированного для сортировки как целых, так и действительных чисел).

2 Строки

Разработать программу, которая должна сделать следующее:

1. Прочитать содержимое текстового файла. Файл может содержать:
   (a) Слова — состоят из латинских строчных и заглавных букв, а также цифр, длина слова должна быть не более 20 символов.
   (b) Знаки препинания — «.», «,», «!», «?», «:», «;».
   (c) Пробельные символы — пробел, табуляция, символ новой строки.
2. Отформатировать текст следующим образом:
   (a) Не должно быть пробельных символов отличных от пробела.
   (b) Не должно идти подряд более одного пробела.
   (c) Между словом и знаком препинания не должно быть пробела.
   (d) После знака препинания всегда должен идти пробел.
   (e) Слова длиной более 10 символов заменяются на слово «Vau!!!».
3. Преобразовать полученный текст в набор строк, каждая из которых содержит целое количество слов (слово должно целиком находиться в строке) и ее длина не превышает 40 символов.

Подсказки:

- Можно создать строки, содержащие символы принадлежащие какой-либо категории, например знаки препинания.
- Для хранения результирующих строк можно использовать std :: vector< std:: string >.

3 Последовательности

Необходимо выполнить *все* задания.

1

Ниже приведен интерфейс класса очереди с приоритетами, который функционирует следующим образом:

В очередь могут быть добавлены элементы, каждому элементу при добавлении присваивается один из трех уровней приоритета (low, normal, high).

Элементы из очереди извлекаются в соответствии с их приоритетами (сначала извлекаются элементы с приоритетом high, потом normal, потом low), элементы с одинаковыми приоритетами извлекаются из очереди в порядки их поступления.

В очереди также может происходить операция акселерации — все элементы с приоритетом low находящиеся в момент акселерации в очереди увеличивают свой приоритет до high и «обгоняют» элементы с приоритетом normal.

Ниже приведен интерфейс этого класса:

```
 1  typedef enum
 2  {
 3    LOW,
 4    NORMAL,
 5    HIGH
 6  } ElementPriority;
 7
 8  typedef struct
 9  {
10    std::string name;
11  } QueueElement;
12
13  class QueueWithPriority
14  {
15    QueueWithPriority();
16
17    ~QueueWithPriority();
18
19    void PutElementToQueue(const QueueElement & element,
20        ElementPriority priority);
21
22    QueueElement GetElementFromQueue();
23
24    void Accelerate();
25  };
```

1. Исправьте ошибки в интерфейсе, обеспечив безопасный интерфейс для применения в большом промышленном проекте.
2. Реализуйте исправленный класс, используя std :: list или std :: deque. Объясните выбор.
3. Напишите набор тестов, демонстрирующих правильную работу.

2

Разработайте программу, которая:

1. Заполняет std :: list < **int** > 15 случайными значениями от 1 до 20, список может содержать от 0 до 20 значений (обязательно проверить на длине списка 0, 1, 2, 3, 4, 5, 7, 14).
2. Выводит содержимое списка в следующем порядке: первый элемент, последний элемент, второй элемент, предпоследний элемент, тритий элемент и т.д.

Например если список содержит:

```
1 2 3 4 5 6 7 8
```

то вывод будет иметь вид

```
1 8 2 7 3 6 4 5
```

Подсказка: можно использовать рекурсию и двунаправленные итераторы.

4 Итераторы

Выполните *все* задания:

1

Напишите программу-«телефонную книжку».

Записи (имя и телефон) должны хранится в каком-либо STL-контейнере (std :: vector или std :: list), причем крайне желательно, чтобы от типа контейнера не зависело ничего, кроме одной строки в программе — объявления этого контейнера (указание: используйте **typedef**).

Программа должна поддерживать следующие операции:

- Просмотр текущей записи.
- Переход к следующей записи.
- Переход к предыдущей записи.
- Вставка записи перед/после просматриваемой.
- Замена просматриваемой записи.
- Вставка записи в конец базы данных.
- Переход вперед/назад через n записей.

Помните, что после вставки элемента итераторы становятся недействительными, поэтому после вставки целесообразно переставлять итератор на начало базы данных.

Постарайтесь реализовать операции вставки и замены с помощью одной и той же функции, которой в зависимости от требуемого действия передается либо обычный итератор, либо адаптер — один из итераторов вставки: **void** modifyRecord(iterator pCurrentRecord, CRecord newRecord).

Необходимо учесть, что клиентскому коду может быть необходимо иметь ссылки на разные записи в книжке одновременно.

2

Реализуйте следующие классы:

- Контейнер, который содержит значения факториала от 1! до 10!.

  Интерфейс класса должен включать в себя как минимум:

  - Конструктор по умолчанию.
  - Функцию получения итератора указывающего на первый элемент контейнера — begin().
  - Функцию получения итератора указывающего на элемент, следующий за последним — end().

  Доступ к элементам этого контейнера возможен только с помощью итераторов, возвращаемых функциями begin() и end().

  Контейнер не должен содержать в памяти свои элементы, они должны вычисляться при обращении к ним через итератор.

- Класс итератора для перечисления элементов этого контейнера, объекты этого класса возвращаются функциями begin() и end(). Итератор должен быть двунаправленным. Итератор должен быть совместимым с STL (проверить это можно используя алгоритм std :: copy() для копирования содержимого разработанного контейнера в std :: vector< **int** >).

5 Алгоритмы I

Написать программу, которая выполняет следующие действия:

1. Заполняет std :: vector< DataStruct > структурами DataStruct, при этом key1 и key2 генерируются случайным образом в диапазоне от −5 до +5, str заполняется из таблицы (таблица содержит 10 произвольных строк, индекс строки генерируется случайным образом).
2. Выводит полученный вектор на печать.

3. Сортирует вектор следующим образом:
   (a) По возрастанию key1.
   (b) Если key1 одинаковые, то по возрастанию key2.
   (c) Если key1 и key2 одинаковые, то по возрастанию длины строки str.
4. Выводит полученный вектор на печать

DataStruct определена следующим образом:

```
1 typedef struct
2 {
3   int         key1;
4   int         key2;
5   std::string str;
6 } DataStruct;
```

6 Алгоритмы II

1

Написать программу, которая выполняет следующие действия:

1. Читает содержимое текстового файла.
2. Выделяет слова, словом считается последовательность символов, разделенных пробелами и/или знаками табуляции и/или символами новой строки.
3. Вывести список слов, присутствующий в тексте без повторений (имеется в виду, что одно и то же слово может присутствовать в списке только один раз).

2

Написать программу, которая выполняет следующие действия (все операции должны выполняться с помощью стандартных алгоритмов):

1. Заполняет вектор геометрическими фигурами. Геометрическая фигура может быть треугольником, квадратом, прямоугольником или пятиугольником. Структура описывающая геометрическую фигуру определена ниже.
2. Подсчитывает общее количество вершин всех фигур, содержащихся в векторе (так треугольник добавляет к общему числу 3, квадрат 4 и т.д.).
3. Подсчитывает количество треугольников, квадратов и прямоугольников в векторе.
4. Удаляет все пятиугольники.
5. На основании этого вектора создает std::vector< Point >, который содержит координаты одной из вершин (любой) каждой фигуры, т.е. первый элемент этого вектора содержит координаты одной из вершин первой фигуры, второй элемент этого вектора содержит координаты одной из вершин второй фигуры и т.д.
6. Изменяет вектор так, чтобы он содержал в начале все треугольники, потом все квадраты, а потом прямоугольники.
7. Распечатывает вектор после каждого этапа работы.

Геометрическая фигура задается следующей структурой:

```
1 typedef struct
2 {
3   int                    vertex_num;
4   std::vector< Point > vertexes;
5 } Shape;
6
```

```
 7 typedef struct
 8 {
 9   int x,y;
10 } Point;
```

Подсказка: кроме алгоритмов рассмотренных в этой работе можно применять все средства описанные в предыдущих работах, включая алгоритмы сортировки.

7 Функторы I

Разработать функтор, позволяющий собирать статистику о последовательности целых чисел (последовательность может содержать числа от -500 до 500). Функтор после обработки последовательности алгоритмом std :: for_each должен предоставлять следующую статистику:

1. Максимальное число в последовательности.
2. Минимальное число в последовательности.
3. Среднее чисел в последовательности.
4. Количество положительных чисел.
5. Количество отрицательных чисел.
6. Сумму нечетных элементов последовательности.
7. Сумму четных элементов последовательности.
8. Совпадают ли первый и последний элементы последовательности.

Проверить работу программы на случайно сгенерированных последовательностях.

8 Функторы II

1

Разработать программу, которая, используя только стандартные алгоритмы и функторы, умножает каждый элемент списка чисел с плавающей точкой на число π.

2

Разработать программу, которая:

1. Реализует иерархию геометрических фигур состоящую из:
   (a) Класс Shape, содержащий:
   - информацию о положении центра фигуры (координаты x и y);
   - метод IsMoreLeft(), позволяющий определить расположена ли данная фигура левее (определяется по положению центра) чем фигура, переданная в качестве аргумента;
   - метод IsUpper, позволяющий определить расположена ли данная фигура выше (определяется по положению центра) чем фигура, переданная в качестве аргумента;
   - чисто виртуальную функцию рисования Draw() (каждая фигура в реализации этой функции должна выводить на стандартный вывод свое название и положение центра).

   (b) Класс Circle, производный от класса Shape.
   (c) Класс Triangle, производный от класса Shape.
   (d) Класс Square, производный от класса Shape.
2. Содержит список, заполненный указателями на различные фигуры.
3. С помощью стандартных алгоритмов и адаптеров выводит все фигуры.
4. С помощью стандартных алгоритмов и адаптеров сортирует список по положению центра слева-направо (имется в виду, что в начале списка должны идти фигуры находящиеся левее) и выводит фигуры.

5. С помощью стандартных алгоритмов и адаптеров сортирует список по положению центра справа-налево и выводит фигуры.

6. С помощью стандартных алгоритмов и адаптеров сортирует список по положению центра сверху-вниз и выводит фигуры.

7. С помощью стандартных алгоритмов и адаптеров сортирует список по положению центра снизу-вверх и выводит фигуры.