B. Задания к лабораторным работам 2 семестра

Алексей Мартынов

8 сентября 2020 г.

Версия 3.1

©2015—2020, доступно по лицензии Creative Commons “Attribution-NonCommercial-NoDerivs” 3.0

1 Векторы

Необходимо выполнить *все* задания.

Работа должна быть выполнена в виде 1 исполняемого файла, принимающего параметры следующим образом:

```
$ ./lab number [args]
```

number представляет собой номер пункта, например, 1 или 2. В качестве args передаются дополнительные параметры, зависящие от пункта.

1. Напишите алгоритм сортировки (любой простейший) коллекции целых чисел, полученных из стандартного ввода, так, чтобы:

   (a) сортировка вектора проводилась с использованием оператора **`operator []`**;

   (b) сортировка вектора проводилась с использованием метода `std::vector<>::at()`.

   (c) сортировка односвязного списка осуществлялась при помощи итераторов [Map, разд. 16.3.1].

   Окончанием ввода необходимо считать состояние End Of File (EOF, конец файла).

   Программа должна запускаться с дополнительным параметром, в зависимости от значения которого, сортировка производится по возрастанию или убыванию:

   ascending по возрастанию;
   descending по убыванию.

   Выведите на стандартный вывод отсортированные коллекции, разделяя элементы пробелами. Коллекция, отсортированная каждым видом доступа, должна быть выведена на отдельной строке. Например,

   ```
   $ cat data
   4 3 2 1
   $ ./lab 1 ascending < data
   1 2 3 4
   1 2 3 4
   1 2 3 4
   ```

2. Прочитайте во встроенный массив C содержимое текстового файла, имя которого передано в параметре. Скопируйте данные в вектор одной строкой кода (без циклов и алгоритмов Standard Template Library (STL)).

   Выведите на стандартный вывод содержимое вектора.

3. Напишите программу, сохраняющую в векторе целые числа, полученные из стандартного ввода (окончанием ввода является число 0). Используя итераторы, удалите все элементы, которые делятся на 2 (не используя алгоритмы STL), если последнее число 1. Если последнее число 2, добавьте после каждого числа, которое делится на 3, три единицы, также используя итераторы. Все изменения должны осуществляться при помощи итераторов и без использования индексов.

   Выведите на стандартный вывод полученную после преобразований коллекцию чисел, разделяя числа пробелом.

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 вектор заданного размера и отсортируйте его содержимое (с помощью любого разработанного ранее алгоритма, модифицированного для сортировки как целых, так и действительных чисел).

   Выведите на стандартный вывод исходную и отсортированную коллекции на отдельных строках, разделяя элементы пробелами.

   Программа должна запускаться с 2 дополнительными параметрами:

   - направление сортировки (ascending или descending);
   - размер вектора.

   Например,

   ```
   $./lab 4 ascending 3
   0.5 0.8 0.4
   0.4 0.5 0.8
   ```

2 Последовательности

Необходимо выполнить *все* задания.

Как и задание 1, это задание должно быть выполнено в виде консольной программы, принимающей в качестве единственного параметра номер пункта задания.

1

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

1. В очередь могут быть добавлены элементы, каждому элементу при добавлении присваивается один из трех уровней приоритета (low, normal, high).
2. 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.
3. 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. Переработайте класс так, чтобы он мог обрабатывать элементы произвольного типа.
2. Исправьте ошибки в интерфейсе, обеспечив безопасный интерфейс для применения в большом промышленном проекте.
3. Реализуйте исправленный класс, используя `std::list<>` или `std::deque<>`. Объясните выбор.
4. Реализуйте программу, обрабатывающую очередь из строк по командам, принимаемым со стандартного ввода. Каждая строка содержит ровно одну команду. Должны поддерживаться следующие команды:
   - add <priority> <data>
     Добавление элемента в очередь с приоритетом <priority> (low, normal или high).

- get
  Получение очередного элемента в очереди в соответствии с приоритетом, в стандартный вывод печатаются данные элемента. В случае, если очередь пуста, выводится строка <EMPTY>.
- accelerate
  Изменение приоритета элементов очереди.

В случае получения неподдерживаемой или неправильно сформированной команды в *стандартный вывод* печатается <INVALID COMMAND>.

2

Разработайте программу, которая:

1. Заполняет `std::list< int >` значениями от 1 до 20 со стандартного ввода, список может содержать от 0 до 20 значений.
2. Выводит содержимое списка в следующем порядке: первый элемент, последний элемент, второй элемент, предпоследний элемент, третий элемент и т.д.
3. В случае получения некорректных данных программа должна выводить сообщение об ошибке в стандартный поток ошибок и завершаться с кодом 1.

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.

3 Итераторы

Выполните *все* задания в виде программы, принимающей в качестве первого параметра номер пункта:

1

Напишите программу-«телефонную книжку», состоящую из 2-х компонентов:

1. Компонент-книжка.

   Записи (имя и телефон) должны хранится в каком-либо контейнере из STL.

   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.

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

   Помните, что после вставки и удаления элемента итераторы могут стать недействительными.

   Телефон представляется в виде последовательности цифр без разделителей.

2. Пользовательский интерфейс, принимающий команды со стандартного ввода по одной на строке и выводящий результаты на стандартный вывод. Должны поддерживаться следующие команды:

- add number "name"

  Добавление записи в конец. Кавычки не являются частью имени. Требуется учесть, что имя может содержать кавычки и обратную черту, (предваренные обратной косой чертой, как в литералах C++ [Map, табл. 3.2]), но не может содержать новую строку (например, "Name \"Nick\"Surname").

- store mark-name new-mark-name

  Сохраняет текущую позицию закладки с именем mark-name как новую закладку с именем new-mark-name. Имя содержит только символы английского алфавита, цифры и знак «минус». После запуска программы доступна 1 закладка с именем current.

- insert before mark-name number "name"

  Добавление записи перед закладкой mark-name.

- insert after mark-name number "name"

  Добавление записи после закладки mark-name.

- delete mark-name

  Удаление записи, на которую указывает закладка mark-name. После удаления закладка указывает на следующий элемент.

- show mark-name

  Показ записи, на которую указывает закладка mark-name. Если записей нет (книжка пустая), выводится <EMPTY>. Вывод должен быть выполнен без служебных последовательностей, в частности, строка из примера команды add должна быть выведена как Name "Nick"Surname.

- move mark-name steps

  Перемещение закладки mark-name на steps элементов. Если steps положительно, то закладка перемещается вперед, иначе — назад. Также в качестве steps могут быть использованы ключевые слова first и last, означающие первую и последнюю запись соответственно. Если параметр steps не число и не зарезервированное ключевое слово, в *стандартный вывод* выводится сообщение <INVALID STEP>.

Работа пользовательского интерфейса заканчивается при наступлении EOF или ошибки ввода-вывода. В случае ошибки, код возврата должен быть равен 2.

В случае получения неправильной команды необходимо вывести в *стандартный вывод* строку <INVALID COMMAND> и продолжить работу.

Если переданное имя закладки не существует в *стандартный вывод* выводится строка <INVALID BOOKMARK>.

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 хранить 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.

  Требования категории итератора должны быть выполнены.

- Выведите содержимое «контейнера» в 2 строки в стандартный вывод: первая строка в прямом направлении, вторая — в обратном при помощи `std:copy()`. Необходимо помнить, что предотвратить применение `std::reverse_iterator<>` в клиентском коде невозможно.

4 Алгоритмы I

Написать программу, которая выполняет следующие действия:

1. Заполняет `std::vector< DataStruct >` структурами `DataStruct`, прочитанными со стандартного ввода. Каждая строка содержит последовательно `key1`, `key2` и `str`, разделенные запятыми, `str` продолжается до конца строки. `key1` и `key2` в диапазоне от -5 до $+5$.
2. 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`.
3. Выводит полученный вектор на печать.

`DataStruct` определена следующим образом:

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

5 Алгоритмы II

Это задание должно быть выполнено в виде единой консольной программы. Выбор задания выполняется при помощи первого параметра при запуске.

1

Выполнить:

1. Чтение содержимого текстового файла, переданного через стандартный ввод.
2. Выделение слов, словом считается последовательность символов, разделенных пробелами и/или знаками табуляции и/или символами новой строки.
3. Вывод списка слов (по одному слову в строке), присутствующих в тексте без повторений (имеется в виду, что одно и то же слово может присутствовать в списке только один раз).

2

С помощью стандартных алгоритмов выполнить следующие действия:

1. Заполнить контейнер геометрическими фигурами, прочитав их со стандартного ввода. Этот пункт должен быть выполнен отдельным действием, нельзя совмещать дальнейшие действия с чтением данных.
2. Подсчитать общее количество вершин всех фигур (так треугольник добавляет к общему числу 3, квадрат 4 и т.д.).
3. Подсчитать количество треугольников, квадратов и прямоугольников.
4. Удалить все пятиугольники.
5. На основании оставшихся данных создать `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. Изменить контейнер так, чтобы он содержал в начале все треугольники, потом все квадраты, а потом прямоугольники.

7. Вывести на стандартный вывод результаты работы в виде:

```
Vertices: 16
Triangles: 1
Squares: 1
Rectangles: 2
Points: (5;5) (10;10) (1;1)
Shapes:
3 (1;1) (2;2) (3;1)
4 (10;10) (10;11) (11;11) (11;10)
4 (5;5) (7;5) (7;6) (5;6)
```

Пример 1: Пример вывода для работы 5.2

Геометрическая фигура задается следующей структурой:

```
1  struct Point
2  {
3    int x,y;
4  };
5
6  using Shape = std::vector< Point >;
```

Каждая точка задается парой координат $(x; y)$, фигуры указывается по одной на строке, первое число задает количество вершин, затем указываются точки-вершины:

```
3 (1;3) (23;3) (15;8)
```

Фигуры не содержат самопересечений, проверять это условие не требуется. В случае, если все точки фигуры совпадают, фигуру следует считать самым строгим вариантом такого многоугольника.

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

6 Функторы I

Разработать функтор, позволяющий собирать статистику о последовательности целых чисел. 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, написав программу, принимающую набор чисел со стандартного ввода и выводящую статистику в формате:

```
Max: 124
Min: -784365
Mean: 765,65
Positive: 15
Negative: 24
Odd Sum:  123
Even Sum: 87464
First/Last Equal: no
```

Пример 2: Пример вывода для работы 6

Если в списке нет значений, то необходимо вывести строку «No Data».

7 Функторы II

Это задание должно быть выполнено в виде единой консольной программы. Выбор задания выполняется при помощи первого параметра при запуске.

1

Используя только стандартные алгоритмы и функторы, умножить каждый элемент списка чисел с плавающей точкой на число π. Числа должны быть прочитаны со стандартного ввода, результаты выведены в стандартный вывод.

2

1. Реализовать 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 в переданный поток 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. Заполнить список указателей на различные фигуры, прочитав их со стандартного ввода.
3. С помощью стандартных алгоритмов и адаптеров вывести все фигуры.
4. С помощью стандартных алгоритмов и адаптеров отсортировать список по положению центра слева-направо (имеется в виду, что в начале списка должны идти фигуры находящиеся левее) и вывести результат фигуры.
5. С помощью стандартных алгоритмов и адаптеров отсортировать список по положению центра справа-налево и вывести фигуры.
6. С помощью стандартных алгоритмов и адаптеров отсортировать список по положению центра сверху-вниз и вывести фигуры.
7. С помощью стандартных алгоритмов и адаптеров отсортировать список по положению центра снизу-вверх и выводит фигуры.

Фигуры при вводе данных задаются по одной на строке, тип фигуры определяется ключевым словом (CIRCLE, TRIANGLE и SQUARE), после которого задаются координаты центра с скобках. Вывод различных списков отметить строками «Original:», «Left-Right:», «Right-Left:», «Top-Bottom:» и «Bottom-Top». Фигуры выводятся по одной на строку в том же формате, что и при вводе ()

```
CIRCLE (1;2)
TRIANGLE (-5;10)
SQUARE (15;5)
CIRCLE (-3;-3)
```

Пример 3: Пример ввода для работы 7.2

```
Original:
CIRCLE (1;2)
TRIANGLE (-5;10)
SQUARE (15;5)
CIRCLE (-3;-3)
Left-Right:
TRIANGLE (-5;10)
```

```
CIRCLE (-3;-3)
CIRCLE (1;2)
SQUARE (15;5)
Right-Left:
SQUARE (15;5)
CIRCLE (1;2)
CIRCLE (-3;-3)
TRIANGLE (-5;10)
Top-Bottom:
CIRCLE (-3;-3)
CIRCLE (1;2)
SQUARE (15;5)
TRIANGLE (-5;10)
Bottom-Top:
TRIANGLE (-5;10)
SQUARE (15;5)
CIRCLE (1;2)
CIRCLE (-3;-3)
```

Пример 4: Пример вывода для работы 7.2

8 Текст

Разработать программу, которая должна сделать следующее:

1. Прочитать текст со стандартного ввода, который может содержать:
   (a) Слова — состоят из латинских строчных и заглавных букв, а также дефиса «-», длина слова должна быть не более 20 символов.
   (b) Слова могут заканчиваться дефисом, но не могут с него начинаться.
   (c) Знаки препинания — «.», «,», «!», «?», «:», «;», «---» (3 дефиса, эквивалент тире). Знаки препинания не могут идти друг за другом, за исключением тире: перед тире может встречаться запятая. *Допускается обработка и других знаков препинания в соответствии с тем, как их понимают настройки пользователя.*
   (d) Числа, состоящие из цифр и десятичного разделителя, а также опционального знака числа («+» и «-»), разделение групп цифр не используется, длина числа — не более 20 символов.
   (e) Пробельные символы — пробел, табуляция, символ новой строки.
   (f) Текст *не* может начинаться с пунктуации.
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, за исключением тире, которое выделяется пробелами с обеих сторон.
   (d) После знака препинания всегда должен идти пробел.
   (e) Не допускается перенос тире на следующую строку.
3. Преобразовать полученный текст в набор строк, каждая из которых содержит целое количество слов и чисел (каждый элемент должен целиком находиться в строке) и ее длина не превышает число символов, задаваемых параметром. При этом каждая строка должна содержать максимальное число слов, не должна заканчиваться пробельными символами и начинаться со знаков препинания.
4. Вывести полученный набор строк на стандартный вывод.

Программа должна принимать необязательный аргумент --line-width с параметром, указывающим длину строки в символах. Если параметр не указан, длина строки принимается в 40 символов. Таким образом, программа может быть запущена:

```
$ ./lab --line-width 70
```

Это дает длину строки в 70 символов. Или так:

```
$ ./lab
```

Что дает значение по умолчанию в 40 символов.

Всю дополнительную информацию, в частности, десятичный разделитель, необходимо получить из настроек пользователя при помощи стандартной библиотеки[1].

[1] При использовании [MinGW] поддержка настроек, отличных от «C» и «POSIX» не требуется.