Задания к лабораторным работам

Алексей Мартынов

28 января 2021 г.

Версия 1.6*

Дополнительные задания выполняются теми, кто к сроку не представляет работу, либо представляет заведомо неполную или некорректную, а также некомпилирующуюся работу[1].

A1. Геометрические фигуры

Все числовые данные в этой работе должны быть представлены значениями с плавающей запятой.

1. Создать файл base-types.hpp, содержащий определения следующих структур:
   - `point_t`, представляющую собой точку на плоскости, координаты должны храниться в полях `x` и `y`.
   - `rectangle_t`, описывающую прямоугольник шириной `width` и высотой `height` с центром в точке `pos`.
2. Создать файл shape.hpp, содержащий определение абстрактного класса `Shape`. Этот класс должен предоставлять следующие методы:

   getArea вычисление площади

   getFrameRect получение ограничивающего прямоугольника для фигуры (см. типы из предыдущего пункта), стороны ограничивающего прямоугольника всегда параллельны осям

   move перемещение центра фигуры, 2 варианта: в конкретную точку и в виде смещений по осям абсцисс и ординат
3. Реализовать классы `Rectangle` и `Circle` в файлах rectangle.hpp, rectangle.cpp, circle.hpp и circle.cpp соответственно.
4. Продемонстрировать правильную работу классов простой программой. Демонстрация должна включать полиморфное применение классов.

Дополнительное задание

Добавить поддержку треугольников. В качестве центра треугольника здесь и далее следует использовать центр тяжести, так как его вычисление наиболее просто.

A3. Составные фигуры

1. *В виде исключения! Все дальнейшие работы должны следовать правилам оформления работ и не содержать скопированного кода, за исключением набора тестов.* Скопируйте исходные тексты задания A1.
2. Перенесите классы фигур в отдельное пространство имен. Имя этого пространства должно быть выбрано совпадающим с фамилией студента в нижнем регистре (соответственно, оно совпадает с частью имени каталога с работами до точки), например, для Петрова Ивана каталог будет называться petrov.ivan, соответственно, имя пространства имен — `petrov`. Это пространство имен должно сохраняться для всех оставшихся работ в этом семестре.
3. Добавить в класс фигуры метод `scale()`, осуществляющий изотропное масштабирование фигуры относительно ее центра с указанным коэффициентом.

   Если в первой работе был реализован треугольник, масштабирование необходимо реализовать и для него.
4. Добавить класс `CompositeShape`, который должен хранить список из произвольных фигур внутри массива. В этой работе *не допускается* использование стандартных контейнеров, необходимо самостоятельно реализовать хранение множества фигур на базе динамического массива. Класс должен быть размещен в файлах composite-shape.hpp и composite-shape.cpp.

   Для `CompositeShape` масштабирование и перемещение работают относительно центра этого объекта, за который принимается центр ограничивающего прямоугольника.
5. Написать тесты, проверяющие:

[1] Если опоздание очевидно с самого начала, можно представлять работу сразу с дополнительными заданиями.

- неизменность ширины и высоты, а также площади фигуры при перемещениях;
- квадратичное изменение площади фигуры при масштабировании;
- наличие и обработку некорректных параметров в функциях;
- корректную работу составной фигуры, необходимо помнить, что может потребоваться реализация дополнительных специальных методов в классе для обеспечения корректной работы.

Для написания тестов необходимо создать файл test-main.cpp, в котором реализовать тесты.

Список выше содержит абсолютный минимум того, что необходимо протестировать. Хорошо исполненная работа должна проверять каждый нетривиальный метод или функцию, добавленные в программу. Идеальное решение будет содержать покрытие тестами 100% реализации задания.

Созданная ранее демонстрационная программа должна быть доработана для демонстрации новых возможностей.

B1. 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 [Мар, 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
```

B3. 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++ [Мар, 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.

1 B4. 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 };
```

B5. 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)
```

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.