Т. Задание к лабораторным по ТП

Алексей Мартынов

5 мая 2022 г.

Версия 1.6

Задания к лабораторным работам семестра по дисциплине «Технологии программирования» выпроняются и принимаются по порядку. В рамках лабораторных работ недопускается использование `std::stringstream`. Для реализации модульных тестов допустипо использовать любые шаблоны и классы стандартной библиотеки

Все реализуемые сущности должны быть расположены в отдельном пространстве имен. Имя этого пространства должно совпадать 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, а имя пространства имен — `petrov`. Это пространство имен должно использоваться для всех работ.

0 Вступительная работа

1. Реализуйте программу, которая выводит на стандартный вывод на отдельной строке фамилию и имя студентая, разделённые символом «.».
2. Работа должна быть выполнена в виде 1-го исполняемого файла, не обрабатывающего параметры командной строки:

```
$ ./lab
petrov.ivan
```

1 Геометрические фигуры

Все числовые данные в этой работе должны быть представлены значениями с плавающей запятой. Использование контейнеров стандартной библиотеки в настоящей работе недопустимо (за исключением `std::string` для обработки ввода)

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

   scale изотропное масштабирование фигуры относительно её центра с указанным коэффициентом
3. Реализовать класс `Rectangle` в файлах rectangle.hpp и rectangle.cpp, соответственно.
4. Дополнительно реализовать ещё 2 (две) геометрические фигуры, указанные преподавателем.
5. Продемонстрировать правильную работу классов программой, демонстрирующей полиморфное применение классов. Использование контейнеров стандартной библиотеки в настоящей работе недопустимо. Управление памятью должно осуществляться с помощью умных указателей, предоставляемых стандартной библиотекой (см. `std::unique_ptr`, `std::shared_ptr` и `std::weak_ptr`).
   - Программа должна быть реализована в виде 1-го исполняемого файла, не обрабатывающего параметры командной строки.
   - Программа должна обрабатывать поток стандартного ввода, содержащий описание геометрических фигур. Каждая строка гарантировано содержит описание не более одной фигуры. Элементы в описании разделены ровно одним пробелом и гарантировано являются вещественными числами (за исключением названия фигур) Описание завершается командой масштабирования фигур. Пример описания:

```
CIRCLE 1.15 2.5 10.1
RECTANGLE -5.9 -3.4 3.0 4.0

SQUARE -1.55 -1.55 10.5
TRIANGLE 1.3 2.2 3.2 5.0 10.0 -5.5
POLYGON 1.3 2.2 3.2 5.0 10.0 -5.5

SCALE -15.0 -7.5 1.5
```

- Каждая фигура описывается своим набором параметров. Отсутствие самопересечений, выпуклость фигур и корректное количество, и только количество, параметров гарантируется, если не сказано иного:
  - Прямоугольник описывается парой координат своих углов: левым нижним и правым верхним. Считайте, что стороны прямоугольника параллельны осям координат. Центром фигуры считайте точку пересечения диагоналей

    ```
    RECTANGLE 1.0 1.0 3.0 4.0
    RECTANGLE 0.0 0.0 -2.0 -10.0
    ```

  - Круг описывается координатами центра и радиусом. Радиус должен быть положительным. Центром фигуры считайте центр окружности

    ```
    CIRCLE 2.0 3.0 15.0
    ```

  - Кольцо описывается координатами цетра и парой радиусов: внешней и внутренней окружности соответственно. Центром фигуры считайте центры окружностей

    ```
    RING 2.0 3.0 20.0 15.0
    ```

  - Эллипс описывается координатами центра и двумя значениями радиусов: по вертикальной оси и по горизонатльной оси. Считайте, что оси эллипса параллельны осям координатам. Центром эллипса считайте точку пересечения осей эллипса

    ```
    ELLIPSE 0.0 0.0 10.0 20.0
    ```

  - Квадрат описывается координатами своего левого нижнего угла и длиной стороны. Считайте, что стороны квадрата параллельны осям координат. Центром фигуры считайте точку пересечения диагоналей

    ```
    SQUARE 6.0 7.0 1.0
    ```

  - Треугольник описывается координатами трёх своих вершин. Условия треугольника должны быть соблюдены. Центром фигуры считать центр тяжести фигуры

    ```
    TRIANGLE 0.0 0.0 1.0 1.0 0.0 1.0
    ```

  - Параллелограмм описывается тремя вершинами, составляющими треугольник, одна из сторон которого является диагональю параллелограмма, а две дургие — сторонами параллелограмма. Стороны параллелограмма формируются первой и последней парой вершин. При этом одна из сторон должна быть параллельна оси абсцисс. Центром фигуры считать точку пересечения диагоналей

    ```
    PARALLELOGRAM 0.0 1.0 10.0 1.0 5.0 0.0
    PARALLELOGRAM 0.0 1.0 10.0 1.0 5.0 5.0
    ```

  - Ромб описывается тремя вершинами, составляющими треугольник, две стороны которого являются частью диагоналей ромба. Считайте, что диагонали ромба должны быть параллельны осям координат. Центром фигуры считайте точку пересечения диагоналей

    ```
    DIAMOND 0.0 5.0 10.0 0.0 0.0 0.0
    ```

  - Правильный многоугольник задаётся тремя вершинами. Вершины должны соответствовать условиям треугольника и формировать прямоугольный треугольник, гипотенуза которого равна радиусу описанной у многоугольника окружности, а один из катетов - радиусу вписанной в многоугольник окружности. Считайте центром фигуры первую из трёх вершин.

    ```
    REGULAR 0.0 0.0 0.0 1.0 1.0 1.0
    ```

– Полигон описывается координатам своих вершин. Центром фигуры считайте центр тяжести фигуры. При этом количество точек в описании должно быть достаточным для формирования полигона. Кроме того, никакие точки не должны совпадать

```
POLYGON 0.0 0.0 1.0 0.0 1.0 1.0 0.0 1.0
```

– Невыпуклый четырёхугольник описывается четыремя вершинами. Первые три должны удовлетворять свойствам треугольника. Четвёртая вершина должна лежать внутри треугольника, формируемого первыми тремя вершинами. Вогнутость четырёхугольника формируется последними тремя вершинами в описании. Центром фигуры считайте точку, формирующую вогнутость

```
CONCAVE 0.0 5.0 0.0 0.0 10.0 -1.0 1.0 1.0
```

– Четырёхугольник с самопересечениями задаётся координатами двух своих пересекающихся сторон. Центром фигуры считайте точку пересечения этих сторон. Кроме того, считайте, что первая и последняя вершины в описании формируют сторону фигуры

```
COMPLEXQUAD 0.0 0.0 10.0 10.0 2.0 0.0 3.0 0.0
```

- Команда масштабирования содержит параметры изотропного масштабирования: по порядку координаты центра, относительно которого необходимо произвести масштабирование, и коэффициент масштабирования. Коэффициент должен быть положительным
- После выполнения масштабирования в соответствии с указанными параметрами, программа должна вывести в стандартный вывод на отдельных строках:
  – суммарную площадь и координаты ограничивающих прямоугольников обрабатываемых фигур в порядке их описания до масштабирования.
  – суммарную площадь и координаты ограничивающих прямоугольников обрабатываемых фигур в порядке их описания после масштабирования

  Элементы должны быть разделены ровно одним пробелом, содержать один и только один знак после запятой (округление проводить в соотвествии с правилами математики) и описаны в следующем порядке.
- Пустые строки в описании фигур игнорируются
- Описания фигур не реализованных в программе должны игнорироваться, как если бы строка была пустой
- Если описание фигуры содержит ошибку, но её описание неверно, программа игнорирует её, но после масштабирования в стандартной поток ошибок должно выводиться сообщение о наличии ошибок в описании поддерживаемых фигур (конкретные фигуры указывать не требуется)
- Программа должна завершаться с сообщением об ошибке и ненулевым кодом возврата, если ввод завершился `EOF` (на Linux: `Ctrl + D` | на Windows: `Ctrl + Z` затем `Enter`), но команда масштабирования не описана или имеет некорректный коэффициент масштабирования
- Описание каждого ограничивающего прямоугольника в выводе должно содержать по порядку координаты левого нижнего угла и координаты правого верхнего угла. Например для входных данных:

```
RECTANGLE -1.0 -1.0 1.0 1.0
SQUARE -2.0 -2.0 4.0
SQUARE -10.0 -5.0 -5.0
SCALE 0.0 0.0 2.0
```

  – Если фигура SQUARE не поддерживается программой, то в стандартный поток вывода должно быть выведено:

```
4.0 -1.0 -1.0 1.0 1.0
16.0 -2.0 -2.0 2.0 2.0
```

  – Если фигура SQUARE поддерживается программой, то в стандартный поток вывода должно быть выведено:

```
20.0 -1.0 -1.0 1.0 1.0 -2.0 -2.0 2.0 2.0
80.0 -2.0 -2.0 2.0 2.0 -4.0 -4.0 4.0 4.0
```

  А в стандартный поток ошибок необходимо вывести сообщение о наличии некорректной фигуры. При этом код возврата должен быть нулевым.

2 Алгоритмы I

Написать программу, которая выполняет следующие действия:

1. Заполняет `std::vector< DataStruct >` структурами `DataStruct`, прочитанными со стандартного ввода. Чтение необходимо осуществлять с помощью итераторов и алгоритмов STL (`std::copy`, итераторы потока и перегрузки оператора побитового сдвига для чтения из потока)
2. Сортирует считанные данные следующим образом:
   (a) По возрастанию `key1`
   (b) По возрастанию `key2`, если `key1` одинаковые
   (c) По возрастанию длины строки `key3`, если прочие поля равны
3. Выводит результаты сортировки на стандартный вывод. Каждая строка должна содержать ровно один объект. Формат каждого выводимого объекта аналогичен формату ввода. Вывод необходимо осуществлять с помощью итераторов и алгоритмов STL (`std::copy`, итераторов потока и перегрузки оператора побитового сдвига для вывода в поток)

Входные данные могут содержать строки с неподдерживаемым форматом данных. Такие строки должны игнорироваться. Формат данных, который необходимо обрабатывать зависит в том числе от определения `DataStruct`

1. Тип `DataStruct` определён следующим образом:

```
struct DataStruct
{
  /* TYPE1 */ key1;
  /* TYPE2 */ key2;
  std::string key3;
};
```

   Конкретные типы полей `key1` и `key2`, а также соответствующий полям формат должны быть указаны преподавателем

2. Каждая запись ограничена парой скобок. Внутри этих скобок в качестве разделителей используются пробельные символы и символы `:`. Например:

```
(:key1 10ull:key2 'c':key3 "Data":)
```

   Порядок описания полей в структуре не определён. Например следующие структуры данных считаются идентичными:

```
(:key1 10ull:key2 'c':key3 "Data":)
(:key2 'c':key1 10ull:key3 "Data":)
(:key3 "Data":key2 'c':key1 10ull:)
```

   Имя поля и соответствующее значение гарантировано разделены ровно одним пробелом. Символы двоеточия гарантировано примыкают к прочим элементам записи. При выводе в поток поля выводятся в том порядке, в котором они заданы в структуре (запоминать исходный порядок не требуется)

3. Поля могут иметь следующий тип и соответствующий формат:

   - `[DBL LIT]` Вещественное поле с двойной точностью (**`double`**) в формате литерала:

     ```
     :keyX 50.0d:
     :keyX 50.0D:
     ```

   - `[DBL SCI]` Вещественное поле с двойной точностью (**`double`**) в научном формате:

     ```
     :keyX 5.45e-2:
     :keyX 5.45E-2:
     ```

     Число должно выводиться в стандартном виде, т.е мантисса должна быть меньше 10 и не меньше 1

   - `[SLL LIT]` Знаковое максимально доступной ёмкости (**`long long`**) в формате литерала:

```
:keyX 89ll:
:keyX -89LL:
```

- [ULL LIT] Беззнаковое максимально доступной ёмкости (**unsigned long long**) в формате литерала:

```
:keyX 89ull:
:keyX 89ULL:
```

- [ULL OCT] Беззнаковое максимально доступной ёмкости (**unsigned long long**) в формате восьмиричного литерала:

```
:keyX 076:
:keyX 01001:
```

- [ULL BIN] Беззнаковое максимально доступной ёмкости (**unsigned long long**) в формате двоичного литерала:

```
:keyX 0b1000101:
:keyX 0B001001:
```

- [ULL HEX] Беззнаковое максимально доступной ёмкости (**unsigned long long**) в формате шестнадцатиричного литерала:

```
:keyX 0xFFFA:
:keyX 0X0100f:
```

  Цифры шестандцатиричного числа выводятся в верхнем регистре

- [CHR LIT] Символ (**char**) в формате символьного литерала:

```
:keyX 'c':
:keyX 'A':
```

- [CMP LSP] Комплексное число (std::complex< **double** >) в следующем виде:

```
:keyX #c(1.0 -1.0):
:keyX #c(-1.0 1.0):
```

  Гарантируется, что вещественная и мнимая часть разделены ровно одним пробелом. При сравнении с другими полями должен быть использован модуль комплексного числа

- [RAT LSP] Рациональное число (std::pair< **long long, unsigned long long** >) в следующем виде:

```
:keyX (:N -2:D 3:):
:keyX (:N 3:D 2:):
```

- Если в качестве варианта задания выданы [CMP LSP] и [RAT LSP], то DataStruct должна быть определена следующем образом:

```
struct DataStruct
{
  std::complex< double > key1;
  std::pair< long long, unsigned long long > key2;
  std::string key3;
};
```

  При этом обрабатываемые записи имеют следующий вид:

```
(:key1 #c(1.0 -1.0):key2 (:N -1:D 5:):key3 "data":)
(:key2 (:N -1:D 5:):key3 "with : inside":key1 #c(2.0 -3.0):)
```

Все вещественные выводятся с точностью до десятичных долей. Символы, характерные для литералов выводятся в нижнем регистре (1.1e+1, 1ull, 0b01, 0x0F).

3 Алгоритмы II

С помощью стандартных алгоритмов (см. <algorithm>) и функторов (см. <functional>) реализуйте следующую программу:

1. Считайте в стандартный контейнер геометрические фигуры из файла, имя которого будет переданно в программу через параметры командной строки. Этот пункт должен быть выполнен отдельным действием, нельзя совмещать дальнейшие действия с чтением данных

   - Геометрические фигуры должны быть заданы следующей структурой:

```
1      struct Point
2      {
3        int x, y;
4      };
5      struct Polygon
6      {
7        std::vector< Point > points;
8      };
```

   - В файле с описаниями фигур на каждой строке содержится не более одного описания фигуры, содержащего по порядку: количество вершин в фигуре, последовательность из указанного количества вершин. Каждая вершина задаётся парой координат. Например, файл с описанием может иметь вид:

```
3 (1;1) (1;3) (3;3)

4 (0;0) (0;1) (1;1) (1;0)
5 (0;0) (0;1) (1;2) (2;1) (2;0)

3 (0;0) (-2;0) (0;-2)
```

     Элементы описания разделены друг от друга ровно одним пробелом.
   - Не соответствующие формату описания фигур должны игнорироваться
   - Если описание фигуры соответствует формату, то гарантируется: отсуствие самопересечений и что любая тройка вершин фигуры формирует невырожденный треугольник

2. Программа должна обрабатывать пользовательский ввод и поддерживать ряд команд. Вещественные числа в результатах выводятся с точностью до одного знака после запятой. Далее приведено описание команд с примерами, в которых подразумевается, что были считаны фигуры, описанные выше:

   - [AREA <EVEN|ODD>] Расчёт суммы площади фигур с нечётным количеством вершин и с чётным (в зависимости от переданного параметра)

```
AREA ODD
1.0
AREA EVEN
7.0
```

   - [AREA <MEAN>] Расчёт среднего площадей фигур

```
AREA MEAN
2.0
```

     Для расчёта среднего требуется по крайней мере одна фигура
   - [AREA <num-of-vertexes>] Расчёт суммы площади фигур с заданным количеством вершин

```
AREA 3
4.0
```

   - [MAX <AREA|VERTEXES>] Расчёт максимального значения площади или количества вершин (в зависимости от переданного параметра)

```
MAX AREA
3.0
MAX VERTEXES
5
```

Для расчёта максимума требуется по крайней мере одна фигура

- `[MIN <AREA|VERTEXES>]` Расчёт минимального значения площади или количества вершин (в зависимости от переданного параметра)

```
MIN AREA
1.0
MIN VERTEXES
3
```

Для расчёта минимума требуется по крайней мере одна фигура

- `[COUNT <EVEN|ODD|num-of-Vertexes>]` Подсчёт количества фигур с нечётным, чётным и конкретным количеством вершин (в зависимости от переданного параметра)

```
COUNT EVEN
3
COUNT ODD
1
COUNT 3
2
```

3. Кроме того, поддержите реализацию ещё нескольких команд, указанных преподавателем (не менее 2-х из перечисленных):

   - `PERMS <Polygon>` Подсчёт количества фигур, которые являются перестановкой координат указанной в качестве параметра. Например, для фигур

```
3 (3;3) (1;1) (1;3)
3 (0;1) (1;1) (0;0)
3 (3;1) (1;1) (3;3)
```

   Ожидается следующий результат:

```
PERMS 3 (1;1) (1;3) (3;3)
2
```

   - `MAXSEQ <Polygon>` Подсчёт максимального количества идущих подряд фигур идентичной указанной в параметрах. Например, для фигур

```
3 (3;3) (1;1) (1;3)
3 (3;3) (1;1) (1;3)
4 (0;0) (1;0) (1;1) (0;1)
3 (3;3) (1;1) (1;3)
3 (3;3) (1;1) (1;3)
3 (3;3) (1;1) (1;3)
4 (0;0) (1;0) (1;1) (0;1)
4 (0;0) (1;0) (1;1) (0;1)
3 (3;3) (1;1) (1;3)
```

   Ожидается следующий результат:

```
MAXSEQ 3 (3;3) (1;1) (1;3)
3
MAXSEQ 4 (0;0) (1;0) (1;1) (0;1)
2
MAXSEQ 4 (1;0) (1;1) (0;1) (0;0)
0
```

   - `RMECHO <Polygon>` Удаление идущих подряд дубликатов фигур идентичных указанной в параметре и вывод количества удалённых фигур. Например, для фигур

```
3 (3;3) (1;1) (1;3)
3 (3;3) (1;1) (1;3)
4 (0;0) (1;0) (1;1) (0;1)
4 (0;0) (1;0) (1;1) (0;1)
3 (3;3) (1;1) (1;3)
3 (3;3) (1;1) (1;3)
```

Ожидается следующий результат:

```
RMECHO 3 (3;3) (1;1) (1;3)
2
```

При этом, оставшиеся фигуры идентичны следующему набору

```
3 (3;3) (1;1) (1;3)
4 (0;0) (1;0) (1;1) (0;1)
4 (0;0) (1;0) (1;1) (0;1)
3 (3;3) (1;1) (1;3)
```

- ECHO <Polygon> Дублирует всякое вхождение указанной в параметре фигуры. Дубликаты добавляются сразу после идентичного элемента. Команда должна выводить количество добавленных фигур. Например, для фигур

```
3 (3;3) (1;1) (1;3)
4 (0;0) (1;0) (1;1) (0;1)
4 (0;0) (1;0) (1;1) (0;1)
3 (3;3) (1;1) (1;3)
```

Ожидается следующий результат:

```
ECHO 3 (3;3) (1;1) (1;3)
2
```

При этом, фигуры в коллекции идентичны следующему набору:

```
3 (3;3) (1;1) (1;3)
3 (3;3) (1;1) (1;3)
4 (0;0) (1;0) (1;1) (0;1)
4 (0;0) (1;0) (1;1) (0;1)
3 (3;3) (1;1) (1;3)
3 (3;3) (1;1) (1;3)
```

- LESSAREA <Polygon> Команда подсчитывает количество фигур с площадью меньшей, чем площадь фигуры, переданной в параметре. Например, для фигур

```
3 (3;3) (1;1) (1;3)
4 (0;0) (1;0) (1;1) (0;1)
3 (3;3) (1;1) (1;3)
4 (0;0) (1;0) (1;1) (0;1)
```

Ожидается следующий результат:

```
LESSAREA 3 (0;0) (2;2) (2;0)
2
```

- INFRAME <Polygon> Команда проверяет лежит ли указанная фигура целиком внутри прямоугольника, ограничивающего сохранённый в коллекции набор фигур: если да - выводится сообщение <TRUE>, иначе - <FALSE>. Например, для фигур

```
4 (0;0) (1;0) (1;1) (0;1)
4 (5;5) (6;5) (6;6) (5;6)
```

Ожидается следующий результат:

```
INFRAME 3 (0;0) (2;2) (2;0)
<TRUE>
INFRAME 3 (-1;-1) (1;1) (1;0)
<FALSE>
```

- `INTERSECTIONS <Polygon>` Команда подсчитывает количество фигур, с которыми пересекается фигура, указанная в параметрах. Например, для фигур

  ```
  4 (0;0) (1;0) (1;1) (0;1)
  4 (1;1) (2;1) (2;2) (1;2)
  ```

  Ожидается следующий результат:

  ```
  INTERSECTIONS 4 (0;0) (2;0) (2;2) (0;2)
  2
  INTERSECTIONS 4 (1;1) (3;1) (3;3) (1;3)
  2
  INTERSECTIONS 4 (2;2) (4;2) (4;4) (2;4)
  1
  ```

- `SAME <Polygon>` Команда подсчитывает количество фигур, совместимых наложением (без поворотов) с указанной в параметрах. Например, для фигур

  ```
  4 (0;0) (1;0) (1;1) (0;1)
  3 (0;0) (1;1) (0;1)
  4 (1;1) (2;1) (2;2) (1;2)
  ```

  Ожидается следующий результат:

  ```
  SAME 4 (-1;-1) (-1;0) (0;0) (0;-1)
  2
  SAME 3 (10;10) (11;11) (10;11)
  1
  SAME 3 (10;10) (10;11) (11;10)
  0
  ```

- `RECTS` Команда подсчитывает количество прямоугольников в коллекции фигур. Например, для фигур

  ```
  4 (0;0) (1;0) (1;1) (0;1)
  3 (0;0) (1;1) (0;1)
  4 (1;1) (0;2) (1;3) (2;2)
  4 (-2;1) (2;3) (3;1) (-1;-1)
  ```

  Ожидается следующий результат:

  ```
  RECTS
  3
  ```

- `RIGHTSHAPES` Команда подсчитывает количество фигур, содержащих прямые углы. Например, для фигур

  ```
  4 (0;0) (1;0) (1;1) (0;1)
  5 (-1;-1) (-2;1) (3;0) (3;-5) (0;-6)
  3 (0;0) (1;1) (0;1)
  4 (1;1) (0;2) (1;3) (2;2)
  ```

  Ожидается следующий результат:

  ```
  RIGHTSHAPES
  3
  ```

4. Если команда по каким-то причинам некорректна, то должно быть выведено сообщение <INVALID COMMAND>

5. Признаком конца ввода команд является `EOF` (на Linux: `Ctrl + D` | на Windows `Ctrl +Z` затем `Enter`)

6. Работа должа быть выполнена в виде 1-го исполняемого файла, принимающего параметры следующим образом

   ```
   $ ./lab filename
   ```

filename представляет собой обязательный параметр. Если он не задан, программа должна завершаться с ненулевым кодом возврата и сообщением об ошибке

Совет-1 Не используйте алгоритм `std::for_each` и циклы. Постарайтесь подобрать более специальный алгоритм

Совет-2 Постарайтесь использовать именно стандартные функторы и композиции из них (см. `std::bind`) Подумайте, какие типы можно реализовать дополнительно и какие операторы для них можно было бы перегрузить