Описание (вариант 2, 23)

Реализовать контейнер для хранения альтернатив и их параметров.

Обобщённый артефакт: плоские геометрические фигуры, размещаемые в координатной сетке.

Базовые альтернативы и их параметры:

- Прямоугольник:
  - целочисленная координата по оси X левого верхнего угла
  - целочисленная координата по оси У левого верхнего угла
  - целочисленная координата по оси X правого нижнего угла
  - целочисленная координата по оси У правого нижнего угла
- Треугольник:
  - целочисленная координата по оси X первого угла
  - целочисленная координата по оси У первого угла
  - целочисленная координата по оси X второго угла
  - целочисленная координата по оси У второго угла
  - целочисленная координата по оси X третьего угла
  - целочисленная координата по оси У третьего угла
- Круг:
  - целочисленная координата по оси X центра
  - целочисленная координата по оси У центра
  - целочисленный радиус

Общая для всех альтернатив переменная – цвет: красный, оранжевый, жёлтый, зелёный, голубой, синий, фиолетовый.

Общая для всех альтернатив функция: вычисление периметра.

Обработка массива результатов общих функций: перемещение периметров, которые превышают среднее арифметическое значение, в начало.

Входные и выходные данные

Форматы входной команды:

- ‘-f <файл с входными данными> <файл для вывода информации о фигурах и их периметрах> <файл для вывода обработанного массива периметров>’
  Например, ‘-f input.txt output.txt output_sorted.txt’.
- ‘-n <количество случайно генерируемых фигур> <файл для вывода информации о фигурах и их периметрах> <файл для вывода обработанного массива периметров>’
  Например, ‘-n 20 output.txt output_sorted.txt’.

Формат описания фигур в файле входных данных:

- Прямоугольник:

1 <цвет>
<целочисленная координата по оси X левого верхнего угла> <целочисленная координата по оси У левого верхнего угла> <целочисленная координата по оси X правого нижнего угла> <целочисленная координата по оси У правого нижнего угла>
Например,
1 1

0 0 1 1
- прямоугольник красного цвета с левым нижним углом в вершине [0, 0] и правым верхним – в [1, 1].

- Треугольник:

2 <цвет>
<целочисленная координата по оси X первого угла> <целочисленная координата по оси У первого угла> <целочисленная координата по оси X второго угла> <целочисленная координата по оси У второго угла> <целочисленная координата по оси X третьего угла> <целочисленная координата по оси У третьего угла>
Например,
2 2
0 0 0 1 1 0
- треугольник оранжевого цвета с координатами вершин: [0, 0], [0, 1], [1, 0].

- Круг:

3 <цвет>
<целочисленная координата по оси X центра> <целочисленная координата по оси У центра> <целочисленный радиус>
Например,
3 7
0 0 1
- круг фиолетового цвета с центром в точке [0, 0] и радиусом 1.

Выходные файлы:

- вывод информации о фигурах и их периметрах:

Filled container:
Container contains <количество фигур> elements.

- для прямоугольника:
<порядковый номер>: It is Rectangle. Left top point: x = <целочисленная координата по оси X левого верхнего угла>, y = <целочисленная координата по оси У левого верхнего угла>. Right bottom point: x = <целочисленная координата по оси X правого нижнего угла>, y = <целочисленная координата по оси У правого нижнего угла>.

- для треугольника:
<порядковый номер>: It is Triangle. A-point: x = <целочисленная координата по оси X первого угла>, y = <целочисленная координата по оси У первого угла>. B-point: x = <целочисленная координата по оси X второго угла>, y = <целочисленная координата по оси У второго угла>. C-point: x = <целочисленная координата по оси X третьего угла>, y = <целочисленная координата по оси У третьего угла>.

- для круга:
<порядковый номер>: It is Circle. Center: x = <целочисленная координата по оси X центра>, y = <целочисленная координата по оси У центра>. Radius: r = <целочисленный радиус>.

Perimeter = <периметр>. Color: <цвет>

- вывод обработанного массива периметров:

Arithmetic mean: <среднее значение>

Perimeteres processed:

<…периметры всех фигур, отсортированные в соответствии с требованием…>

Метрики

Состав: 6 заголовочных файлов и 6 файлов реализации.

Размер файлов:

- общий – 20КБ
- исполняемый код – 12.8КБ
- исполняемый файл – 88.5КБ

Время исполнения:

- 20 фигур – 4млс
- 100 фигур – 4млс
- 500 фигур – 6млс
- 2000 фигур – 19млс
- 10000 фигур – 72млс

Исходный код:

main.cpp:

```
//---------------------------------------------------------------------------
---
// main.cpp - содержит главную функцию,
// обеспечивающую простое тестирование
//---------------------------------------------------------------------------
---

#include <cstdlib> // для функций rand() и srand()
#include <ctime>   // для функции time()
#include <cstring>
#include "container.h"

void errMessage1() {
    printf("incorrect command line!\nWaited:\ncommand -f infile outfile01
outfile02\n"
           "Or:\ncommand -n number outfile01 outfile02\n");
}

void errMessage2() {
    printf("incorrect qualifier value!\nWaited:\ncommand -f infile outfile01
outfile02\n"
           "Or:\ncommand -n number outfile01 outfile02\n");
}

//---------------------------------------------------------------------------
---
int main(int argc, char* argv[]) {
    if (argc != 5) {
        errMessage1();
        return 1;
    }

    printf("Start");
    Container c;

    if (!strcmp(argv[1], "-f")) {
        FILE* file;
        file = fopen(argv[2], "r");
        c.In(file);
    }
    else if (!strcmp(argv[1], "-n")) {
        auto size = atoi(argv[2]);
        if ((size < 1) || (size > 10000)) {
            printf("incorrect number of figures = %d. Set 0 < number <=
10000\n", size);
            return 3;
        }
        // системные часы в качестве инициализатора
        srand(static_cast<unsigned int>(time(0)));
        // Заполнение контейнера генератором случайных чисел
        c.InRnd(size);
    }
    else {
        errMessage2();
        return 2;
    }

    // Вывод содержимого контейнера в файл
    FILE* file1;
    file1 = fopen(argv[3], "w");
    fprintf(file1, "Filled container:\n");
```

```
    c.Out(file1);

    // Вывод 2й части задания
    FILE* file2;
    file2 = fopen(argv[4], "w");
    c.ProcessingVar23(file2);

    printf("\nFinished");
    return 0;
}
```

container.h:

```
#ifndef __container__
#define __container__

//----------------------------------------------------------------------------
---
// container.h - содержит тип данных,
// представляющий простейший контейнер
//----------------------------------------------------------------------------
---

#include "stdio.h"
#include "shape.h"

//----------------------------------------------------------------------------
---
// Простейший контейнер на основе одномерного массива
class Container {
public:
    Container();
    ~Container();
    // Ввод содержимого контейнера из указанного потока
    void In(FILE *file);
    // Случайный ввод содержимого контейнера
    void InRnd(int size);
    // Вывод содержимого контейнера в указанный поток
    void Out(FILE *file);
    void ColorPrint(int n, FILE *file, int i);
    // Вывод обработки данных контейнера
    void ProcessingVar23(FILE* file);

private:
    void Clear();    // Очистка контейнера от данных
    int len; // текущая длина
    Shape* storage[10000];
};

#endif
```

container.cpp:

```
//----------------------------------------------------------------------------
---
// container.cpp - содержит функции обработки контейнера
//----------------------------------------------------------------------------
---

#include "stdio.h"
#include "container.h"

//----------------------------------------------------------------------------
---
```

```
// Конструктор контейнера
Container::Container(): len{0} { }

//-----------------------------------------------------------------------------
---
// Деструктор контейнера
Container::~Container() {
    Clear();
}

//-----------------------------------------------------------------------------
---
// Очистка контейнера от элементов (освобождение памяти)
void Container::Clear() {
    for (int i = 0; i < len; i++) {
        delete storage[i];
    }
    len = 0;
}

//-----------------------------------------------------------------------------
---
// Ввод содержимого контейнера из указанного потока
void Container::In(FILE *file) {
    int k;
    while (fscanf(file, "%d", &k) != EOF) {
        if ((storage[len] = Shape::StaticIn(file, k)) != 0) {
            len++;
        }
    }
}

//-----------------------------------------------------------------------------
---
// Случайный ввод содержимого контейнера
void Container::InRnd(int size) {
    while (len < size) {
        if ((storage[len] = Shape::StaticInRnd()) != nullptr) {
            len++;
        }
    }
}

//----------------------------  -------------------------------------------
-----
// Вывод содержимого контейнера в указанный поток
void Container::Out(FILE* file) {
    fprintf(file, "Container contains %d elements.\n", len);
    for (int i = 0; i < len; i++) {
        fprintf(file, "%d: ", i+1);
        storage[i]->Out(file);
        Container::ColorPrint(storage[i]->color, file, i);
    }
}

// Запись цвета в файл
void Container::ColorPrint(int n, FILE *file, int i) {
    switch (n) {
        case 1:
            fprintf(file, "red\n");
            break;
        case 2:
            fprintf(file, "orange\n");
            break;
```

```
        case 3:
            fprintf(file, "yellow\n");
            break;
        case 4:
            fprintf(file, "green\n");
            break;
        case 5:
            fprintf(file, "cyan\n");
            break;
        case 6:
            fprintf(file, "blue\n");
            break;
        case 7:
            fprintf(file, "purple\n");
            break;
        case 8:
            fprintf(file, "cannot identify\n");
            break;
        default:
            break;
    }
}

//-------------------------------------------------------------------------------
---
// Вывод обработки данных контейнера
void Container::ProcessingVar23(FILE* file) {
    double sum = 0.0;
    for (int i = 0; i < len; i++) {
        sum += storage[i]->Perimeter();
    }
    double mean = sum / len;
    fprintf(file, "Arithmetic mean: %f\n", mean);

    fprintf(file, "Perimeters processed:\n");
    for (int i = 0; i < len; i++) {
        if (storage[i]->Perimeter() > mean) {
            fprintf(file, "%f   ", storage[i]->Perimeter());
        }
    }

    for (int i = 0; i < len; i++) {
        if (storage[i]->Perimeter() <= mean) {
            fprintf(file, "%f   ", storage[i]->Perimeter());
        }
    }
}
```

shape.h:

```
#ifndef __shape__
#define __shape__

//-------------------------------------------------------------------------------
---
// shape.h - содержит описание обобщающей геометрической фигуры
//-------------------------------------------------------------------------------
---

#include "stdio.h"
#include "random.h"

//-------------------------------------------------------------------------------
---
```

```
// класс, обобщающий все имеющиеся фигуры
class Shape {
protected:
    static Random rnd20;
    static Random rnd3;
    static Random rnd7;

public:
    int color;
    virtual ~Shape() {};
    // Ввод обобщенной фигуры
    static Shape *StaticIn(FILE *file, int k);
    virtual void In(FILE *file) = 0;
    // Случайный ввод обобщенной фигуры
    static Shape *StaticInRnd();
    virtual void InRnd() = 0;
    // Вывод обобщенной фигуры
    virtual void Out(FILE *file) = 0;
    // Вычисление периметра обобщенной фигуры
    virtual double Perimeter() = 0;
};

#endif
```

shape.cpp:

```
//---------------------------------------------------------------------------
---
// shape.cpp - содержит процедуры связанные с обработкой обобщенной фигуры
// и создания произвольной фигуры
//---------------------------------------------------------------------------
---

#define _CRT_SECURE_NO_WARNINGS
#include "stdio.h"
#include "rectangle.h"
#include "triangle.h"
#include "circle.h"
//---------------------------------------------------------------------------
---
Random Shape::rnd20(1, 20);
Random Shape::rnd3(1, 3);
Random Shape::rnd7(1, 7);

//---------------------------------------------------------------------------
---
// Ввод параметров обобщенной фигуры из файла
Shape *Shape::StaticIn(FILE *file, int k) {
    Shape* sp = nullptr;
    int color;
    fscanf(file, "%d", &color);

    switch(k) {
        case 1:
            sp = new Rectangle;
            sp->color = color;
            break;
        case 2:
            sp = new Triangle;
            sp->color = color;
            break;
        case 3:
            sp = new Circle;
            sp->color = color;
```

```
            break;
        default:
            return 0;
    }
    sp->In(file);
    return sp;
}

// Случайный ввод обобщенной фигуры
Shape *Shape::StaticInRnd() {
    auto k = Shape::rnd3.Get();
    auto color = Shape::rnd7.Get();
    Shape* sp = nullptr;
    switch(k) {
        case 1:
            sp = new Rectangle;
            sp->color = color;
            break;
        case 2:
            sp = new Triangle;
            sp->color = color;
            break;
        case 3:
            sp = new Circle;
            sp->color = color;
            break;
    }
    sp->InRnd();
    return sp;
}
```

rectangle.h:

```
#ifndef __rectangle__
#define __rectangle__

//---------------------------------------------------------------------------
---
// rectangle.h - содержит описание прямоугольника  и его интерфейса
//---------------------------------------------------------------------------
---

#include "stdio.h"
#include "random.h"
#include "shape.h"

// прямоугольник
class Rectangle: public Shape {
public:
    virtual ~Rectangle() {}
    // Ввод параметров прямоугольника из файла
    virtual void In(FILE *file);
    // Случайный ввод параметров прямоугольника
    virtual void InRnd();
    // Вывод параметров прямоугольника в форматируемый поток
    virtual void Out(FILE* file);
    // Вычисление периметра прямоугольника
    virtual double Perimeter();

private:
    int x_l_t, y_l_t; // координаты левого верхнего угла
    int x_r_b, y_r_b; // координаты правого нижнего угла
};
```

```
#endif //__rectangle__
```

rectangle.cpp:

```
//--------------------------------------------------------------------------
---
// rectangle.cpp - содержит процедуры для работы с прямоугольником
//--------------------------------------------------------------------------
---

#include "stdio.h"
#include "rectangle.h"

//--------------------------------------------------------------------------
---
// Ввод параметров прямоугольника из файла

void Rectangle::In(FILE* file) {
    fscanf(file, "%d", &x_l_t);
    fscanf(file, "%d", &y_l_t);
    fscanf(file, "%d", &x_r_b);
    fscanf(file, "%d", &y_r_b);
}

// Случайный ввод параметров прямоугольника
void Rectangle::InRnd() {
    x_l_t = Shape::rnd20.Get();
    y_l_t = Shape::rnd20.Get();
    do {
        x_r_b = Shape::rnd20.Get();
    } while (x_r_b == x_l_t);
    do {
        y_r_b = Shape::rnd20.Get();
    } while (y_r_b == y_l_t);
}

//--------------------------------------------------------------------------
---
// Вывод параметров прямоугольника в форматируемый поток
void Rectangle::Out(FILE* file) {
    fprintf(file, "It is Rectangle. Left top point: x = %d, y = %d. Right
bottom point: x = %d, y = %d. Perimeter = %f. Color: ",
            x_l_t, y_l_t, x_r_b, y_r_b, Perimeter());
}

//--------------------------------------------------------------------------
---
// Вычисление периметра прямоугольника
double Rectangle::Perimeter() {
    int x = std::abs(x_l_t - x_r_b);
    int y = std::abs(y_l_t - y_r_b);
    return 2.0 * (x + y);
}
```

triangle.h:

```
#ifndef __triangle__
#define __triangle__

//--------------------------------------------------------------------------
---
// triangle.h - содержит описание треугольника
```

```
//---------------------------------------------------------------------------
---

#include "stdio.h"
#include "random.h"
#include "shape.h"

// треугольник
class Triangle: public Shape {
public:
    virtual ~Triangle() {}
    // Ввод параметров треугольника из файла
    virtual void In(FILE *file);
    // Случайный ввод параметров треугольника
    virtual void InRnd();
    // Вывод параметров треугольника в форматируемый поток
    virtual void Out(FILE *file);
    // Вычисление периметра треугольника
    virtual double Perimeter();

private:
    int x_1, y_1; // координаты 1го угла
    int x_2, y_2; // координаты 2го угла
    int x_3, y_3; // координаты 3го угла
};

#endif //__triangle__
```

triangle.cpp:

```
//---------------------------------------------------------------------------
---
// triangle.cpp - содержит процедуры для работы с треугольником
//---------------------------------------------------------------------------
---

#include "stdio.h"
#include "math.h"
#include "triangle.h"

//---------------------------------------------------------------------------
---
// Ввод параметров треугольника из потока
void Triangle::In(FILE* file) {
    fscanf(file, "%d", &x_1);
    fscanf(file, "%d", &y_1);
    fscanf(file, "%d", &x_2);
    fscanf(file, "%d", &y_2);
    fscanf(file, "%d", &x_3);
    fscanf(file, "%d", &y_3);
}

// Случайный ввод параметров треугольника
void Triangle::InRnd() {
    int a, b, c, x, y;
    do {
        x_1 = Shape::rnd20.Get();
        y_1 = Shape::rnd20.Get();
        x_2 = Shape::rnd20.Get();
        y_2 = Shape::rnd20.Get();
        x_3 = Shape::rnd20.Get();
        y_3 = Shape::rnd20.Get();

        x = std::abs(x_1 - x_2);
```

```
        y = std::abs(y_1 - y_2);
        a = sqrt(x * x + y * y);

        x = std::abs(x_2 - x_3);
        y = std::abs(y_2 - y_3);
        b = sqrt(x * x + y * y);

        x = std::abs(x_3 - x_1);
        y = std::abs(y_3 - y_1);
        c = sqrt(x * x + y * y);
    } while (a >= b + c || b >= a + c || c >= a + b);
}

//--------------------------------------------------------------------------------
---
// Вывод параметров треугольника в поток
void Triangle::Out(FILE* file) {
    fprintf(file, "It is Triangle. A-point: x = %d, y = %d. B-point: x = %d,
y = %d. C-point: x = %d, y = %d. Perimeter = %f. Color: ",
            x_1, y_1, x_2, y_2, x_3, y_3, Perimeter());
}

//--------------------------------------------------------------------------------
---
// Вычисление периметра треугольника
double Triangle::Perimeter() {
    int x, y;
    x = std::abs(x_1 - x_2);
    y = std::abs(y_1 - y_2);
    double a = sqrt(x * x + y * y);

    x = std::abs(x_2 - x_3);
    y = std::abs(y_2 - y_3);
    double b = sqrt(x * x + y * y);

    x = std::abs(x_3 - x_1);
    y = std::abs(y_3 - y_1);
    double c = sqrt(x * x + y * y);
    return a + b + c;
}
```

circle.h:

```
#ifndef __circle__
#define __circle__

//--------------------------------------------------------------------------------
---
// circle.h - содержит описание круга
//--------------------------------------------------------------------------------
---

#include "stdio.h"
#include "random.h"
#include "shape.h"

// круг
class Circle: public Shape {
public:
    virtual ~Circle() {}
    // Ввод параметров треугольника из файла
    virtual void In(FILE* file);
    // Случайный ввод параметров треугольника
    virtual void InRnd();
```

```
    // Вывод параметров треугольника в форматируемый поток
    virtual void Out(FILE* file);
    // Вычисление периметра треугольника
    virtual double Perimeter();

private:
    int x_c, y_c; // координаты центра
    int r; // радиус
};

#endif //__circle__
```

circle.cpp:

```
//---------------------------------------------------------------------------
---
// circle.cpp - содержит процедуры для работы с кругом
//---------------------------------------------------------------------------
---

#include "stdio.h"
#include "math.h"
#include "circle.h"

//---------------------------------------------------------------------------
---
// Ввод параметров круга из потока
void Circle::In(FILE* file) {
    fscanf(file, "%d", &x_c);
    fscanf(file, "%d", &y_c);
    fscanf(file, "%d", &r);
}

// Случайный ввод параметров круга
void Circle::InRnd() {
    x_c = Shape::rnd20.Get();
    y_c = Shape::rnd20.Get();
    r = Shape::rnd20.Get();
}

//---------------------------------------------------------------------------
---
// Вывод параметров круга в поток
void Circle::Out(FILE* file) {
    fprintf(file, "It is Circle. Center: x = %d, y = %d. Radius: r = %d.
Perimeter = %f. Color: ",
            x_c, y_c, r, Perimeter());
}

//---------------------------------------------------------------------------
---
// Вычисление длины круга
double Circle::Perimeter() {
    return 2 * M_PI * r;
}
```

random.h:

```
#ifndef __random__
#define __random__

#include <cstdlib>
#include <ctime>   // для функции time()
```

```
//-------------------------------------------------------------------------
---
// random.h - содержит генератор случайных чисел в диапазоне от 1 до 20
//-------------------------------------------------------------------------
---

class Random {
public:
    Random(int fst, int lst) {
        if(fst <= lst) {
            first = fst;
            last = lst;
        } else {
            first = lst;
            last = fst;
        }
        // системные часы в качестве инициализатора
        srand(static_cast<unsigned int>(time(0)));
    }

    // Генерация случайного числа в заданном диапазоне
    int Get() {
        return rand() % (last - first + 1) + first;
    }

private:
    int first;
    int last;
};

#endif //__random__
```