Описание (вариант 2, 23)

Реализовать контейнер плоских геометрических фигур, размещаемых в координатной сетке, с вариациями: прямоугольник, треугольник, круг.

Входные данные - тип фигуры и цвет, обозначаемые цифрой, далее:
- прямоугольник: целочисленные координаты левого верхнего и правого нижнего углов
- треугольник: целочисленные координаты всех трёх углов
- круг: целочисленные координаты центра и радиус
(реализовано в виде команды -f <файл с входными данными> < файл для вывода информации о фигурах и их периметров> < файл для вывода отсортированного массива периметров>)

Формат файла входных данных:
<число, обозначающее фигуру> <число, задающее цвет>
<параметры фигуры>
Например:
1 [прямоугольник] 1 [красный]
1 1 [левый верхний угол в точке 1,1] 2 2 [правый нижний – в точке 2,2]

Реализовать возможность случайной генерации (реализовано в виде команды -n <число фигур> <файл для вывода информации о фигурах и их периметров> <файл для вывода отсортированного массива периметров>).

Вычислить периметр каждой фигуры и отсортировать полученные результаты, переместив в начало периметры, превышающие среднее арифметическое значение.

Метрики

Программа состоит из 6ти заголовочных файлов и 6ти модулей с реализацией.

Общий размер файлов с кодом: ~19.8КБ.

Время работы (случайная генерация фигур):
- 5 фигур: 2млс
- 20 фигур: 3 млс
- 100 фигур: 9 млс
- 500 фигур: 11 млс
- 2 000 фигур: 56 млс
- 10 000 фигур: 222 млс

Тей Даяна, БПИ-208

Исходный код:

Main.cpp:

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

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

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[]) {
    //unsigned int start_time = clock(); // начальное время
    if (argc != 5) {
        errMessage1();
        return 1;
    }

    printf("Start");
    container c;
    Init(c);

    if (!strcmp(argv[1], "-f")) {
        FILE* file;
        file = fopen(argv[2], "r");
        In(c, file);
    }
    else if (!strcmp(argv[1], "-n")) {
        auto size = atoi(argv[2]);
        if ((size < 1) || (size > 10000)) {
            printf("incorrect numer 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
        InRnd(c, size);
    }
    else {
        errMessage2();
        return 2;
    }

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

    // Вывод 2й части задания
```

```
    FILE* file2;
    file2 = fopen(argv[4], "w");
    ProcessingVar23(c, file2);

    Clear(c);
    printf("\nFinished");
    //unsigned int end_time = clock(); // конечное время
    //printf("%d", end_time - start_time); // искомое время
    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
struct container {
    enum { max_len = 10000 }; // максимальная длина
    int len; // текущая длина
    shape* cont[max_len];
};

// Инициализация контейнера
void Init(container& c);

// Очистка контейнера от элементов (освобождение памяти)
void Clear(container& c);

// Ввод содержимого контейнера из указанного потока
void In(container& c, FILE *file);

// Случайный ввод содержимого контейнера
void InRnd(container& c, int size);

// Вывод содержимого контейнера в указанный поток
void Out(container& c, FILE* file);

// Вывод обработки данных контейнера
void ProcessingVar23(container& c, FILE* file);

#endif
```

Container.cpp:

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

#define _CRT_SECURE_NO_WARNINGS
#include "stdio.h"
#include "container.h"

//----------------------------------------------------------------------------
// Инициализация контейнера
void Init(container& c) {
    c.len = 0;
}
```

```
//-----------------------------------------------------------------------------
// Очистка контейнера от элементов (освобождение памяти)
void Clear(container& c) {
    for (int i = 0; i < c.len; i++) {
        delete c.cont[i];
    }
    c.len = 0;
}

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

//-----------------------------------------------------------------------------
// Случайный ввод содержимого контейнера
void InRnd(container& c, int size) {
    while (c.len < size) {
        if ((c.cont[c.len] = InRnd()) != nullptr) {
            c.len++;
        }
    }
}

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

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

    fprintf(file, "Perimeteres processed:\n");
    for (int i = 0; i < c.len; i++) {
        if (Perimeter(*(c.cont[i])) > mean) {
            fprintf(file, "%f   ", Perimeter(*(c.cont[i])));
        }
    }

    for (int i = 0; i < c.len; i++) {
        if (Perimeter(*(c.cont[i])) <= mean) {
            fprintf(file, "%f   ", Perimeter(*(c.cont[i])));
        }
    }
}
```

Shape.h:

```
#ifndef __shape__
#define __shape__

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

#include "stdio.h"
#include "rectangle.h"
#include "triangle.h"
#include "circle.h"

//---------------------------------------------------------------------------
// структура, обобщающая все имеющиеся фигуры
struct shape {
    // значения ключей для каждой из фигур
    enum key { RECTANGLE, TRIANGLE, CIRCLE };
    enum color { RED, ORANGE, YELLOW, GREEN, CYAN, BLUE, PURPLE, NONE };
    key k; // ключи
    color col;
    // используемые альтернативы
    union { // используем простейшую реализацию
        rectangle r;
        triangle t;
        circle c;
    };
};

// Ввод обобщенной фигуры
shape* In(FILE *file, int k);

// Получение типа цвета по числу
shape::color Color(int n);

// Случайный ввод обобщенной фигуры
shape* InRnd();

// Вывод обобщенной фигуры
void Out(shape& s, FILE* file);

// Запись цвета в файл
void ColorPrint(shape::color color, FILE* file);

// Вычисление периметра обобщенной фигуры
double Perimeter(shape& s);

#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 "shape.h"

//---------------------------------------------------------------------------
// Ввод параметров обобщенной фигуры из файла
shape* In(FILE *file, int k) {
    shape* sp;
    int color;
    fscanf(file, "%d", &color);
    switch (k) {
```

```
    case 1:
        sp = new shape;
        sp->k = shape::RECTANGLE;
        sp->col = Color(color);
        In(sp->r, file);
        return sp;
    case 2:
        sp = new shape;
        sp->k = shape::TRIANGLE;
        sp->col = Color(color);
        In(sp->t, file);
        return sp;
    case 3:
        sp = new shape;
        sp->k = shape::CIRCLE;
        sp->col = Color(color);
        In(sp->c, file);
        return sp;
    default:
        return 0;
    }
}

// Получение типа цвета по числу
shape::color Color(int n) {
    switch (n) {
    case 1:
        return shape::RED;
    case 2:
        return shape::ORANGE;
    case 3:
        return shape::YELLOW;
    case 4:
        return shape::GREEN;
    case 5:
        return shape::CYAN;
    case 6:
        return shape::BLUE;
    case 7:
        return shape::PURPLE;
    default:
        return shape::NONE;
    }
}

// Случайный ввод обобщенной фигуры
shape* InRnd() {
    shape* sp;
    auto k = rand() % 3 + 1;
    auto color = rand() % 7 + 1;
    switch (k) {
    case 1:
        sp = new shape;
        sp->k = shape::RECTANGLE;
        sp->col = Color(color);
        InRnd(sp->r);
        return sp;
    case 2:
        sp = new shape;
        sp->k = shape::TRIANGLE;
        sp->col = Color(color);
        InRnd(sp->t);
        return sp;
    case 3:
        sp = new shape;
```

```
        sp->k = shape::CIRCLE;
        sp->col = Color(color);
        InRnd(sp->c);
        return sp;
    default:
        return 0;
    }
}

//---------------------------------------------------------------------------
// Вывод параметров текущей фигуры в поток
void Out(shape& s, FILE *file) {
    switch (s.k) {
    case shape::RECTANGLE:
        Out(s.r, file);
        break;
    case shape::TRIANGLE:
        Out(s.t, file);
        break;
    case shape::CIRCLE:
        Out(s.c, file);
        break;
    default:
       fprintf(file, "Incorrect figure!");
       return;
    }
    ColorPrint(s.col, file);
}

// Запись цвета в файл
void ColorPrint(shape::color color, FILE *file) {
    switch (color) {
    case shape::RED:
        fprintf(file, "red\n");
        break;
    case shape::ORANGE:
        fprintf(file, "orange\n");
        break;
    case shape::YELLOW:
        fprintf(file, "yellow\n");
        break;
    case shape::GREEN:
        fprintf(file, "green\n");
        break;
    case shape::CYAN:
        fprintf(file, "cyan\n");
        break;
    case shape::BLUE:
        fprintf(file, "blue\n");
        break;
    case shape::PURPLE:
        fprintf(file, "purple\n");
        break;
    case shape::NONE:
        fprintf(file, "cannot identify\n");
        break;
    default:
        break;
    }
}

//---------------------------------------------------------------------------
// Вычисление периметра фигуры
double Perimeter(shape& s) {
    switch (s.k) {
```

```
    case shape::RECTANGLE:
        return Perimeter(s.r);
        break;
    case shape::TRIANGLE:
        return Perimeter(s.t);
        break;
    case shape::CIRCLE:
        return Perimeter(s.c);
        break;
    default:
        return 0.0;
    }
}
```

Rectangle.h:

```
#ifndef __rectangle__
#define __rectangle__

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

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

// прямоугольник
struct rectangle {
    int x_l_t, y_l_t; // координаты левого верхнего угла
    int x_r_b, y_r_b; // координаты правого нижнего угла
};

// Ввод параметров прямоугольника из файла
void In(rectangle& r, FILE *file);

// Случайный ввод параметров прямоугольника
void InRnd(rectangle& r);

// Вывод параметров прямоугольника в форматируемый поток
void Out(rectangle& r, FILE* file);

// Вычисление периметра прямоугольника
double Perimeter(rectangle& r);

#endif //__rectangle__
```

Rectangle.cpp:

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

#define _CRT_SECURE_NO_WARNINGS
#include "stdio.h"
#include "rectangle.h"

//----------------------------------------------------------------------------
// Ввод параметров прямоугольника из файла
void In(rectangle& r, FILE* file) {
    fscanf(file, "%d", &r.x_l_t);
    fscanf(file, "%d", &r.y_l_t);
    fscanf(file, "%d", &r.x_r_b);
    fscanf(file, "%d", &r.y_r_b);
}

// Случайный ввод параметров прямоугольника
void InRnd(rectangle& r) {
```

```
    r.x_l_t = Random();
    r.y_l_t = Random();
    do {
        r.x_r_b = Random();
    } while (r.x_r_b == r.x_l_t);
    do {
        r.y_r_b = Random();
    } while (r.y_r_b == r.y_l_t);
}

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

//--------------------------------------------------------------------------
// Вычисление периметра прямоугольника
double Perimeter(rectangle& r) {
    int x = std::abs(r.x_l_t - r.x_r_b);
    int y = std::abs(r.y_l_t - r.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"

// треугольник
struct triangle {
    int x_1, y_1; // координаты 1го угла
    int x_2, y_2; // координаты 2го угла
    int x_3, y_3; // координаты 3го угла
};

// Ввод параметров треугольника из файла
void In(triangle& t, FILE *file);

// Случайный ввод параметров треугольника
void InRnd(triangle& t);

// Вывод параметров треугольника в форматируемый поток
void Out(triangle& t, FILE* file);

// Вычисление периметра треугольника
double Perimeter(triangle& t);

#endif //__triangle__
```

Triangle.cpp:

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

#define _CRT_SECURE_NO_WARNINGS
#include "stdio.h"
```

```cpp
#include "math.h"
#include "triangle.h"

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

// Случайный ввод параметров треугольника
void InRnd(triangle& t) {
    int a, b, c, x, y;
    do {
        t.x_1 = Random();
        t.y_1 = Random();
        t.x_2 = Random();
        t.y_2 = Random();
        t.x_3 = Random();
        t.y_3 = Random();

        x = std::abs(t.x_1 - t.x_2);
        y = std::abs(t.y_1 - t.y_2);
        a = sqrt(x * x + y * y);

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

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

//---------------------------------------------------------------------------
// Вывод параметров треугольника в поток
void Out(triangle& t, 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: ",
        t.x_1, t.y_1, t.x_2, t.y_2, t.x_3, t.y_3, Perimeter(t));
}

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

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

    x = std::abs(t.x_3 - t.x_1);
    y = std::abs(t.y_3 - t.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"

// круг
struct circle {
    int x_c, y_c; // координаты центра
    int r; // радиус
};

// Ввод параметров круга из файла
void In(circle& c, FILE* file);

// Случайный ввод параметров круга
void InRnd(circle& c);

// Вывод параметров круга в форматируемый поток
void Out(circle& c, FILE* file);

// Вычисление длины круга
double Perimeter(circle& c);

#endif //__circle__
```

Circle.cpp:

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

#define _CRT_SECURE_NO_WARNINGS
#define _USE_MATH_DEFINES
#include "stdio.h"
#include "math.h"
#include "circle.h"

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

// Случайный ввод параметров круга
void InRnd(circle& c) {
    c.x_c = Random();
    c.y_c = Random();
    c.r = Random();
}

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

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

```cpp
// Вычисление длины круга
double Perimeter(circle& c) {
    return 2 * M_PI * c.r;
}
```

Random.h:

```cpp
#ifndef __random__
#define __random__

#include <cstdlib>

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

inline auto Random() {
    return rand() % 20 + 1;
}

#endif //__random__
```