Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет: «Информационных технологий и прикладной математики»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Объектно-ориентированное программирование»

**Лабораторная работа № 3**

**Тема: Механизмы наследования в C++**

|  |  |
| --- | --- |
| Cтудент: | Королев И.М. |
| Группа: | 8О-208Б |
| Преподаватель: | Чернышов Л.Н. |
| Дата: |  |
| Оценка: |  |

Москва 2020

**Оглавление**

[1. Постановка задачи 3](#_Toc54570088)

[2. Описание программы 4](#_Toc54570089)

[3. Тесты 6](#_Toc54570093)

[4. Результаты выполнения тестов 7](#_Toc54570094)

[5. Листинг программы 13](#_Toc54570095)

[6. Выводы 22](#_Toc54570096)

[Список используемых источников 23](#_Toc54570097)

# 1. Постановка задачи

Разработать классы согласно варианту задания, классы должны наследоваться от базового класса Figure. Фигуры являются фигурами вращения. Все классы должны поддерживать набор общих методов:

1. Вычисление геометрического центра фигуры;
2. Вывод в стандартный поток вывода std::cout координат вершин фигуры;
3. Вычисление площади фигуры.

Создать программу, которая позволяет:

* Вводить из стандартного потока std::cin фигуры, согласно варианту задания.
* Сохранять созданные фигуры в динамический массив std::vector<Figure\*>.
* Вызывать для всего массива общие функции (вывод для каждой фигуры в массиве геометрического центра, координат вершин и площади).
* Вычислять общую площадь фигур в массиве.
* Удалять из массива фигуру по индексу.

Таблица 1 – вариант задания

|  |  |  |  |
| --- | --- | --- | --- |
| **Вариант** | **Фигура № 1** | **Фигура № 2** | **Фигура № 3** |
| 29 | Ромб | 5-угольник | 6-угольник |

# 2. Описание программы

Программа выполняет чтение данных из консоли или файла (с помощью cat | ./исполняемый файл) и выполняет заданные действия. При запуске программы появляется меню, в котором будут предложены следующие варианты действий:

1. Добавить в динамический массив новую фигуру.
2. Вывести все фигуры из динамического массива, их геометрический центр, координаты их вершин и их площадь.
3. Вывести общую площадь фигур, находящихся в динамическом массиве.
4. Удалить фигуру из динамического массива по заданному индексу.
5. Завершить работу программы.

В программе был реализован класс *Figure*, от которого наследуются классы *Rhombus*, *Pentagon* и *Hexagon*.

В классах *Rhombus*, *Pentagon* и *Hexagon* переопределены методы вывода геометрического центра, вывода координат вершин и получения площади фигуры. Именно для этого в классе *Figure* эти методы были объявлены как виртуальные. Как пример, если бы функция площади фигуры не была переопределена в наследующихся классах, то она могла бы находиться у всех фигур одинаково, что является неверным.

Для нахождения общей площади в динамическом массиве программа проходит по всему массиву и суммирует площади отдельных фигур в нём.

Для удаления фигуры из динамического массива к его началу прибавляется индекс, и фигура, которая стоит на указанном месте удаляется.

Если введённый угол ромба больше 180 градусов или меньше либо равен 0 градусам, если сторона фигуры отрицательная, то программа попросит ввести данные снова.

# Переменные и методы класса Figure

*string Name*  – название фигуры

*void PrintName()* – функция вывода названия фигуры.

*virtual void PrintCenter()* – функция вывода геометрического центра фигуры.

*virtual void PrintCoordinates()* – функция вывода координат вершин фигуры.

*virtual float Square()* – функция нахождения площади фигуры.

# Переменные и методы классов, наследуемых от класса Figure

Были созданы три класса, которые наследуются от класса Figure: *Rhombus*, *Pentagon*, *Hexagon*. У всех трёх классов есть переменные:

*float Center\_x*  – координата геометрического центра фигуры по оси x.

*float Center\_y*  – координата геометрического центра фигуры по оси y.

*float SideLength*  – длина стороны фигуры.

У класса *Rhombus* также есть переменная *Angle*, которая обозначает один из углов ромба.

У всех трёх классов есть методы:

*void GetCenter()*  – функция получения координат геометрического центра фигуры.

*void GetSide()*  – функция получения размера стороны фигуры.

*void PrintCenter()*  – функция вывода геометрического центра фигуры.

*void PrintCoordinates()*  – функция вывода координат вершин фигуры.

*float Square()*  – функция нахождения площади фигуры.

У класса *Rhombus* также есть метод *GetAngle()* – функция получения размера одного из углов ромба.

# Описание функций

*Figure \*enter\_in\_figure(string figure)* – функция ввода значений заданной фигуры.

*Figure \*figure\_enter()* – проверяет, существует ли введённая фигура, и, если существует, то вызывает функцию *enter\_in\_figure(string figure)* для ввода значений этой фигуры.

*float square\_all\_figures(vector<Figure\*> vector)* – функция подсчёта общей площади фигур, находящихся в динамическом массиве. Проходит по всему динамическому массиву и суммирует все площади фигур, после чего возвращает результат.

*void figure\_delete(vector<Figure\*> &vector, int index)* – функция удаления фигуры из динамического массива по индексу. Прибавляет заданный индекс к началу массива и удаляет фигуру, стоящую на полученном месте.

*int menu()* – функция вывода диалогового меню и последующей работы с введёнными значениями.

# 3. Тесты

Были проведены тесты, на вводе которых написаны параметры фигур и действия, которые можно выполнять в программе.

Таблица 2 – таблица тестов

|  |  |  |
| --- | --- | --- |
| test\_01.txt | test\_02.txt | test\_03.txt |
| 1 rhombus 0 0 4 90  1 hexagon 0 0 -6  6  1 pentagon 0 0 7  2  3  4 1  2  3  5 | 1 pentagon -17.5 -232 17  3  1 rhombus 2.3 -11.9 13 43  1 rhombus -19.4 3.8 0.7 36  1 hexagon 212 5 134  2  3  4 3  4 0  2  5 | 1 hexagon -2 19 337  1 rhombus 176.1 -34 47.5 17  2  3  1 pentagon -4 -71 22.7  4 4  0  1 hexagon 21 41 6.4  2  3  5 |

# 4. Результаты выполнения тестов

В результатах теста отражены случаи, когда происходит вывод результата для введённого действия или введённое значение является неверным. Вывод всего меню опускается.

Таблица 2 – test\_01.txt и результаты его выполнения

|  |  |
| --- | --- |
| **Ввод** | **Результаты** |
| 1 rhombus 0 0 4 90 1 hexagon 0 0 -6 | Please, re-enter side size: |
| 6 1 pentagon 0 0 7  2 | rhombus Center (x, y): (0, 0) (x1, y1 = (2.82843, 0) (x2, y2 = (0, 2.82843) (x3, y3 = (-2.82843, 0) (x4, y4 = (0, -2.82843) Square: 16  hexagon Center (x, y): (0, 0) (x1, y1) = (6, 0) (x2, y2) = (3, 5.1962) (x3, y3) = (-3, 5.1962) (x4, y4) = (-6, 0) (x5, y5) = (-3, -5.1962) (x6, y6) = (3, -5.1962) Square: 93.5307  pentagon Center (x, y): (0, 0) (x1, y1) = (5.9546, 0) (x2, y2) = (1.8401, 5.6631) (x3, y3) = (-4.8173, 3.5) (x4, y4) = (-4.8173, -3.5) (x5, y5) = (1.8401, -5.6631) Square: 84.3034 |
| 3 | Total squre of all figures: 193.834 |
| 4 1 | The figure has been removed! |

Продолжение таблицы 2

|  |  |
| --- | --- |
| 2 | rhombus Center (x, y): (0, 0) (x1, y1 = (2.82843, 0) (x2, y2 = (0, 2.82843) (x3, y3 = (-2.82843, 0) (x4, y4 = (0, -2.82843) Square: 16  pentagon Center (x, y): (0, 0) (x1, y1) = (5.9546, 0) (x2, y2) = (1.8401, 5.6631) (x3, y3) = (-4.8173, 3.5) (x4, y4) = (-4.8173, -3.5) (x5, y5) = (1.8401, -5.6631) Square: 84.3034 |
| 3 | Total squre of all figures: 100.303 |
| 5 | Program completed! |

Таблица 3 – test\_02.txt и результаты его выполнения

|  |  |
| --- | --- |
| **Ввод** | **Результаты** |
| 1 pentagon -17.5 -232 17  3 | Total square of all figures: 497.218 |
| 1 rhombus 2.3 -11.9 13 43 1 rhombus -19.4 3.8 0.7 36 1 hexagon 212 5 134 2 | pentagon Center (x, y): (-17.5, -232) (x1, y1) = (-3.0389, -232) (x2, y2) = (-13.0313, -218.247) (x3, y3) = (-29.1992, -223.5) (x4, y4) = (-29.1992, -240.5) (x5, y5) = (-13.0313, -245.753) Square: 497.218  rhombus Center (x, y): (2.3, -11.9) (x1, y1) = (14.3954, 0) (x2, y2) = (0, -7.13548) (x3, y3) = (-9.79543, 0) (x4, y4) = (0, -16.6645) Square: 115.258  rhombus Center (x, y): (-19.4, 3.8) (x1, y1) = (-18.7343, 0) (x2, y2) = (0, 4.01631) (x3, y3) = (-20.0657, 0) (x4, y4) = (0, 3.58369) Square: 0.288  hexagon Center (x, y): (212, 5) (x1, y1) = (346, 5) (x2, y2) = (279, 121.047) (x3, y3) = (145, 121.047) (x4, y4) = (78, 5) (x5, y5) = (145, -111.047) (x6, y6) = (279, -111.047) Square: 46651.1 |
| 3 | Total square of all figures: 47263.8 |
| 4 3 | The figure has been removed! |
| 4 0 | The figure has been removed! |

Продолжение таблицы 3

|  |  |
| --- | --- |
| 2 | rhombus Center (x, y): (2.3, -11.9) (x1, y1 = (14.3954, 0) (x2, y2 = (0, -7.13548) (x3, y3 = (-9.79543, 0) (x4, y4 = (0, -16.6645) Square: 115.258  rhombus Center (x, y): (-19.4, 3.8) (x1, y1) = (-18.7343, 0) (x2, y2) = (0, 4.01631) (x3, y3) = (-20.0657, 0) (x4, y4) = (0, 3.58369) Square: 0.288 |
| 5 | Program completed! |

Таблица 4 – test\_03.txt и результаты его выполнения

|  |  |
| --- | --- |
| **Ввод** | **Результаты** |
| 1 hexagon -2 19 337 1 rhombus 176.1 -34 47.5 17 2 | hexagon Center (x, y): (-2, 19) (x1, y1) = (335, 19) (x2, y2) = (166.5, 310.851) (x3, y3) = (-170.5, 310.851) (x4, y4) = (-339, 19) (x5, y5) = (-170.5, -272.851) (x6, y6) = (166.5, -272.851) Square: 295061  rhombus Center (x, y): (176.1, -34) (x1, y1 = (223.078, 0) (x2, y2 = (0, -26.9791) (x3, y3 = (129.122, 0) (x4, y4 = (0, -41.0209) Square: 659.664 |
| 3 | Total square of all figures: 295721 |
| 1 pentagon -4 -71 22.7 4 4 | Write an index of figure: The index is entered incorrectly Please, write an index: |
| 0 | The figure has been removed! |

Продолжение таблицы 4

|  |  |
| --- | --- |
| 1 hexagon 21 41 6.4 2 | rhombus Center (x, y): (176.1, -34) (x1, y1 = (223.078, 0) (x2, y2 = (0, -26.9791) (x3, y3 = (129.122, 0) (x4, y4 = (0, -41.0209) Square: 659.664  pentagon Center (x, y): (-4, -71) (x1, y1) = (15.3098, -71) (x2, y2) = (1.967, -52.6353) (x3, y3) = (-19.6219, -59.65) (x4, y4) = (-19.6219, -82.35) (x5, y5) = (1.967, -89.3647) Square: 886.545  hexagon Center (x, y): (21, 41) (x1, y1) = (27.4, 41) (x2, y2) = (24.2, 46.5426) (x3, y3) = (17.8, 46.5426) (x4, y4) = (14.6, 41) (x5, y5) = (17.8, 35.4574) (x6, y6) = (24.2, 35.4574) Square: 106.41 |
| 3 | Total squre of all figures: 1652.63 |
| 5 | Program completed! |

# 5. Листинг программы

**CMakeLists.txt**

cmake\_minimum\_required (VERSION 3.2)

project (OOP)

add\_executable (oop\_exercise\_03 main.cpp menu.cpp functions.cpp figure.cpp

rhombus.cpp hexagon.cpp pentagon.cpp)

set\_target\_properties(oop\_exercise\_03 PROPERTIES CXX\_STANDART 14

CXX\_STANDART\_REQUIRED ON)

**figure.hpp**

#pragma once

#include <string>

class Figure {

protected:

    std::string Name;

public:

    Figure(const char \*figure);

    void PrintName();

    virtual void PrintCenter() = 0;

    virtual void PrintCoordinates() = 0;

    virtual float Square() = 0;

    virtual ~Figure();

};

**figure.cpp**

#include <iostream>

#include "figure.hpp"

Figure::Figure(const char \*figure): Name(figure) {}

void Figure::PrintName() {

    std::cout << Name;

}

Figure::~Figure() {}

**rhombus.hpp**

#pragma once

#include <iostream>

#include "figure.hpp"

class Rhombus: public Figure {

private:

    float Center\_x;

    float Center\_y;

    float SideLength;

    float Angle;

public:

    Rhombus();

    void GetCenter();

    void GetSide();

    void GetAngle();

    void PrintCenter() override;

    void PrintCoordinates() override;

    float Square() override;

    ~Rhombus();

};

**rhombus.cpp**

#include <iostream>

#include <cmath>

#include "rhombus.hpp"

#define PI 3.14159265

Rhombus::Rhombus(): Figure("rhombus"), Center\_x(0), Center\_y(0), SideLength(0), Angle(90) {}

void Rhombus::GetCenter() {

    std::cin >> Center\_x >> Center\_y;

}

void Rhombus::GetSide() {

    std::cin >> SideLength;

    while(SideLength < 0) {

        std::cout << "Entered incorrect size!\n";

        std::cout << "Pleale, re-enter side size: ";

        std::cin >> SideLength;

    }

}

void Rhombus::GetAngle() {

    while(1) {

        std::cin >> Angle;

        if(Angle >= 180 || Angle <= 0) {

            std::cout << "Entered invalid angle!" <<"\n";

            std::cout << "Please, re-enter the angle: ";

        }

        else {

            break;

        }

    }

}

void Rhombus::PrintCenter() {

    std::cout << "(" << Center\_x << ", " << Center\_y << ")\n";

}

void Rhombus::PrintCoordinates() {

    float angle = Angle / 2;

    float radius\_x = cos(angle \* PI / 180) \* SideLength;

    float radius\_y = sin(angle \* PI / 180) \* SideLength;

    for (int i = 0; i < 4; i++) {

        if(i % 2 == 0) {

            std::cout << "(x" << i + 1 << ", y)" << i + 1 << " = ";

            std::cout << "(" << Center\_x + radius\_x << ", " << 0 << ")\n";

            radius\_x \*= -1;

        }

        else {

            std::cout << "(x" << i + 1 << ", y)" << i + 1 << " = ";

            std::cout << "(" << 0 << ", " << Center\_y + radius\_y << ")\n";

            radius\_y \*= -1;

        }

    }

}

float Rhombus::Square() {

    return round(cos(Angle \* PI / 360) \* sin(Angle \* PI / 360) \* SideLength \*

SideLength \* 2 \* 10000) / 10000;

}

Rhombus::~Rhombus() {}

**pentagon.hpp**

pragma once

#include <iostream>

#include "figure.hpp"

class Pentagon: public Figure {

private:

    float Center\_x;

    float Center\_y;

    float SideLength;

public:

    Pentagon();

    Pentagon(float center\_x, float center\_y, float side);

    void GetCenter();

    void GetSide();

    void PrintCenter() override;

    void PrintCoordinates() override;

    float Square() override;

    ~Pentagon();

};

**pentagon.cpp**

#include <iostream>

#include <cmath>

#include "figure.hpp"

#include "pentagon.hpp"

#include <iomanip>

#define PI 3.14159265

Pentagon::Pentagon(): Center\_x(0), Center\_y(0), SideLength(0),

Figure("pentagon") {}

Pentagon::Pentagon(float center\_x, float center\_y, float side):

Figure("pentagon"), Center\_x(center\_x), Center\_y(center\_y), SideLength(side) {}

void Pentagon::GetCenter() {

    std::cin >> Center\_x >> Center\_y;

}

void Pentagon::GetSide() {

    std::cin >> SideLength;

    while(SideLength < 0) {

        std::cout << "Entered incorrect size!\n";

        std::cout << "Pleale, re-enter side size: ";

        std::cin >> SideLength;

    }

}

void Pentagon::PrintCenter() {

    std::cout << "(" << Center\_x << ", " << Center\_y << ")\n";

}

void Pentagon::PrintCoordinates() {

    float radius = SideLength / sqrt(2 \* (1 - cos(72 \* PI / 180))); // Radius to get coorditates

    for(int i = 0; i < 5; i++) {

        float x = Center\_x + cos(72 \* i \* PI / 180) \* radius;

        float y = Center\_y + sin(72 \* i \* PI / 180) \* radius;

        std::cout << "(x" << i + 1 << ", y" << i + 1 << ") = ";

        std::cout << "(" << round(x \* 10000) / 10000 << ", " << round(y \* 10000) / 10000 << ")\n";

    }

}

float Pentagon::Square() {

    float radius = SideLength / sqrt(2 \* (1 - cos(72 \* PI / 180))); // Radius to get square

    return radius \* radius \* sin(72 \* PI / 180) / 2 \* 5;

}

Pentagon::~Pentagon() {}

**hexagon.hpp**

#pragma once

#include <iostream>

#include "figure.hpp"

class Hexagon: public Figure { /\* Сторона шестиугольника равна расстоянию от его центра до вершины \*/

private:

    float Center\_x;

    float Center\_y;

    float SideLength;

public:

    Hexagon();

    Hexagon(float center\_x, float center\_y, float side);

    void GetSide();

    void GetCenter();

    void PrintCenter() override;

    void PrintCoordinates() override;

    float Square() override;

    ~Hexagon();

};

**hexagon.cpp**

#include <iostream>

#include <iomanip>

#include <cmath>

#include "hexagon.hpp"

#define PI 3.14159265

Hexagon::Hexagon(): Figure("hexagon"), Center\_x(0), Center\_y(0), SideLength(0) {}

Hexagon::Hexagon(float center\_x, float center\_y, float side): Figure("hexagon"), Center\_x(center\_x), Center\_y(center\_y), SideLength(side) {}

void Hexagon::GetSide() {

    std::cin >> SideLength;

    while(SideLength < 0) {

        std::cout << "Entered incorrect size!\n";

        std::cout << "Pleale, re-enter side size: ";

        std::cin >> SideLength;

    }

}

void Hexagon::GetCenter() {

    std::cin >> Center\_x >> Center\_y;

}

void Hexagon::PrintCenter() {

    std::cout << "(" << Center\_x << ", " << Center\_y << ")\n";

}

void Hexagon::PrintCoordinates() {

    for (int i = 0; i < 6; i++) {

        float x = Center\_x + cos(60 \* i \* PI / 180) \* SideLength;

        float y = Center\_y + sin(60 \* i \* PI / 180) \* SideLength;

        std::cout << "(x" << i + 1 << ", y" << i + 1 << ") = ";

        std::cout << "(" << round(x \* 10000) / 10000 << ", " << round(y \* 10000) / 10000 << ")\n";

    }

}

float Hexagon::Square() {

    return 3 \* sqrt(3) \* SideLength \* SideLength / 2;

}

Hexagon::~Hexagon() {}

**functions.hpp**

#pragma once

#include <string>

#include <vector>

#include "figure.hpp"

#include "rhombus.hpp"

#include "pentagon.hpp"

#include "hexagon.hpp"

Figure \* enter\_in\_figure(std::string figure); /\* Input function of all figure

Parameters \*/

Figure \* figure\_enter(); /\* Enter the figure name and input all figure

 parameters \*/

float square\_all\_figures(std::vector<Figure\*> vector); /\* Summ of squares of all figures \*/

void figure\_delete(std::vector<Figure\*> &vector, int index); /\* Delete figure in vector \*/

**functions.cpp**

#include <iostream>

#include "functions.hpp"

// Input function of all figure parameters

Figure \*enter\_in\_figure(std::string figure) {

    Figure \*ptr\_figure = nullptr;

    if (figure == "rhombus") {

        Rhombus \*ptr\_rhombus = new Rhombus();

        std::cout << "Write center coordinates: ";

        ptr\_rhombus->GetCenter();

        std::cout << "Write side size: ";

        ptr\_rhombus->GetSide();

        std::cout << "Write an angle: ";

        ptr\_rhombus->GetAngle();

        ptr\_figure = (Figure\*)ptr\_rhombus;

    }

    else if (figure == "pentagon") {

        Pentagon \*ptr\_pentagon = new Pentagon();

        std::cout << "Write center coordinates: ";

        ptr\_pentagon->GetCenter();

        std::cout << "Write side size: ";

        ptr\_pentagon->GetSide();

        ptr\_figure = (Figure\*)ptr\_pentagon;

    }

    else if (figure == "hexagon") {

        Hexagon \*ptr\_hexagon = new Hexagon();

        std::cout << "Write center coordinates: ";

        ptr\_hexagon->GetCenter();

        std::cout << "Write side size: ";

        ptr\_hexagon->GetSide();

        ptr\_figure = (Figure\*)ptr\_hexagon;

    }

    return ptr\_figure;

}

// Enter the figure name and input all figure parameters

Figure \*figure\_enter() {

    std::string figure\_name;

    std::cout << "Enter a name of figure (Possible names: rhombus, pentagon,

hexagon): ";

    std::cin >> figure\_name;

    while (figure\_name != "rhombus" && figure\_name != "pentagon" &&

figure\_name != "hexagon") {

        std::cout << "Entered invalid figure name\n";

        std::cout << "Please, enter figure name: ";

        std::cin >> figure\_name;

    }

    return enter\_in\_figure(figure\_name);

}

// Summ of squares of all figures

float square\_all\_figures(std::vector<Figure\*> vector) {

    float square = 0;

    for (int i = 0; i < vector.size(); i++) {

        square += vector[i]->Square();

    }

    return square;

}

// Delete figure in vector

void figure\_delete(std::vector<Figure\*> &vector, int index) {

    Figure \*ptr = vector[index];

    vector.erase(vector.begin() + index);

    delete ptr;

}

**menu.hpp**

#pragma once

int menu();

**menu.cpp**

#include <iostream>

#include <string>

#include <vector>

#include "functions.hpp"

#include "figure.hpp"

#include "menu.hpp"

int menu() {

    std::vector<Figure\*> figures;

    while(1) {

        int value;

        std::cout << "Menu:\n";

        std::cout << "1 - Enter the new figure\n";

        std::cout << "2 - Print all figures and their parameters\n";

        std::cout << "3 - Print total square of all figures\n";

        std::cout << "4 - Delete figure from array\n";

        std::cout << "5 - Exit the program\n";

        std::cin >> value;

        if (value == 1) {

            figures.push\_back(figure\_enter());

        }

        else if (value == 2) {

            for(int i = 0; i < figures.size(); i++) {

                figures[i]->PrintName();

                std::cout << "\n";

                std::cout << "Center (x, y): ";

                figures[i]->PrintCenter();

                figures[i]->PrintCoordinates();

                std::cout << "Square: " << figures[i]->Square() << "\n\n";

            }

        }

        else if (value == 3) {

            std::cout << "Total squre of all figures: " << square\_all\_figures(figures) << "\n\n";

        }

        else if (value == 4) {

            int index;

            std::cout << "Write an index of figure: ";

            std::cin >> index;

            while(index < 0 || index > figures.size() - 1) {

                std::cout << "The index is entered incorrectly\n";

                std::cout << "Please, write an index: ";

                std::cin >> index;

            }

            figure\_delete(figures, index);

            std::cout << "The figure has been removed!\n";

        }

        else if (value == 5) {

            for (int i = 0; i < figures.size(); i++) {

                delete figures[i];

            }

            figures.clear();

            std::cout << "Program completed!";

            return 0;

        }

        else {

            std::cout << "The entered option does not exist\n";

            std::cout << "\n";

            continue;

        }

    }

    return 0;

}

**main.cpp**

#include <iostream>

#include "menu.hpp"

int main() {

    menu();

}

Ссылка на github репозиторий: <https://github.com/HarryLiker/oop_exercise_03>

# 6. Выводы

Была написана программа реализации класса Figure и классов Rhombus, Pentagon, Hexagon, которые от него наследуются. Программа может находить и выводить геометрический центр, координаты вершин, площадь фигуры. Также программа может выводить все фигуры из динамического массива, находить общую площадь фигур и удалять фигуру из массива по заданному индексу.

Были получены знания о наследовании – свойстве, позволяющем создавать классы-потомки на основе существующего класса, при котором характеристики класса-родителя присваиваются классу-потомку. Были получены навыки работы с виртуальными функциями и их переопределением в классах-потомках.

# Список используемых источников

1. Р. Лафоре, В.Е. Зайцев «Объектно-ориентированное программирование в C++» - Москва: «Питер», 2004.

2. Информация о наследовании в C++ - [Электронный ресурс]. - URL: <https://ravesli.com/urok-154-bazovoe-nasledovanie-v-c/> (дата обращения: 20.10.2020).

3. Информация о наследовании в C++ - [Электронный ресурс]. - URL: <https://habr.com/ru/post/445948/> (дата обращения: 22.10.2020).

4. Информация о виртуальных функциях - [Электронный ресурс]. - URL: <https://ravesli.com/urok-163-virtualnye-funktsii-i-polimorfizm/> (дата обращения: 22.10.2020).

5. Информация о переопределении методов - [Электронный ресурс]. - URL: <https://ravesli.com/urok-159-vyzov-i-pereopredelenie-metodov-roditelskogo-klassa/>

(дата обращения: 24.10.2020).