Министерство образования и науки Российской Федерации

Федеральное государственное автономное

образовательное учреждение высшего образования

«Национальный исследовательский нижегородский

государственный университет им. Н.И. Лобачевского»

Институт Информационных Технологий Математики и Механики

**Отчет по практическим работам**

**за второй семестр**

**Выполнил**:

студент группы 381703-2

Аникин Сергей Алексеевич

**Проверил**:

доцент каф. МОСТ

Сысоев А.В.

Нижний Новгород, 2018

**Оглавление**

|  |  |
| --- | --- |
| Введение | 3 |
| Лабораторная работа №1 |  |
| Постановка задачи | 4 |
| Руководство пользователя | 4 |
| Руководство программиста | 5 |
| Лабораторная работа №2 |  |
| Постановка задачи | 6 |
| Руководство пользователя | 6 |
| Руководство программиста | 9 |
| Лабораторная работа №3 |  |
| Постановка задачи | 11 |
| Руководство пользователя | 11 |
| Руководство программиста | 15 |
| Лабораторная работа №4 |  |
| Постановка задачи | 17 |
| Руководство пользователя | 17 |
| Руководство программиста | 18 |
| Лабораторная работа №6 |  |
| Постановка задачи | 20 |
| Руководство пользователя | 20 |
| Руководство программиста | 23 |
| Заключение | 25 |
| Приложение | 26 |

**Введение**

Данная практическая работа выполнялась с целью научиться делать всевозможные программы, выполняющие задачи различного рода, размера и сложности.

**Лабораторная работа №1**

**Постановка задачи**

* Разработать класс **Рациональная дробь**.
* Класс должен хранить корректные дроби (знаменатель не равен 0) и выполнять с ними 4 стандартные арифметические операции. Дробь должна храниться в несократимом виде.
* Класс должен содержать все необходимые конструкторы, оператор присваивания, а также «уметь» выводить себя на консоль.

**Руководство пользователя**

При запуске программы от пользователя запросят ввести числитель и знаменатель двух дробей, с которыми будут производиться операции (рис.1).

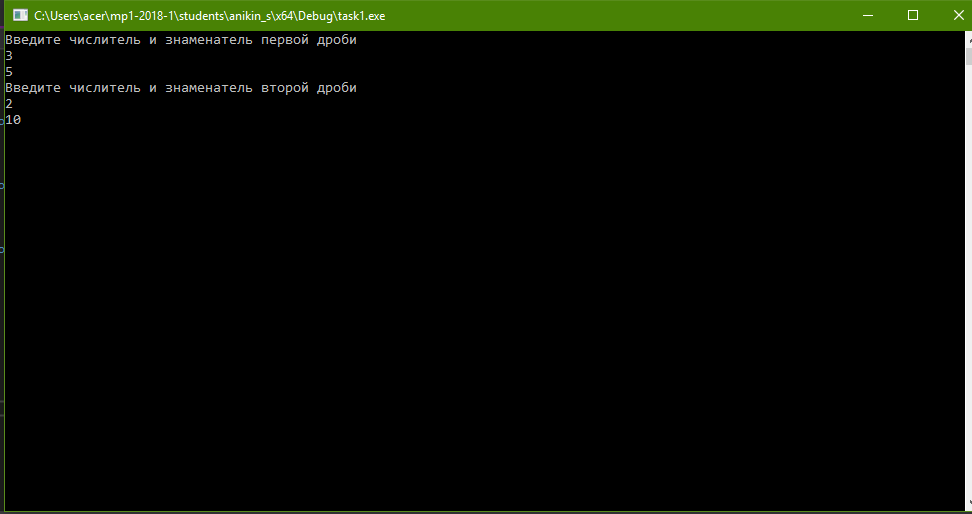


Рисунок 1: Ввод данных

Далее программа выводит результаты 4 основных арифметических операций: сложение, разность, умножение и деление соответственно (рис.2).

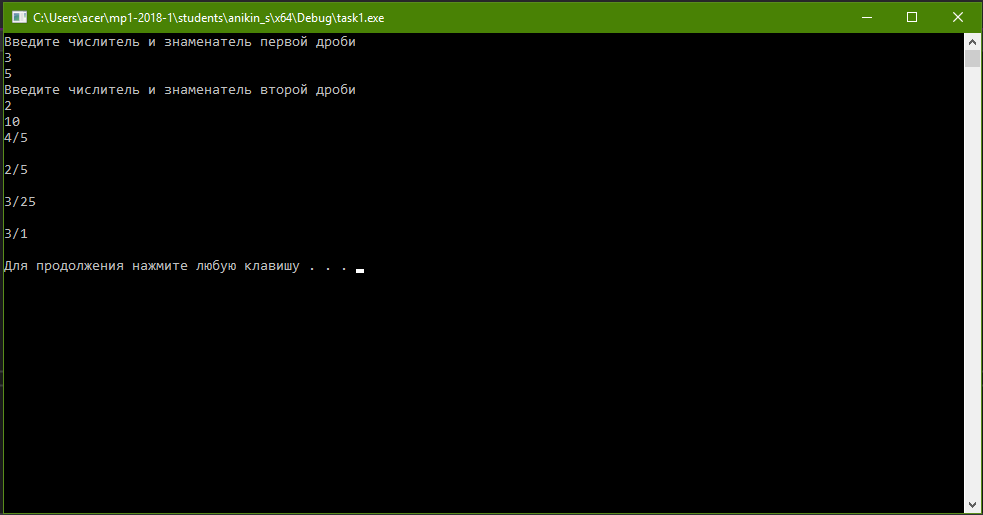


Рисунок 2: Результат (эксперимент)

**Руководство программиста**

Class TFraction

Поля:

1. int num – числитель
2. int denom – знаменатель
3. bool exist – существование дроби

Методы:

1. int nod – метод вычисления наибольшего общего делителя числителя и знаменателя
2. void total – приведение дроби к несократимому виду
3. TFraction (int x, int y) – конструктор
4. bool IsExist () – проверка на существование дроби
5. void Print () – вывод на консоль
6. TFraction operator=(const TFraction& s) – оператор присваивания
7. TFraction operator+(const TFraction& s) – оператор сложения
8. TFraction operator-(const TFraction& s) – оператор вычитания
9. TFraction operator\*(const TFraction& s) – оператор умножения
10. TFraction operator / (const TFraction& s) – оператор деления

**Лабораторная работа №2**

**Постановка задачи:**

* Разработать класс **Полином**.
* Класс должен хранить полином (многочлен) от одной переменной (х). Степень полинома n находится в диапазоне от 0 до 12.
* Класс должен предоставлять следующие операции: 1) задать степень полинома, 2) задать коэффициенты мономов полинома, 3) узнать степень полинома, 4) узнать значение коэффициента по его номеру, 5) вычислить значение полинома в заданной точке х, 6) найти производную полинома.
* Класс должен содержать все необходимые конструкторы, деструктор, оператор присваивания, а также «уметь» выводить себя на консоль.

**Руководство пользователя**

В самом начале программа запросит вас ввести степень полинома и соответствующие коэффициенты (рис.1).

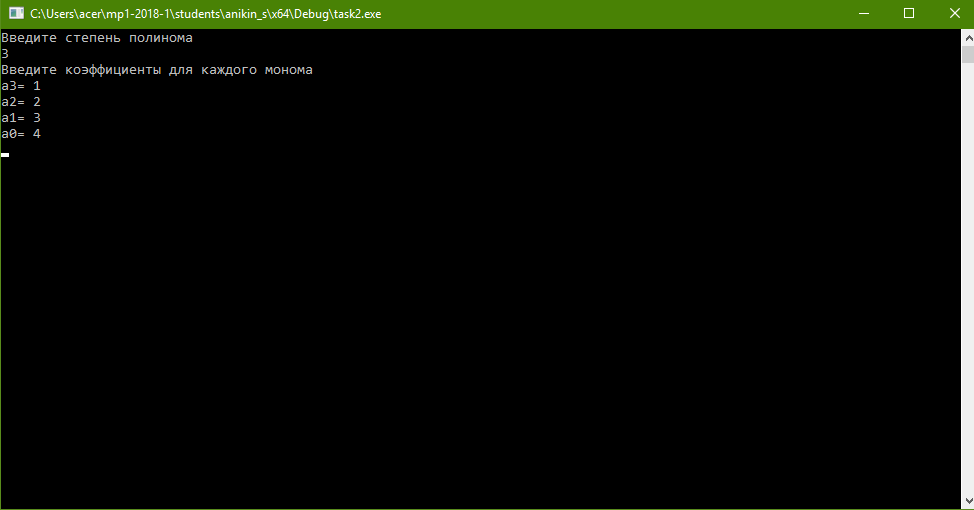


Рисунок 1: Ввод полинома

Затем вам дается возможность выбрать одну из пяти операций (рис.2):

1. Вывести полином на экран.
2. Вывести степень полинома на экран.
3. Узнать коэффициент по его порядковому номеру.
4. Вычислить значение полинома в заданной точке Х.
5. Найти и вывести на экран производную полинома.

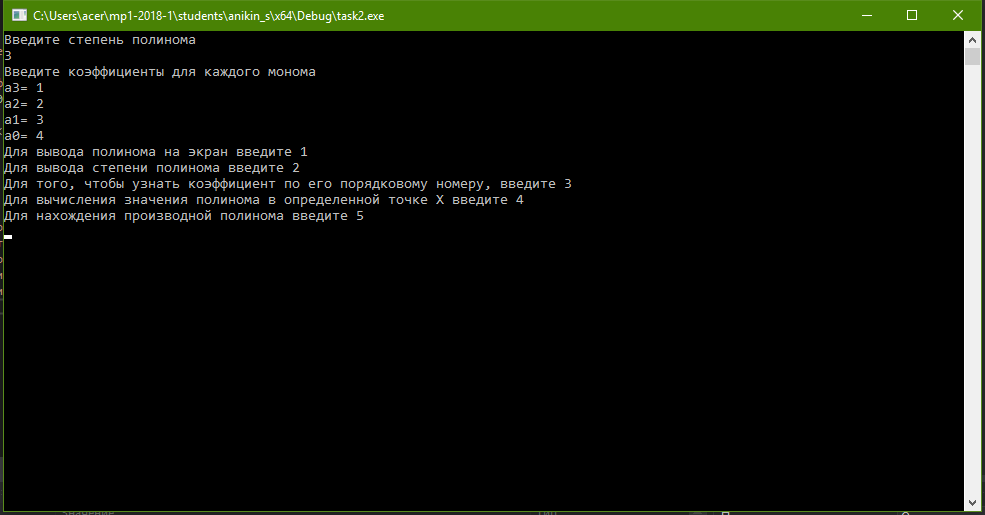


Рисунок 2: Выбор операции

При вводе «1» на экран консоли выводится ваш введенный полином (рис.3). Значение «x^0» считать за 1.

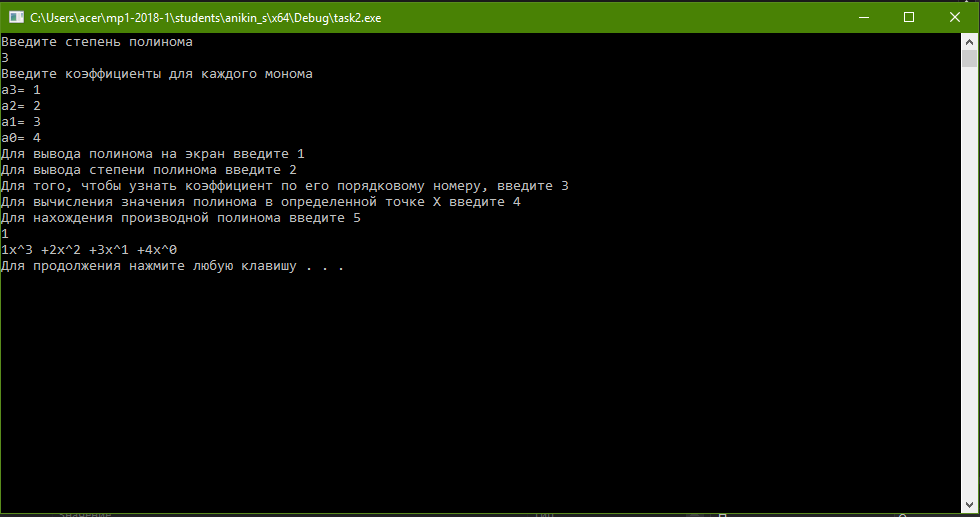


Рисунок 3: Вывод полинома

При вводе «2» на экран выводится степень полинома (рис.4).

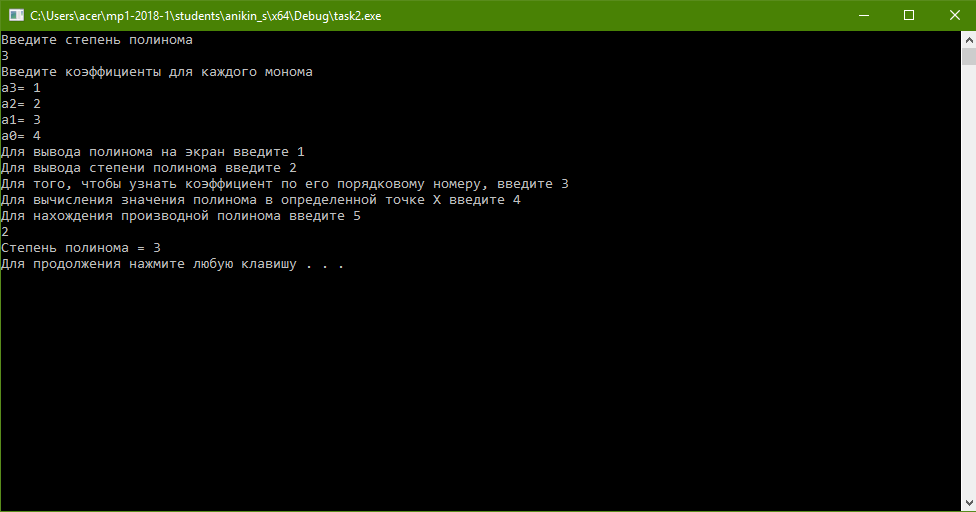


Рисунок 4: Вывод степени полинома

При вводе «3» вас спрашивают порядковый номер коэффициента. После ввода номера вам выводится на экран значение выбранного коэффициента (рис.5).

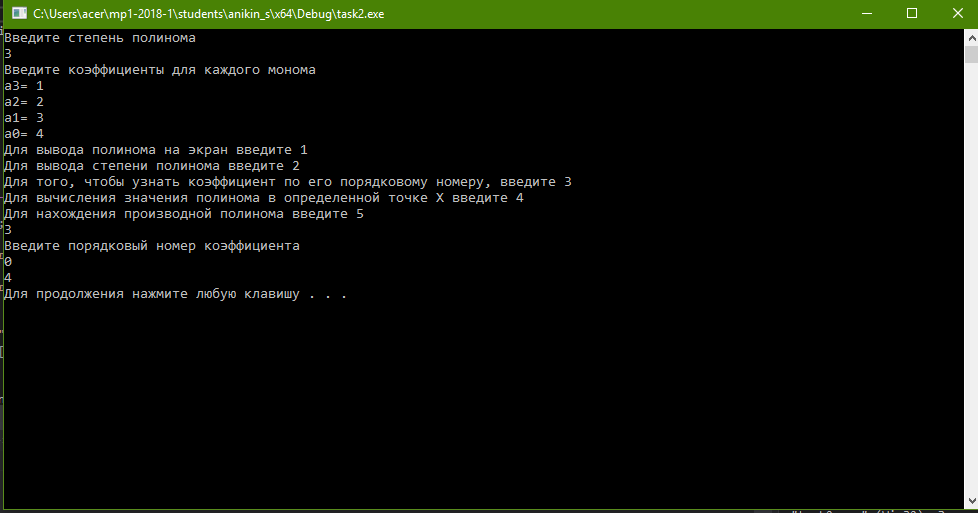


Рисунок 5: Нахождение коэффициента по его номеру

При вводе «4» вас спрашивают значение точки «Х», в которой требуется посчитать полином. После ввода значения точки «Х» на экран выводится значение полинома в этой точке (рис.6).

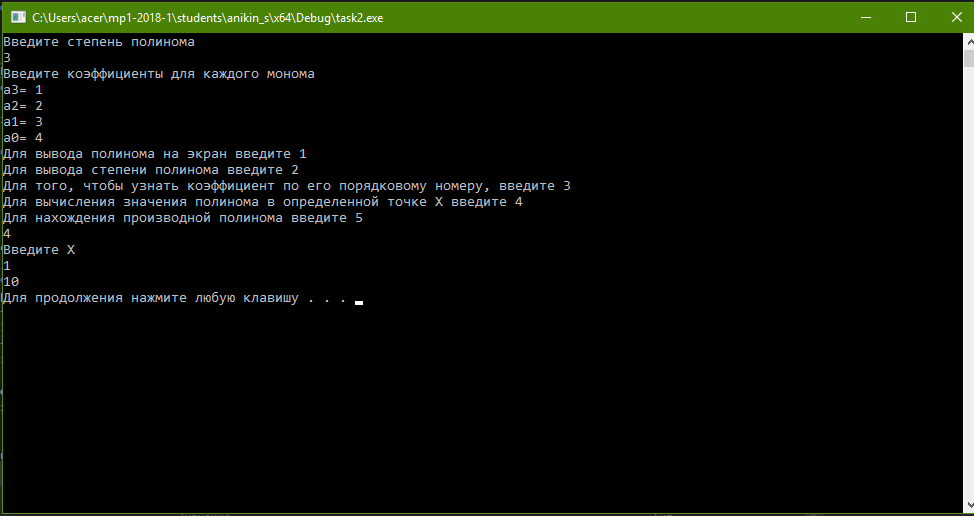


Рисунок 6: Подсчет значения в точке

При вводе «5» на экран консоли выводится полином, который является производной заданного вами полинома (рис.7).

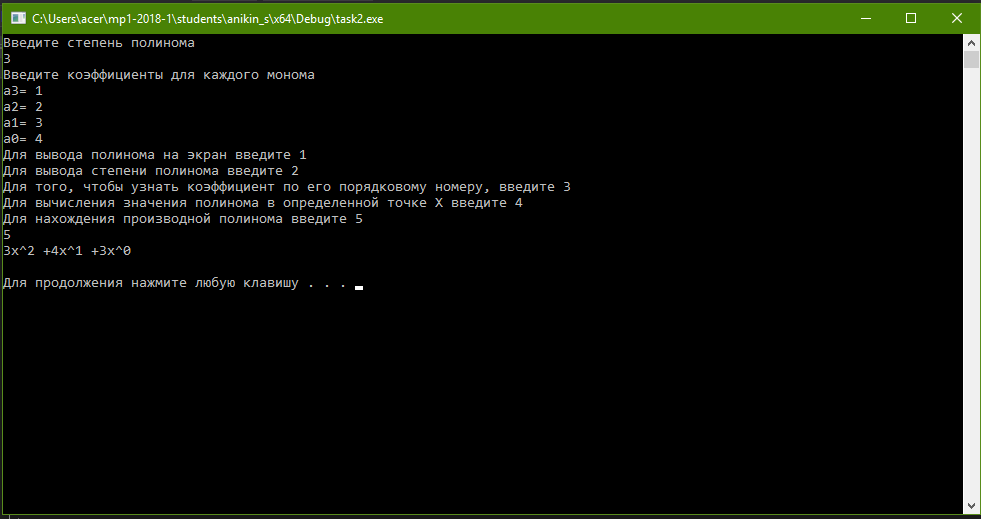


Рисунок 7: Производная

**Руководство программиста**

Class TPolinom

Поля:

1. int n – степень полинома
2. double \*a – массив коэффициентов

Методы:

1. TPolinom(int \_n, double \*\_a) – конструктор
2. TPolinom operator=(const TPolinom&s) – оператор присваивания
3. TPolinom(const TPolinom&s) – конструктор копирования
4. void Out() – вывод полинома на консоль
5. int Out\_n() – вывод степени полинома на консоль
6. double Find\_a(int i) – метод нахождения коэффициента по его порядковому номеру
7. double Calc(int x) – метод подсчета полинома в заданной точке «х»
8. TPolinom Diff() – метод вычисления производной полинома
9. ~TPolinom() – деструктор (удаляет выделенную под массив коэффициентов память)

**Лабораторная работа №3**

**Постановка задачи:**

* Разработать класс **Расчет интегралов**.
* Класс должен позволять вычислять приближенное значение интеграла от произвольной функции одной переменной, заданной в виде функции языка C++. Интеграл необходимо вычислять в заданных пределах интегрирования, используя методы левых, правых и средних прямоугольников.
* Класс должен содержать необходимые служебные методы (конструкторы, деструктор и пр.).
* Класс должен предоставлять следующие операции: 1) задать текущую функцию, 2) задать пределы интегрирования, 3) узнать пределы интегрирования, 4) задать число отрезков метода прямоугольников, 5) выбрать метод вычисления, 6) вычислить значение интеграла выбранным методом, 7) вывести результат вычисления на экран.

**Руководство пользователя**

При запуске программы вас попросят ввести пределы интегрирования функции. Сначала вы должны ввести значение левого предела, а затем значение правого предела. Вводить значения необходимо строго в таком порядке для отсутствия логических ошибок и неправильного подсчета дальнейших значений. Затем вы должны выбрать, по какой функции будут производиться вычисления (рис. 1).

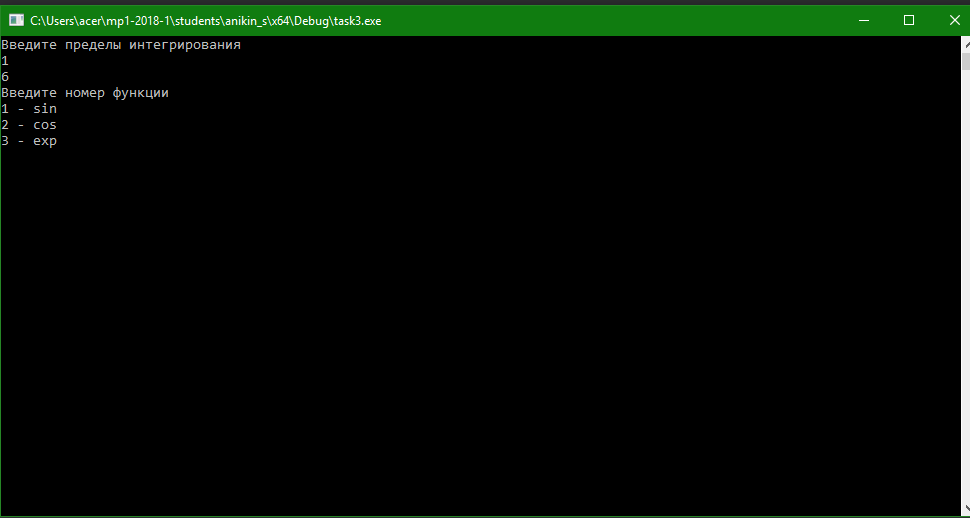


Рисунок 1: начальный экран

Далее вам представится возможность выбрать одну из 7 возможных операций: подсчет интеграла методом левых, правых, средних прямоугольников, задать новые пределы и функцию и вывести на экран пределы и функцию. Для выбора операции необходимо ввести ее номер (рис.2). Для выхода из программы введите 0.

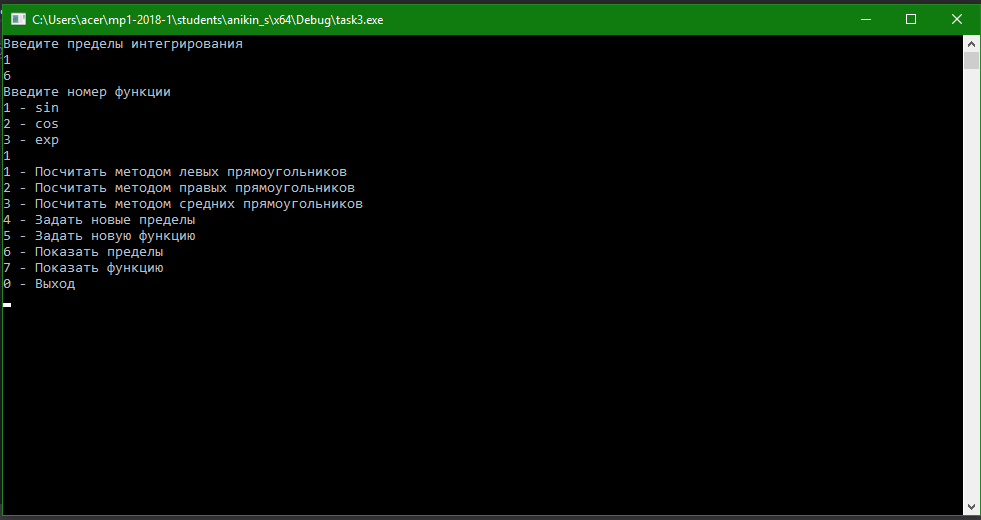


Рисунок 2: меню выбора

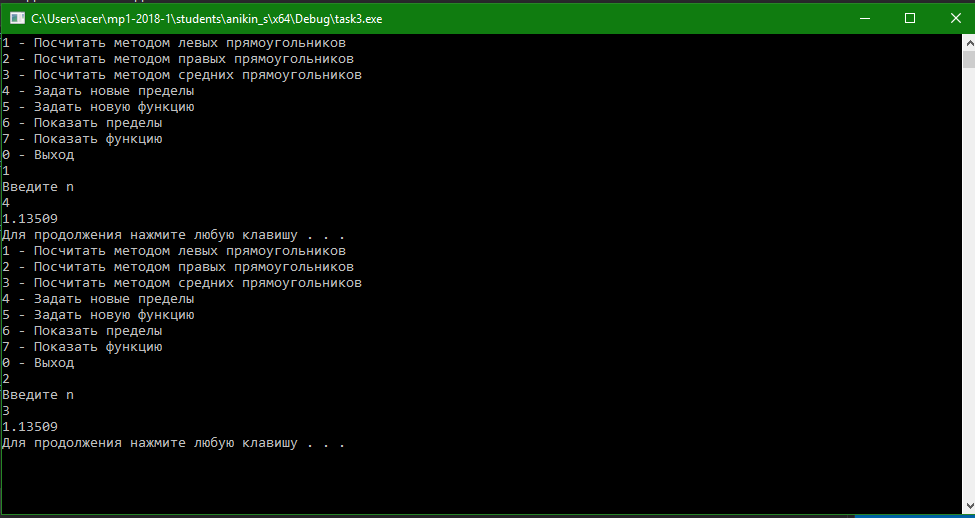


Рисунок 3: метод левых и правых прямоугольников

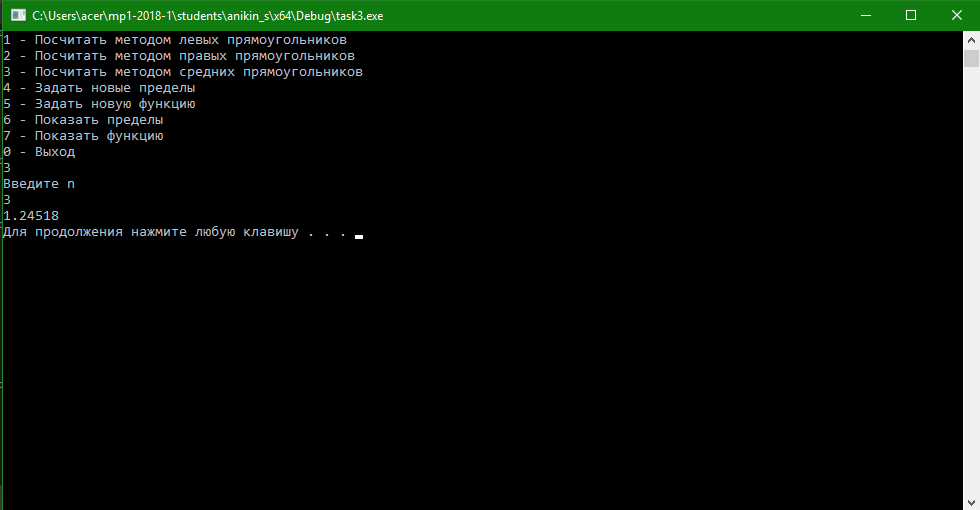


Рисунок 4: метод средних прямоугольников

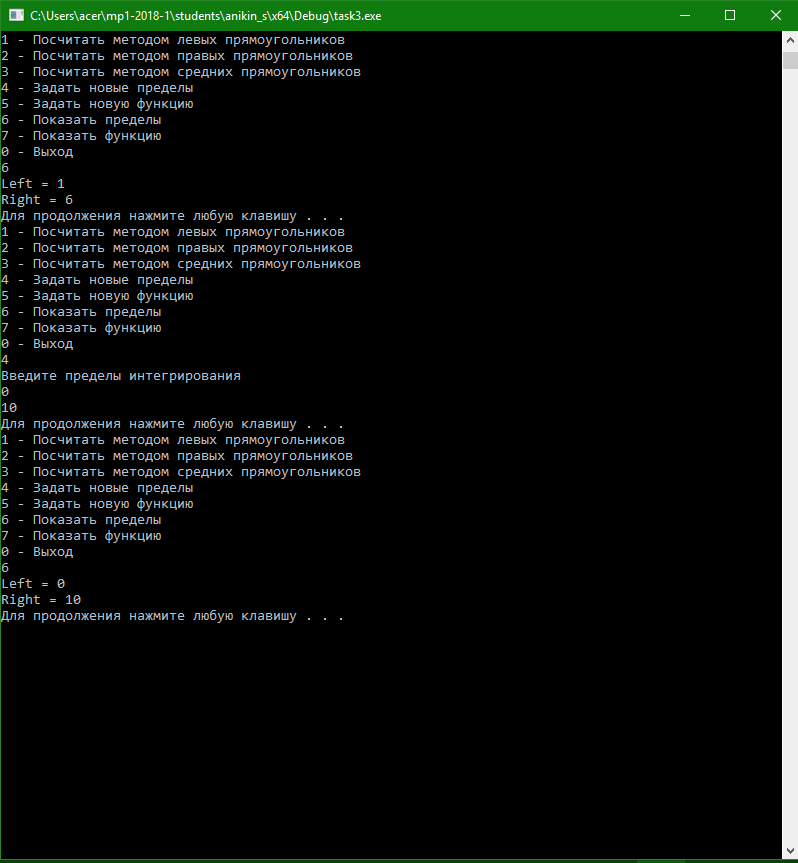


Рисунок 5: задание и вывод пределов интегрирования

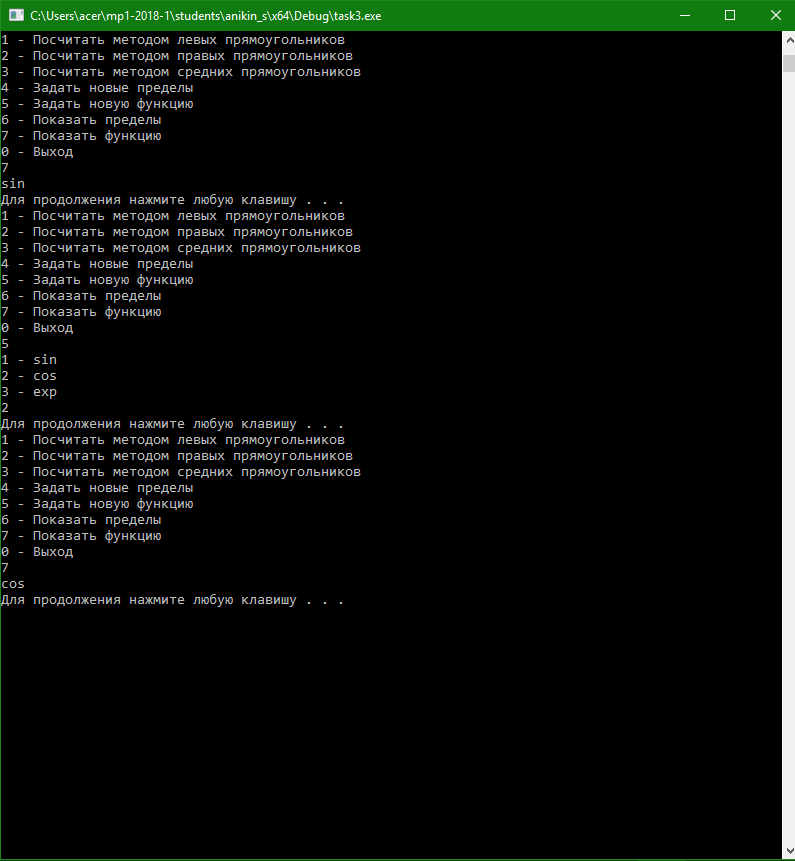


Рисунок 6: задание и вывод функции

**Руководство программиста**

double f1(double x) – возвращает sin(x)

double f2(double x) – возвращает cos (x)

double f3(double x) – возвращает exp(x)

class Integral

Поля:

1. int left – левый предел
2. int right – правый предел
3. double total – результат подсчета
4. double(\*F)(double x) - указатель на тип функции

Методы:

1. Integral(int a=0, int b=0, int f=0) – конструктор
2. Integral operator=(const Integral&s) – оператор присваивания
3. void SetBounds(int a, int b) – задать пределы интегрирования
4. void SetFunction(double(\*FF)(double x)) – задать функцию
5. double GetLeftBound() – вывести на консоль левый предел
6. double GetRightBound() – вывести на консоль правый предел
7. double CalcLeft(int n) – подсчитать методом левых прямоугольников
8. double CalcMid(int n) – подсчитать методом средних прямоугольников
9. double CalcRight(int n) – подсчитать методом правых прямоугольников
10. void Show() – вывести на экран результат подсчета

**Лабораторная работа №4**

**Постановка задачи**

* Разработать класс **Термометр**.
* Класс должен хранить историю наблюдений за температурой в течение одного календарного (не високосного) года. Наблюдения описываются датой (день, месяц, год) и временем (в часах). При поступлении нового наблюдения для уже существующих даты и времени старое наблюдение заменяется. Температура задается в градусах Цельсия.
* Класс должен содержать необходимые служебные методы.
* Класс должен предоставлять следующие операции: 1) установить начальные дату и время наблюдений, 2) узнать начальные дату и время наблюдений, 3) задать наблюдение, 4) узнать температуру в выбранном наблюдении, 5) задать серию наблюдений для выбранной даты, 6) найти среднюю температуру для выбранной даты, выбранного месяца, за всю историю наблюдений, 7) найти среднюю дневную или ночную температуру для выбранного месяца, 8) сохранить историю наблюдений в файл, 9) считать историю наблюдений из файла.

**Руководство пользователя**

В начале программы вас попросят ввести год, месяц, день, время (только час) и температуру (в градусах Цельсия), для того, чтобы можно было хоть с чем – то поработать (рис.1).

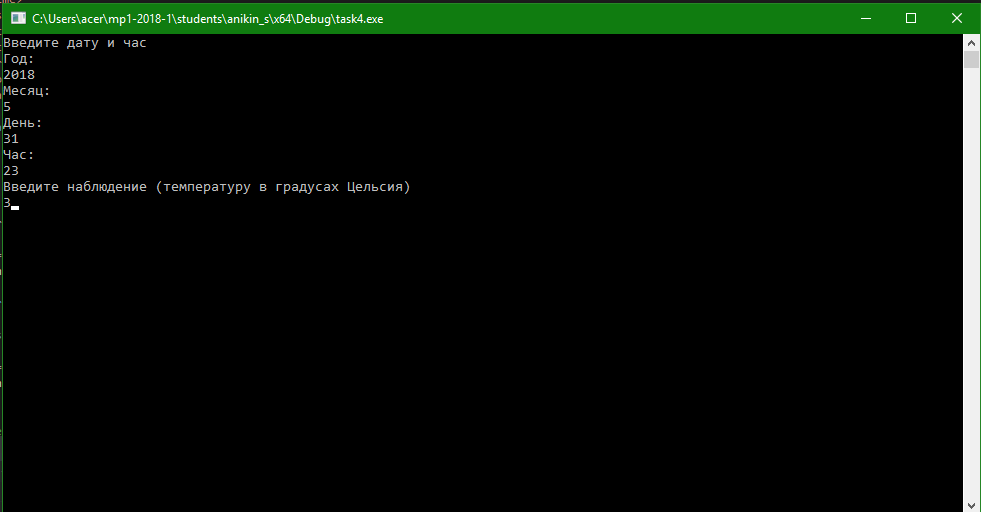


Рисунок 1: ввод данных

Затем вас переносят в меню операций, где вы можете: узнать последнюю выбранную дату, узнать температуру в выбранном наблюдении, установить серию наблюдений, найти среднюю температуру в выбранный день, месяц, год, среднюю дневную и ночную температуру, сохранить данные в текстовый файл и прочитать из файла (рис.2).

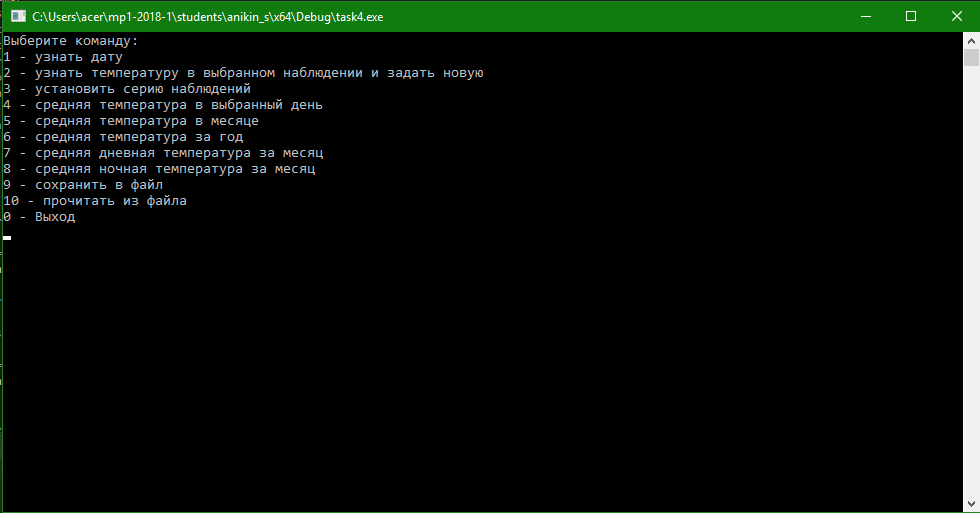


Рисунок 2: меню выбора

Внимание(!!!) : данная программа является не полностью логически сформированной, так как:

1. Наблюдения могут быть заданы не для всех дат и часов, поэтому выделять для них место заранее не выгодно.
2. Задание серии наблюдений сделано неверно. Температура в каждом наблюдении должна быть своя.
3. Вычисление средней температуры также сделано неверно. В подсчет берутся и те даты, для которых наблюдение не задано (ноль градусов).

**Руководство программиста**

struct Date

поля:

* int day, month, year – день, месяц, год

методы:

* Date(int dd = 0, int mm = 0, int yy = 0) : day(dd), month(mm), year(yy) – конструктор
* int GetDay() – возвращает значение дня
* int GetMonth() – возвращает значение месяца
* int GetYear() – возвращает значение года

struct Time

поля:

* int hour – час

методы:

* Time(int hh = 0) : hour(hh) – конструктор
* int GetHour() – возвращает час

class Thermometer

поля:

1. int degree – температура в Цельсиях
2. Date d – дата
3. Time t – время
4. vector<vector<int>> year365 – массив дней
5. double mid – среднее значение (дня, месяца, года, дневное и ночное за месяц)

методы:

1. Thermometer(Date dd, Time tt, int deg = 0) : d(dd), t(tt) – конструктор
2. Thermometer operator=(const Thermometer&s) – оператор присваивания
3. void SetDegree(Date dd, Time tt, int deg) – занести наблюдение
4. int GetDegree(Date dd, Time tt) – вернуть наблюдение (в выбранном дне) на консоль
5. int GetDayT(Date d) – вернуть день
6. int GetMonthT(Date d) – вернуть месяц
7. int GetYearT(Date d) – вернуть год
8. int GetHourT(Time t) – вернуть час
9. int CodeNumberForDay(Date dd) – возвращает кодовое число дня (день + месяц\*100)
10. int FirstCodeMonth(int mm) – возвращает кодовое число первого дня месяца
11. int LastCodeMonth(int mm) – возвращает кодовое число последнего дня месяца
12. void SetSeriesOfDegrees(Date dd, int i, int deg) – задать серию наблюдений
13. double GetMiddleDate(Date dd) – средняя в дне
14. double GetMiddleMonth(int mm) – средняя в месяце
15. double GetMiddle() – средняя в году
16. double GetMiddleDay(int mm) – средняя дневная в месяце
17. double GetMiddleNight(int mm) – средняя ночная в месяце
18. void Save() – сохранить в файл
19. void Read() – считать из файла
20. ~Thermometer() – деструктор для массива дней

**Лабораторная работа №6**

**Постановка задачи**

Разработать систему классов и реализовать с ее помощью игру **Змейка**.

Требования (правила).

* Играет один игрок (человек), управляющий «змейкой».
* Игра идет на прямоугольном поле N x M клеток. Поле ограничено «стенами» так что вместе со стенами размер поля – (N + 2) x (M + 2) клеток.
* При старте игры змейка имеет длину 5 клеток, форму в виде горизонтального отрезка и располагается в произвольном месте поля, не пересекая и не касаясь стен.
* При старте игры «голова» змейки располагается слева, «хвост» справа. Голова змейки окрашена в цвет, отличный от цвета остальных клеток ее тела.
* При старте игры в произвольной клетке поля (не совпадающей с клетками, занятыми змейкой) возникает «пища».
* При запуске игрового процесса (по специальной команде или автоматически при старте игры) змейка начинает автоматическое движение влево с некоторой заданной скоростью.
* Движение заключается в том, что за каждый такт голова змейки перемещается на одну клетку в текущем направлении движения, а клетка, в которой располагался хвост, становится пустой.
* Игрок может сменить направление движения змейки с помощью клавиш-«стрелок» (вверх, вниз, влево, вправо).
* Если на текущем такте движения голова змейки должна будет занять клетку стены или клетку, которая уже занята любой из клеток ее тела, игра прекращается и считается проигранной.
* Задача игрока вырастить змейку до заданного при старте игрового процесса размера. Змейка вырастает в длину на одну клетку (с хвоста) при каждом поглощении пищи, т.е. в тот момент, когда ее голова на очередном такте движения занимает клетку, в которой расположена пища. На этом же такте в произвольном месте игрового поля (не совпадающей с клетками, занятыми змейкой) снова появляется пища.
* Если змейка выросла до заданной при старте игрового процесса длины, игра считается выигранной.

**Руководство пользователя**

В начале работы вас требуют ввести размеры поля, на котором вы будете играть (рис.1). Рекомендуется ставить ширину (50) и высоту (20) или ширину (60) и высоту (25), или другие примерные значения, чтобы примерно получался квадрат. Чем больше размеры карты, тем медленнее будет двигаться змейка.



Рисунок 1: размеры карты

А далее вы просто играете в змейку (wasd – управление)… Что? На этом все. Или вы что – то еще ожидали? Нет, сэр (или мэм). На этом все. Приятной игры :)

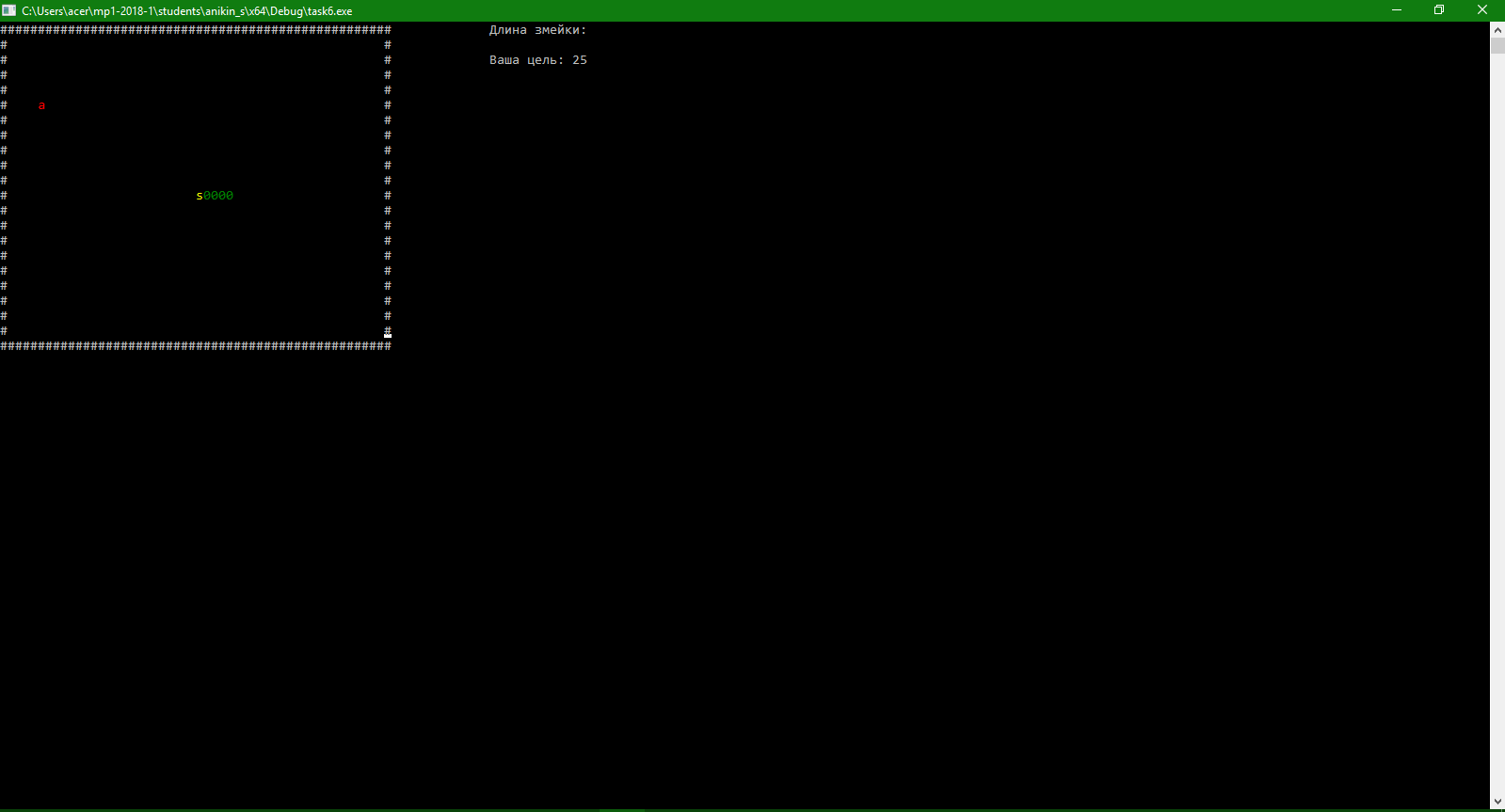


Рисунок 2: игра «Змейка»

…Ну ладно. Так уж и быть. Правила пересказывать не буду, не маленький уже. Просто покажу, что условия победы и поражения работают исправно. Ах да, еще кое – что: с начала игры не показывается длина змейки (баг системы, простите), но после первого яблока все снова становится в порядке. Начальная же длина змейка равна 5 клеткам (вместе с головой).

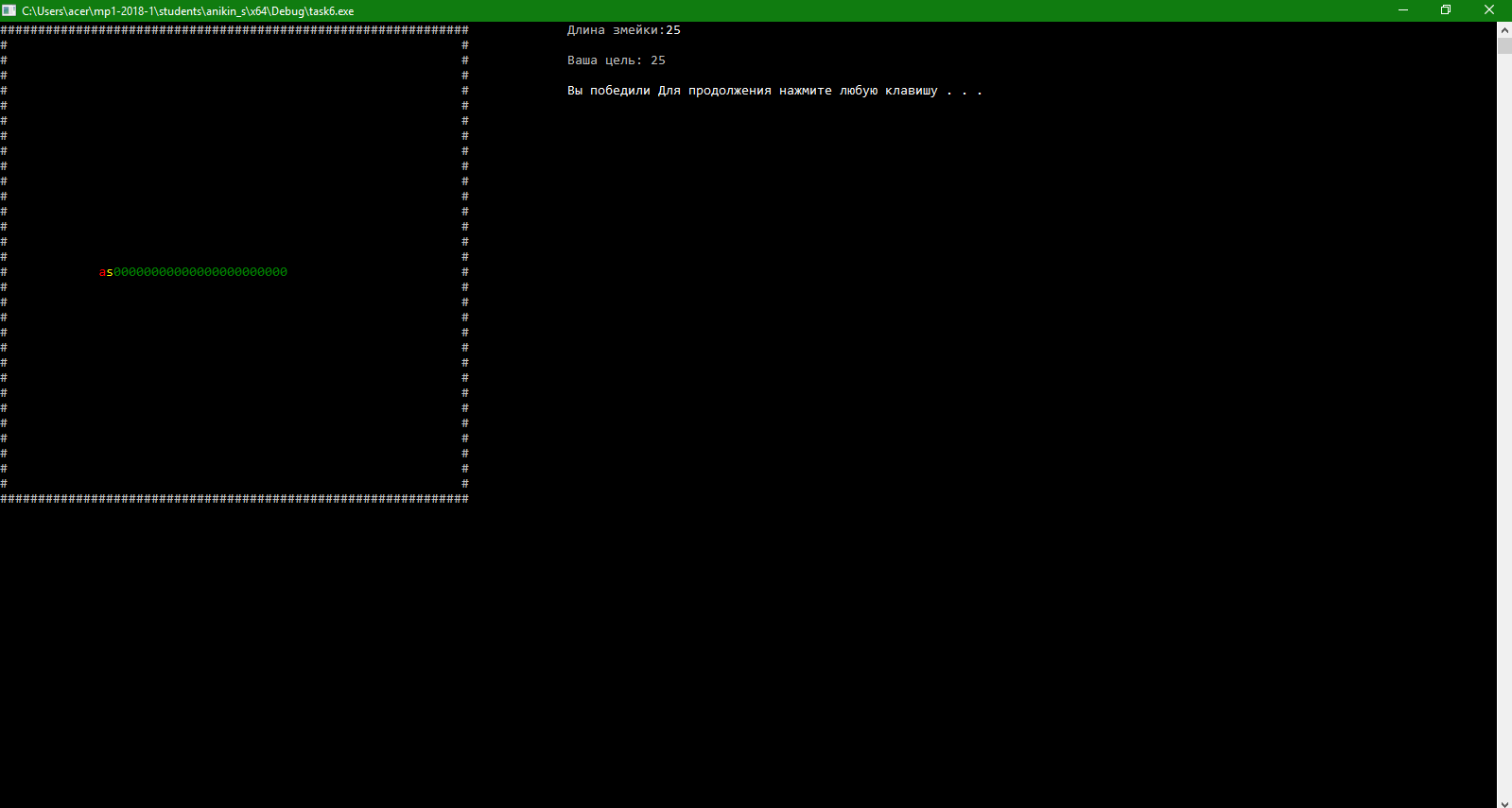


Рисунок 3: победа

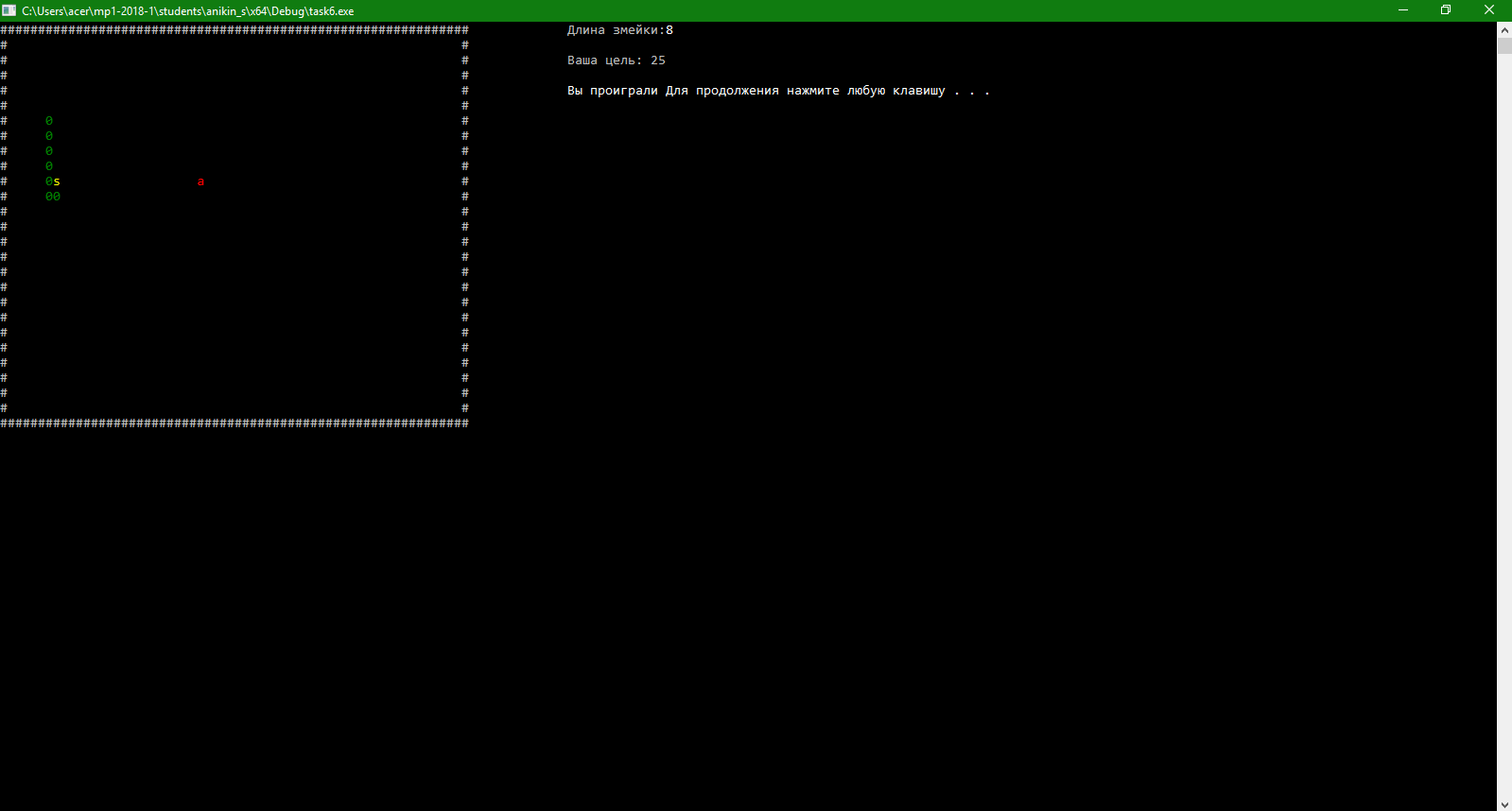


Рисунок 4: поражение из – за столкновения с самим собой

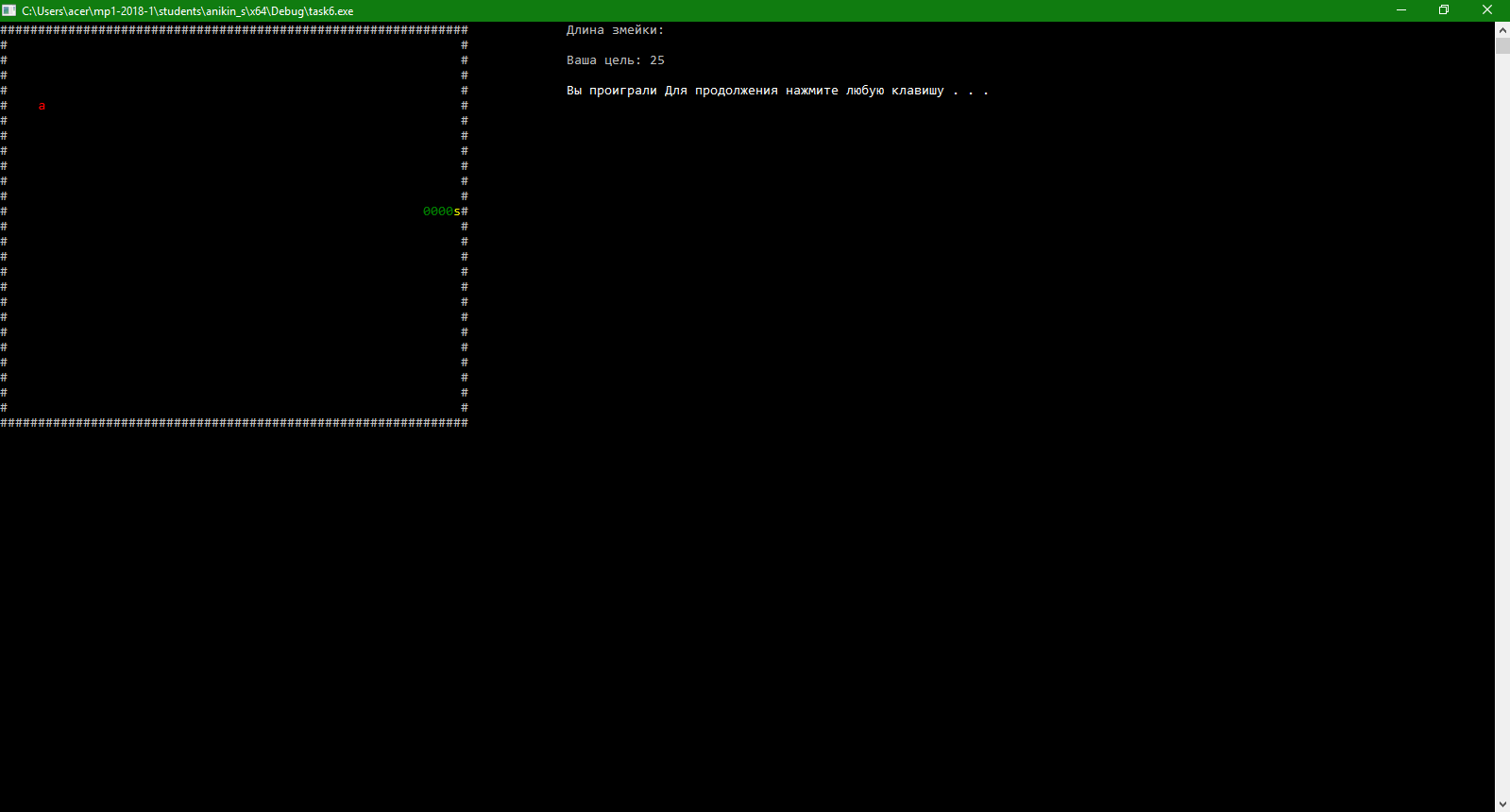


Рисунок 5: поражение из – за столкновения со стенкой

**Руководство программиста**

struct Snake

поля:

* vector<int> x – координаты х каждой ячейки тела
* vector<int> y – координаты у каждой ячейки тела
* int headX, headY – координаты головы х, у
* int length – длина змейки

методы:

* Snake(int \_length = 5, int \_headX = 0, int \_headY = 0) : length(\_length), headX(\_headX), headY(\_headY) – конструктор

struct Fruit

поля:

* int fruitX, fruitY – координаты фрукта х, у

методы:

* Fruit(int \_fruitX = 5, int \_fruitY = 5) : fruitX(\_fruitX), fruitY(\_fruitY) – конструктор
* void CreateFruit(int \_width, int \_height) – создание фрукта на карте

struct Map

поля:

* int height, width – высота и ширина карты
* Snake Snake – змея
* Fruit Fruit – фрукт

методы:

* Map(int \_height = 10, int \_width = 10) : height(\_height), width(\_width) – конструктор
* void gotoxy(int x, int y) – перемещение курсора
* void CreateSnake() – задает расположение змеи на карте
* void PaintMap() – рисование карты

class TheGame

поля:

1. int Direction – направление движения
2. int Purpose – цель
3. enum Directon { LEFT, RIGHT, UP, DOWN } – перечисление для направления
4. Directon dir – направление
5. bool GameOver – конец игры / поражение
6. bool GameVictory - победа
7. Map Map – карта со змеей и фруктом

методы:

1. TheGame(bool \_GameOver = false, bool \_GameVictory = false) – конструктор
2. void gotoxy(int x, int y) – перемещение курсора
3. void Input() – считывание нажатий на клавиатуру
4. void Logic() – логика игры
5. bool GetGameOver() – возвращает конец игры / поражение
6. bool GetGameVictory() – возвращает победу
7. void MotionSnake() – перерисовка змеи
8. void Jury() – выводит результат игры
9. void CreateMap(int \_width, int \_height) – создание карты

**Заключение**

В результате работы было написано 5 классов и программы, которые показывают одно из возможных применений этих классов.

В процессе выполнения работ были изучены такие новые библиотеки, как <vector>, <string>, <iostream>, <fstream>, <windows.h>, понятия как класс, объект, поле, метод, инкапсуляция, конструктор, деструктор, наследование, полиморфизм, перегрузка, пространство имен, исключения и т.п.

**Приложение**

**#include <iostream>**

**#include <cstdio>**

**#include <conio.h>**

**#include <vector>**

**#include <Windows.h>**

**#include <clocale>**

**#include <time.h>**

**using namespace std;**

**struct Snake**

**{**

**vector<int> x;**

**vector<int> y;**

**int headX, headY;**

**int length;**

**Snake(int \_length = 5, int \_headX = 0, int \_headY = 0) : length(\_length), headX(\_headX), headY(\_headY) {};**

**};**

**struct Fruit**

**{**

**int fruitX, fruitY;**

**Fruit(int \_fruitX = 5, int \_fruitY = 5) : fruitX(\_fruitX), fruitY(\_fruitY) {};**

**void CreateFruit(int \_width, int \_height)**

**{**

**srand(time(NULL));**

**fruitX = (rand() % \_width) +1;**

**fruitY = (rand() % \_height) +1;**

**}**

**};**

**struct Map**

**{**

**int height, width;**

**Snake Snake;**

**Fruit Fruit;**

**Map(int \_height = 10, int \_width = 10) : height(\_height), width(\_width) {}**

**void gotoxy(int x, int y)**

**{**

**HANDLE Console = GetStdHandle(STD\_OUTPUT\_HANDLE);**

**if (!Console)**

**return;**

**COORD pos;**

**pos.X = x;**

**pos.Y = y;**

**SetConsoleCursorPosition(Console, pos);**

**}**

**void CreateSnake()**

**{**

**Snake.headX = (width + 2) / 2;**

**Snake.headY = (height + 2) / 2;**

**for (int i = 1; i < 5; i++)**

**{**

**Snake.x.push\_back(Snake.headX + i);**

**Snake.y.push\_back(Snake.headY);**

**}**

**}**

**void PaintMap()**

**{**

**for (int i = 0; i < width + 2; i++)**

**cout << "#";**

**for (int i = 1; i < height + 1; i++)**

**{**

**gotoxy(0, i);**

**cout << "#";**

**gotoxy(width + 1, i);**

**cout << "#";**

**}**

**gotoxy(0, height + 1);**

**for (int i = 0; i < width + 2; i++)**

**cout << "#";**

**gotoxy(width + 15, 0);**

**cout << "Длина змейки: ";**

**gotoxy(width + 15, 2);**

**cout << "Ваша цель: 25";**

**CreateSnake();**

**}**

**};**

**class TheGame**

**{**

**private:**

**int Direction;**

**int Purpose;**

**enum Directon { LEFT, RIGHT, UP, DOWN };**

**Directon dir;**

**bool GameOver;**

**bool GameVictory;**

**Map Map;**

**void gotoxy(int x, int y)**

**{**

**HANDLE Console = GetStdHandle(STD\_OUTPUT\_HANDLE);**

**if (!Console)**

**return;**

**COORD pos;**

**pos.X = x;**

**pos.Y = y;**

**SetConsoleCursorPosition(Console, pos);**

**}**

**public:**

**TheGame(bool \_GameOver = false, bool \_GameVictory = false)**

**{**

**GameOver = \_GameOver;**

**GameVictory = \_GameVictory;**

**Purpose = 25;**

**dir = LEFT;**

**}**

**void Input()**

**{**

**if (\_kbhit())**

**{**

**switch (\_getch())**

**{**

**case 'a':**

**if (dir != RIGHT)**

**dir = LEFT;**

**break;**

**case 'd':**

**if (dir != LEFT)**

**dir = RIGHT;**

**break;**

**case 'w':**

**if (dir != DOWN)**

**dir = UP;**

**break;**

**case 's':**

**if (dir != UP)**

**dir = DOWN;**

**break;**

**}**

**}**

**}**

**void Logic()**

**{**

**int tmpX = Map.Snake.x[0];**

**int tmpY = Map.Snake.y[0];**

**int tmpXX, tmpYY;**

**Map.Snake.x[0] = Map.Snake.headX;**

**Map.Snake.y[0] = Map.Snake.headY;**

**switch (dir)**

**{**

**case LEFT:**

**Map.Snake.headX--;**

**break;**

**case RIGHT:**

**Map.Snake.headX++;**

**break;**

**case UP:**

**Map.Snake.headY--;**

**break;**

**case DOWN:**

**Map.Snake.headY++;**

**break;**

**default:**

**break;**

**}**

**for (int i = 1; i < Map.Snake.length - 1; i++)**

**{**

**tmpXX = Map.Snake.x[i];**

**tmpYY = Map.Snake.y[i];**

**Map.Snake.x[i] = tmpX;**

**Map.Snake.y[i] = tmpY;**

**tmpX = tmpXX;**

**tmpY = tmpYY;**

**}**

**if (Map.Snake.headX > Map.width || Map.Snake.headX < 1 || Map.Snake.headY > Map.height || Map.Snake.headY < 1)**

**GameOver = true;**

**for (int i = 2; i < Map.Snake.length - 1; i++)**

**if ((Map.Snake.x[i] == Map.Snake.headX) && (Map.Snake.y[i] == Map.Snake.headY))**

**GameOver = true;**

**if ((Map.Snake.headX == Map.Fruit.fruitX) && (Map.Snake.headY == Map.Fruit.fruitY))**

**{**

**MessageBeep(0);**

**bool flag = true;**

**while (flag)**

**{**

**flag = false;**

**Map.Fruit.CreateFruit(Map.width, Map.height);**

**for (int i = 0; i < Map.Snake.length - 1; i++)**

**if ((Map.Fruit.fruitX == Map.Snake.x[i]) && (Map.Fruit.fruitY == Map.Snake.y[i]))**

**{**

**flag = true;**

**break;**

**}**

**}**

**Map.Snake.length++;**

**gotoxy(Map.width + 28, 0);**

**cout << Map.Snake.length;**

**if (Map.Snake.length == Purpose)**

**GameVictory = true;**

**Map.Snake.x.push\_back(0);**

**Map.Snake.y.push\_back(0);**

**}**

**}**

**bool GetGameOver()**

**{**

**return GameOver;**

**}**

**bool GetGameVictory()**

**{**

**return GameVictory;**

**}**

**void MotionSnake()**

**{**

**HANDLE hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE);**

**CONSOLE\_CURSOR\_INFO structCursorInfo;**

**GetConsoleCursorInfo(hConsole, &structCursorInfo);**

**structCursorInfo.bVisible = FALSE;**

**SetConsoleCursorInfo(hConsole, &structCursorInfo);**

**for (int i = 1; i < Map.height + 1; i++)**

**{**

**for (int j = 1; j < Map.width + 1; j++)**

**{**

**if (i == Map.Snake.headY && j == Map.Snake.headX)**

**{**

**gotoxy(j, i);**

**SetConsoleTextAttribute(hConsole, (WORD)((0 << 4) | 14));**

**cout << "s";**

**}**

**else if (i == Map.Fruit.fruitY && j == Map.Fruit.fruitX)**

**{**

**gotoxy(j, i);**

**SetConsoleTextAttribute(hConsole, (WORD)((0 << 4) | 12));**

**cout << "a";**

**}**

**else**

**{**

**bool print = false;**

**for (int k = 0; k < Map.Snake.length - 1; k++)**

**{**

**if (Map.Snake.x[k] == j && Map.Snake.y[k] == i)**

**{**

**gotoxy(j, i);**

**SetConsoleTextAttribute(hConsole, (WORD)((0 << 4) | 2));**

**cout << "0";**

**print = true;**

**}**

**}**

**if (!print)**

**{**

**gotoxy(j, i);**

**SetConsoleTextAttribute(hConsole, (WORD)((0 << 4) | 0));**

**cout << " ";**

**}**

**}**

**}**

**}**

**SetConsoleTextAttribute(hConsole, (WORD)((0 << 4) | 15));**

**}**

**void Jury()**

**{**

**if (GameVictory)**

**{**

**gotoxy(Map.width + 15, 4);**

**cout << "Вы победили ";**

**}**

**if (GameOver)**

**{**

**gotoxy(Map.width + 15, 4);**

**cout << "Вы проиграли ";**

**}**

**}**

**void CreateMap(int \_width, int \_height)**

**{**

**Map.width = \_width;**

**Map.height = \_height;**

**Map.PaintMap();**

**}**

**};**

**int main()**

**{**

**setlocale(LC\_ALL,"rus");**

**HWND hWnd = GetForegroundWindow();**

**ShowWindow(hWnd, SW\_SHOWMAXIMIZED);**

**int x, y;**

**TheGame Game;**

**cout << "Введите ширину карты: " << endl;**

**cin >> x;**

**cout << "Введите высоту карты: " << endl;**

**cin >> y;**

**system("cls");**

**Game.CreateMap(x, y);**

**while (!Game.GetGameOver() && !Game.GetGameVictory())**

**{**

**Game.MotionSnake();**

**Game.Input();**

**Game.Logic();**

**}**

**Game.Jury();**

**system("pause");**

**}**