Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное   
учреждение высшего профессионального образования

«Нижегородский государственный университет им. Н.И. Лобачевского»

Институт информационных технологий математики и механики

**Отчет по лабораторным работам за второй семестр**

**Выполнил**:студент группы 381703-2

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Рябушева И.В.

Подпись

**Научный руководитель**:

Доцент каф.МОСТ

Кандидат технических наук

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Сысоев А.В

Подпись

Нижний Новгород

2018

**Оглавление**

[**Введение** 3](#_Toc515573068)

[Блок **1** 4](#_Toc515573070)

[**Постановка задачи** 4](#_Toc515573071)

[**Руководство пользователя** 5](#_Toc515573072)

[**Руководство программиста** 6](#_Toc515573073)

[**Блок 2** 7](#_Toc515573074)

[**Постановка задачи** 7](#_Toc515573075)

[**Руководство пользователя** 8](#_Toc515573076)

[**Руководство программиста** 10](#_Toc515573077)

[**Блок 3** 11](#_Toc515573078)

[**Постановка задачи** 11](#_Toc515573079)

[**Руководство пользователя** 12](#_Toc515573080)

[**Руководство программиста** 14](#_Toc515573081)

[**Блок 4** 15](#_Toc515573082)

[**Постановка задачи** 15](#_Toc515573083)

[**Руководство пользователя** 16](#_Toc515573084)

[**Руководство программиста** 18](#_Toc515573085)

[**Блок 5** 19](#_Toc515573086)

[**Постановка задачи** 19](#_Toc515573087)

[**Руководство пользователя** 20](#_Toc515573088)

[**Руководство программиста** 21](#_Toc515573089)

[**Блок 6** 22](#_Toc515573090)

[**Постановка задачи** 22](#_Toc515573091)

[**Руководство пользователя** 23](#_Toc515573092)

[**Руководство программиста** 25](#_Toc515573093)

[**Заключение** 26](#_Toc515573094)

[**Литература** 27](#_Toc515573095)

[**Приложение** 28](#_Toc515573096)

**Введение**

Целью лабораторных работ является обучение объектно-ориентированному программированию, овладение практическими навыками в разработке и проектировании программ. Каждая из работ требует разработать класс или систему классов.

**Блок 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 программа демонстрирует возможности разработанного класса Rationale. Она выводит на экран результаты выполнения четырех 4 основных арифметических операций (сумма, вычитание, умножение и деление) с рациональными дробями, знаменатель которых не равен нулю, на консоль.

Для выполнения данных операций с иными дробями, то в функции main() требуется изменить значения числителя и знаменателя для соответствующих дробей.

**Руководство программиста**

Данная программа написана с помощью одного класса Rationale и функции main.

Класс содержит три поля: p – числитель, q – знаменатель и метод NOD, который позволяет найди НОД числителя и знаменателя, чтобы дроби хранились в несократимом виде, а также конструктор, оператор присваивания, четыре метода для выполнения основных арифметических операций с дробями и метод, позволяющий выводить дробь на консоль.

Функция main имеет линейный вид, в ходе ее выполнения демонстрируются возможности класса.

**Блок 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**

При запуске программы появляется консольное окно, в которое нужно ввести степень полинома и соответствующее количество коэффициентов.

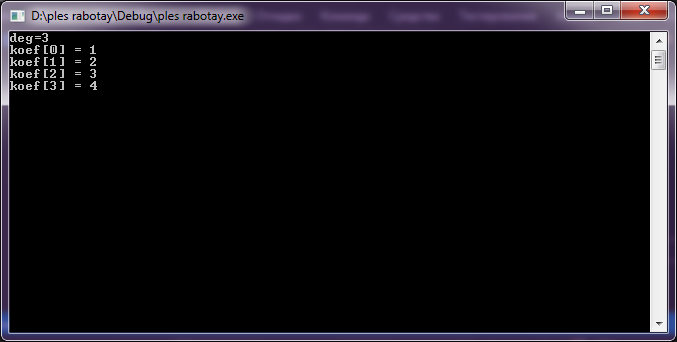


Рис. 1 Ввод данных

Затем выводится полученный полином, а затем программа требует ввести значение точки x, в которой требуется вычислить значение полинома в ней. Далее программа выводит полученное значение полинома в заданной точке и производную полинома.

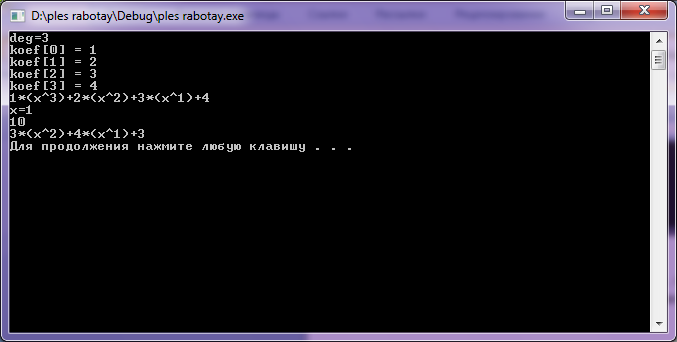


Рис. 2 Результат выполнения программы

**Руководство программиста**

Данная программа написана с помощью одного класса Polynome и функции main.

Класс содержит два поля: deg – степень полинома и динамический массив указателей koef типа double для хранения вещественных коэффициентов заданного полинома, конструктор, деструктор для удаления динамического массива указателей, оператор присваивания и пять методов: GetDeg – возвращает степень полинома, GetKoef – возвращает вещественный коэффициент с заданной пользователем позиции из динамического массива указателей koef, Calculate – вычисляет значение полинома в заданной пользователем точке, derivative – позволяет найти производную от заданного полинома, а также метод Print для вывода полинома на консоль.

В ходе выполнения функции main демонстрируются возможности класса.

**Блок 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 появляется консольное окно, в которое нужно ввести границы интегрирования функции. А затем требуется ввести номер соответствующей функции, которую вы хотите проинтегрировать.



Рис. 3 Ввод данных

После выбора функции в консоли появляется меню и пользователю нужно выбрать интересующую функцию программы.

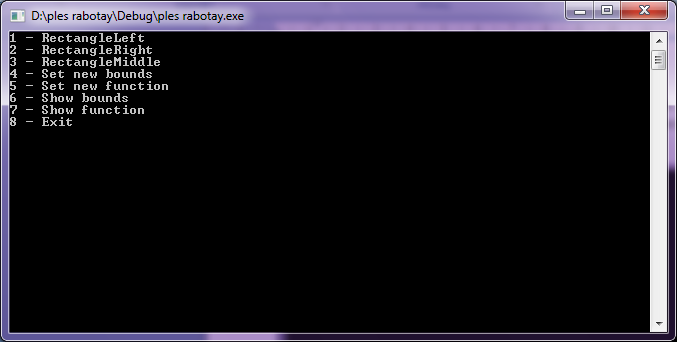


Рис. 4 Консольное меню

**Руководство программиста**

Данная программа написана с помощью одного класса Integral и функции main.

В модификаторе доступа private находятся такие поля, как left и right типа double для хранения пределов интегрирования функции, Func для хранения номера функции, а также метод Calculate, который позволяет вычислять значение функции в некоторой точке. В классе реализован набор set, get методов, конструктор, оператор присваивания и три метода для вычисления интеграла в заданных пределах интегрирования различными способами (RectangleLeft - метод левых, RectangleRight - правых и RectangleMiddle - средних прямоугольников).

**Блок 4**

**Постановка задачи**

Разработать класс **Весы** **напольные**.

Класс должен хранить историю наблюдений за показаниями веса членов семьи (до пяти человек). Показания описываются датой (день, месяц, год) и именем члена семьи. При поступлении нового наблюдения для уже существующей даты старое наблюдение заменяется. Вес задается в килограммах с точностью до 50 граммов.

Класс должен содержать необходимые служебные методы.

Класс должен предоставлять следующие операции: 1) установить начальную дату наблюдений, 2) узнать начальную дату наблюдений, 3) задать наблюдение, 4) узнать вес в выбранном наблюдении, 5) найти средний вес члена семьи в выбранном месяце или за всю историю наблюдений, 6) найти минимальный вес члена семьи в выбранном месяце или за всю историю наблюдений и дату, когда он наблюдался 7) найти максимальный вес члена семьи в выбранном месяце или за всю историю наблюдений и дату, когда он наблюдался, 8) сохранить историю наблюдений в файл, 9) считать историю наблюдений из файла.

**Руководство пользователя**

При запуске программы выводится консольное меню. Для выбора нужного функционала, необходимо ввести число, написанное напротив пункта. Функционал пунктов меню интуитивно понятен и соответствует названию.

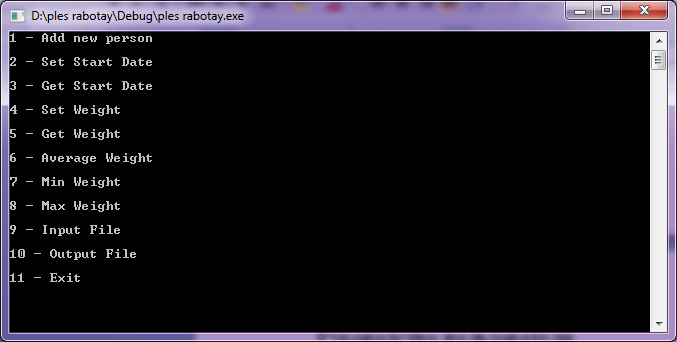


Рис. 5 Консольное меню

Данные можно считать из файла или добавлять отдельно, указывая имя члена семьи, дату и вес. А также можно сохранить историю наблюдений в файл.

**Руководство программиста**

В программе представлена система объектов, состоящая из трех структур и класса.

Структура Date используется для хранения даты и структура Weight для хранения наблюдения веса, соответствующего определенной дате, структура Person содержит вектор всех записанных наблюдений некоторого члена семьи. В самом классе Scales содержится вектор объектов типа Person. В классе реализованы методы считывания истории наблюдения из файла и сохранения в файл, а также методы: AddName - добавить нового члена семьи, SetWeight - добавить наблюдения для определенного человека, GetWeight - узнать вес в выбранном наблюдении, AvgWeightMonth - найти средний вес члена семьи в выбранном месяце или AvgWeightAll - за всю историю наблюдений, MinWeightMonth - найти минимальный вес члена семьи в выбранном месяце или MinWeightAll - за всю историю наблюдений, MaxWeightMonth - найти максимальный вес члена семьи в выбранном месяце или MaxWeightAll - за всю историю наблюдений.

**Блок 5**

**Постановка задачи**

Разработать классы **Банкомат** и **Процессинговый центр**.

Класс **Банкомат** должен имитировать работу банкомата по приему и выдаче наличных денежных средств (в рублях). Класс должен поддерживать работу с купюрами в 100, 200, 500, 1000, 2000, 5000 рублей. Выдаваемая или принимаемая за одну операцию сумма ограничена 40 купюрами (независимо от их достоинства). Купюры каждого достоинства хранятся в отдельной кассете. Емкость каждой кассеты – 2000 тысяч купюр. Считать, что начальная загрузка банкомата составляет 80% для каждой кассеты. Считать, что клиент банка идентифицируется по номеру пластиковой карты (для упрощения – номер карты от «0001» до «9999»).

База клиентов хранится в классе **Процессинговый центр**. Номера выданных клиентам карт могут идти не подряд. Считать, что информация о клиенте состоит из номера карты, ФИО владельца, суммы на счету, (для упрощения – без копеек), PIN-кода (последовательность из 4-х цифр, каждая от 0 до 9).

Класс **Банкомат** должен предоставлять следующие операции: 1) принять карту клиента, 2) найти клиента по номеру карты, 3) проверить PIN-код, 4) распечатать состояние счета клиента, 5) выдать клиенту наличные (списав выданную сумму со счета), 6) принять от клиента наличные (зачислив полученную сумму на счет), 7) заблокировать карту клиента, если до ее возврата клиенту три раза подряд набран неверный PIN-код, 8) вернуть карту клиенту.

Все операции должны сопровождаться необходимыми проверками на указанные выше ограничения.

Класс **Процессинговый центр** должен использоваться для поддержки работы класса **Банкомат** и может быть разработан в минимально-необходимом варианте.

**Руководство пользователя**

При запуске программы появляется консольное окно, в которое требуется ввести номер вашей карты, а затем пароль (если пароль был введен неверно три раза, то карта блокируется).

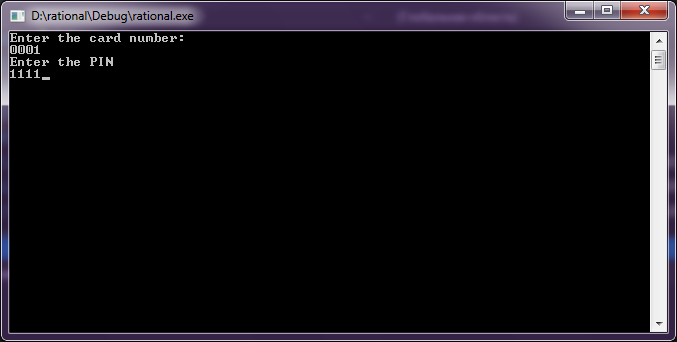


Рис. 6 Ввод данных

Если пароль был введен верно, то выводится консольное меню и пользователь может выбрать необходимую операцию (зачислить или снять деньги со своего счета, проверить состояние счета или завершить операцию).

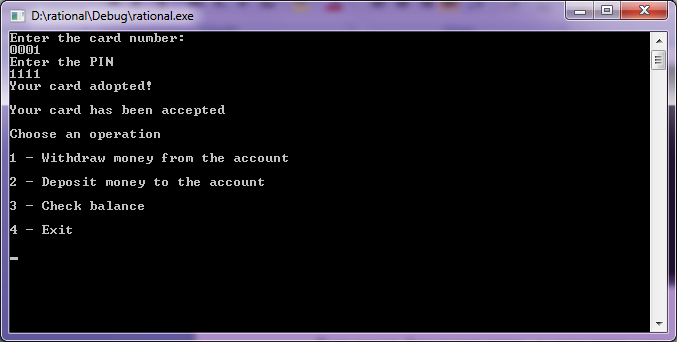


Рис. 7 Перечень возможных операций

**Руководство программиста**

Два класса (Person и PC) связаны отношением агрегации. Первый объект представляет собой набор полей, хранящих информацию о человеке: FullName - ФИО, Numb – номер карты, Sum – состояние счета, Pin – пин-код и Block – заблокирована карта или нет. Реализованы все необходимые методы для взаимодействия с этим классом: get, set и пр. Класс PC хранит вектор объектов типа Person, а также в классе реализованы методы: AddClient – добавить клиента, CheckSum – проверить состояние счета, FindByNumb – найти карту по номеру, CheckByNumb - проверить существует ли карта с таким номером, CheckByPin – проверить корректен ли введенный пин-код, LooseCash – снять деньги, AddCash – зачислить деньги на счет, GetStatus – проверить заблокирована карта или нет, SetBlock – заблокировать карту. В классе банкомат ATM хранятся кейсы с купюрами номиналом 100, 200, 500, 1000, 2000, 5000 , указатель на переменную типа РС и переменная i для хранения позиции клиента в процессинговом центре. В классе банкомат реализованы методы: проверка статуса карты, заблокировать карту, поиск карты по номеру, проверка пин-кода, проверить баланс карты, зачислить и снять деньги со счета.

**Блок 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**

После запуска программы начинается игра. Управление змейкой происходит с помощью клавиш-«стрелок» (вверх, вниз, влево, вправо). Под игровым полем написан текущий счет и счет, необходимый для победы.

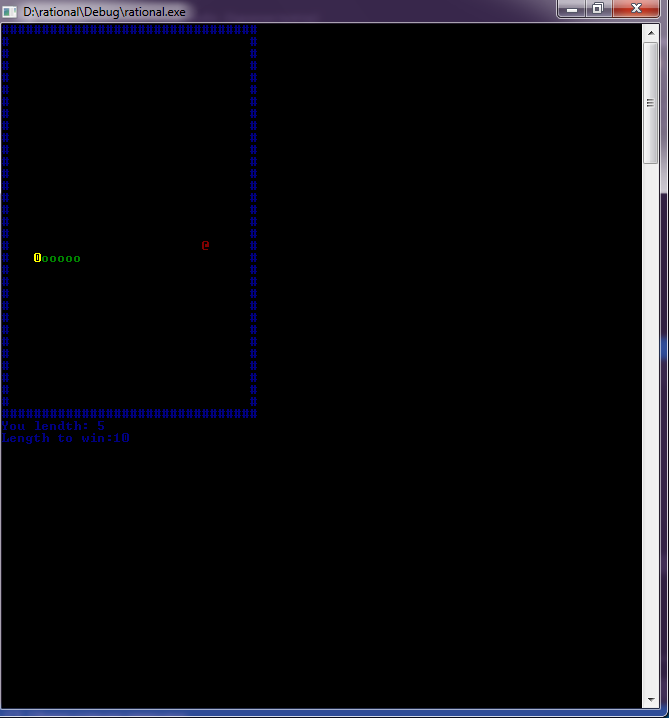


Рис. 8 Игра

Если змейка врезается в стенку или в себя, то игра заканчивается.

**Руководство программиста**

enum Direction { RIGHT, LEFT, UP, DOWN } – перечисление направлений, в которых может двигаться змейка.

В модификаторе доступа private класса Game находятся такие поля, как х, у – координаты головы змейки, 2 массива tailX[100] и tailY[100] для хранения координат хвоста, nTail – длина хвоста, foodX и foodY – координаты еды, width, height – ширина и высота поля, dir – направление движения змейки, speed – скорость, lenghtToWin – длина для победы, 2 переменные типа bool GameOver, Win для хранения состояния игры, а также метода GetRandFood и GetRandSnake для получения еды и змейки в рандомных местах. В классе реализованы методы Map для построения карты игры, Control для изменения направления движения змейки, Move для передвижения змейки.

**Заключение**

В ходе работы были спроектированы и реализованы разнообразные программы с использованием концепции объектно-ориентированного программирования, тем самым были приобретены важные навыки в данной области.

**Литература**

1. Design Patterns: Elements of Reusable Object-Oriented Software («Приёмы объектно-ориентированного проектирования. Паттерны проектирования») Эрих Гамма, Ричард Хелм, Ральф Джонсон, Джон Влиссидес, ISBN 0-201-63361-2, 5-469-01136-4
2. Кнут Д. Э. Искусство программирования, том 4, A. Комбинаторные алгоритмы, часть 1 = The Art of Computer Programming, Volume 4A: Combinatorial Algorithms, Part 1 / под ред. Ю. В. Козаченко. — 1. — Москва: Вильямс, 2013. — Т. 4. — 960 с. — ISBN 978-5-8459-1744-7
3. Бьерн Страуструп Язык программирования С++

**Приложение**

**Код лабораторной работы №6**

#include<iostream>

#include <windows.h>

#include <time.h>

#include <conio.h>

using namespace std;

enum Direction { RIGHT, LEFT, UP, DOWN };

class Game

{

bool GameOver;

bool Win;

int x, y;

int foodX, foodY;

int tailX[100];

int tailY[100];

int nTail = 5;

Direction dir;

int width, height;

int speed;

int lengthToWin;

void GetRandFood()

{

srand(time(NULL));

foodX = rand() % width;

foodY = rand() % height;

}

void GetRandSnake() {

x = rand() % (width - 4) + 1;

y = rand() % height + 1;

int m = x;

for (int i = 0; i < 4; i++) {

m++;

tailX[i] = m;

tailY[i] = y;

}

}

public:

Game(int \_width, int \_height, int \_speed, int \_lengthToWin)

{

GameOver = false;

Win = false;

width = \_width;

height = \_height;

speed = \_speed;

lengthToWin = \_lengthToWin;

dir = LEFT;

GetRandSnake();

GetRandFood();

};

void Map()

{

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

system("cls");

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

{

SetConsoleTextAttribute(hConsole, (WORD)((0 << 4) | 1));

cout << "#";

}

cout << endl;

for (int i = 0; i < height + 1; i++) {

for (int j = 0; j < width + 2; j++)

{

if (j == 0 || j == width) {

SetConsoleTextAttribute(hConsole, (WORD)((0 << 4) | 1));

cout << "#";

}

if (i == y && j == x) {

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

cout << "0";

}

else if (i == foodY && j == foodX) {

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

cout << "@";

}

else {

bool print = false;

for (int k = 0; k < nTail; k++) {

if (tailX[k] == j && tailY[k] == i) {

print = true;

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

cout << "o";

}

}

if (!print)

cout << " ";

}

}

cout << endl;

}

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

{

SetConsoleTextAttribute(hConsole, (WORD)((0 << 4) | 1));

cout << "#";

}

cout << endl;

cout << "You lendth: " << nTail << endl;

cout << "Length to win:" << lengthToWin << endl;

}

void Control() {

Sleep(speed);

if (\_kbhit())

{

switch (\_getch())

{

case 75:

if (dir != RIGHT)

dir = LEFT;

break;

case 77:

if (dir != LEFT)

dir = RIGHT;

break;

case 72:

if (dir != DOWN)

dir = UP;

break;

case 80:

if (dir != UP)

dir = DOWN;

break;

case 'x':

GameOver = true;

break;

}

}

}

void Move()

{

int prX = tailX[0];

int prY = tailY[0];

int pr2X, pr2Y;

tailX[0] = x;

tailY[0] = y;

for (int i = 1; i < nTail; i++)

{

pr2X = tailX[i];

pr2Y = tailY[i];

tailX[i] = prX;

tailY[i] = prY;

prX = pr2X;

prY = pr2Y;

}

switch (dir)

{

case RIGHT:

x++;

break;

case LEFT:

x--;

break;

case UP:

y--;

break;

case DOWN:

y++;

break;

}

if (x > width || x < 0 || y > height || y < 0)

{

GameOver = true;

}

for (int i = 2; i < nTail; i++)

{

if (tailX[i] == x && y == tailY[0])

GameOver = true;

}

if (x == foodX && y == foodY)

{

bool tmp = true;

while (tmp)

{

tmp = false;

GetRandFood();

for (int i = 0; i <= nTail; i++)

if (tailX[i] == foodX && tailY[i] == foodY)

{

tmp = true;

break;

}

}

nTail++;

}

if (lengthToWin == nTail)

Win = true;

}

bool Over()

{

return GameOver;

};

bool GameWin()

{

return Win;

};

};

int main()

{

HWND hWnd = GetForegroundWindow();

ShowWindow(hWnd, SW\_SHOWMAXIMIZED);

srand(time(NULL));

/\*int \_width;

int \_height;

int \_speed;

int \_lengthToWin;

cout << "Enter width" << endl;

cin >> \_width;

cout << "Enter height" << endl;

cin >> \_height;

cout << "Enter speed" << endl;

cin >> \_speed;

cout << "Enter length to win" << endl;

cin >> \_lengthToWin;

//Game game(\_width, \_height, \_speed, \_lengthToWin);\*/

Game game(30, 30, 150, 10);

//system("cls");

while (!game.Over() && !game.GameWin())

{

game.Map();

game.Control();

game.Move();

}

if (game.Over())

{

cout << "GAME OVER" << endl;

cout << "YOU LOSE" << endl;

}

if (game.GameWin())

{

cout << "YOU WON" << endl;

}

system("pause");

return 0;

}