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

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Илюхин И.И

Подпись

**Научный руководитель**:

Доцент каф.МОСТ

Кандидат технических наук

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

Подпись

Нижний Новгород

2017

**Содержание**

**Введение** **2**

**Лабораторная работа номер 1 “Рациональная дробь”3**

Постановка задачи3

Руководство пользователя4

Руководство программиста5

**Лабораторная работа номер 2 “Полином”** **6**

Постановка задачи 6

Руководство пользователя 7

Руководство программиста 8

**Лабораторная работа номер 3 “Расчет интегралов”** **9**

Постановка задачи 9

Руководство пользователя 10

Руководство программиста 11

**Лабораторная работа номер 4 “Шагомер”** **12**

Постановка задачи 12

Руководство пользователя 13

Руководство программиста 14

**Лабораторная работа номер 5 “Банкомат”** **16**

Постановка задачи 16

Руководство пользователя 17

Руководство программиста 18

**Лабораторная работа номер 6 “Змейка”** **19**

Постановка задачи 20

Руководство пользователя 22

Руководство программиста 21

**Заключение** **22**

**Список литературы** **23**

**Приложение**  **24**

**Введение**

**Объе́ктно-ориенти́рованное программи́рование (ООП)** — методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определённого класса, а классы образуют иерархию наследования.

Идеологически ООП — подход к программированию как к моделированию информационных объектов, решающий на новом уровне основную задачу структурного программирования: структурирование информации с точки зрения управляемости, что существенно улучшает управляемость самим процессом моделирования, что в свою очередь особенно важно при реализации крупных проектов.

**Лабораторная работа номер 1**

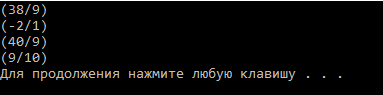
**“Рациональная дробь”**

**Постановка задачи**

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

**Руководство пользователя**

1. При запуске программа продемонстрирует возможности разработанного класса TRational. Если вы хотите сделать какие либо операции с дробями, то прочитайте пункт 2.

****

1. Нужно написать TRational a(число, число не ноль);

TRational b( число, число не ноль);

c = a + b;

c.print();

после этих операций в вашем инт мейне, программа выведем сумму двух дробей, аналогично с делением, умноженим и разностью.

1. Нажать на кнопку локальный отладчик Windows и он выведет сумму, разность, умножение и деление.

**Руководство программиста**

Программа написана при помощи одного класса (class), который назван TRational.

В модификаторе доступа private находятся следующие вспомогательные поля: bool exist; // Существует ли дробь

int p; //Числитель

int q; //Знаменатель

int NOD(int a, int b);

А также методы: MakeFinal(проверка), Add(сумма), Subtract(разность), Multiply(умножение), Devide(деление), IsExist(существование).

**Лабораторная работа номер 2**

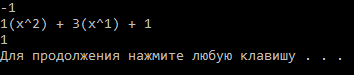
**“Полином”**

**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 в консоли выводится



Для того, чтобы у пользователя вывелись результаты, нужно ввести массив в инт мейне с коэффициентами уравнения, задать степень многочлена, посчитать значение уравнения в какой то точке, и вывести его в консоль.

**Руководство программиста**

**Программа написана при помощи** class Polynom.

В модификаторе доступа private находятся следующие вспомогательные поля: double\* koef; //коэффицент

int deg; //степень

А также множество методов: GetDegree() (получение степени)

double Calculate(double x) (подсчет)

Derivative() (производная) ToConsole() (в консоль)

**Лабораторная работа номер 3**

**“Расчет интегралов”**

**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 в консоли выводится результат вычисления интеграла в определенных пределах интегрирования, с заданным числом отрезков в методе прямоугольника.



В i.SetBounds( , ); мы должны задать пределы интегрирования.

В i.SetParams( , , ); идут 3 переменные, первая число, второй параметр, на выбор из: SIN, COS, EXP, LOG, третий параметр идет на выбор из методов прямоугольников, а именно LEFT, MID, RIGHT.

Дальше мы должны подсчитать это все, i.Calculate(true);

**Руководство программиста**

**Программа написана при помощи** class Integral.

В модификаторе доступа private находятся следующие вспомогательные поля: double leftBound;(левая граница)

double rightBound; (права граница)

functions func; (переменная из enum functions { SIN, COS, EXP, LOG };)

method meth; (переменная из enum method { LEFT, MID, RIGHT };)

int N; (Переменная используемая для подсчета интеграла)

А также различные методы: double inFunc(double x) (для использования SIN,COS…) , double calcIntegralL() (вычисление с помощью левых прямоугольников) , double calcIntegralM() (вычисление с помощью средних прямоугольников) , double calcIntegralR() вычисление с помощью правых прямоугольников) , void setBounds(double a, double b) (устанавливаем границы) , void getBounds(double \*a, double \*b) (получаем границы) , void setParams(int \_N, functions f = SIN, method m = LEFT) ( устанавливаем параметры) , double calculate(bool toConsole=false) (проверка, при которой если правда, тогда выводим значение в консоль)

**Лабораторная работа номер 4**

**“Шагомер”**

**Постановка задачи**

* Разработать класс Шагомер.
* Класс должен хранить историю подсчета шагов владельца. Каждый подсчет описывается датой (день, месяц, год) и интервалом времени (час, минута начала движения, час, минута окончания движения). Подсчет ведется в шагах с точностью до единицы.
* Класс должен содержать необходимые служебные методы.
* Класс должен предоставлять следующие операции: 1) установить дату начала подсчетов, 2) узнать дату начала подсчетов, 3) задать подсчет, 4) получить информацию о выбранном подсчете, 5) найти среднее число шагов в выбранном месяце или за всю историю наблюдений, 6) найти среднее число шагов в выбранный день недели за всю историю наблюдений, 7) найти максимальное число шагов в день в выбранном месяце или за всю историю наблюдений и дату, когда оно было достигнуто, 8) сохранить историю подсчетов в файл, 9) считать историю подсчетов из файла.

**Руководство пользователя**

Для начала нужно ввести 2 переменные из классов Pedometr, Date. Обозначить день, месяц и год с переменной обозначенный выше в классе Date, далее надо подсчитать количество шагов с переменной обозначенный выше в классе Date, сохранить все это в файл и потом при необходимо открыть его.

**Руководство программиста**

Программа написана при помощи двух классов class SingleCount (с ним мы работаем изначально, в нем мы используем

int steps;(шаги)

int startHour;(старт по часу)

int startMinute;(старт по минуте)

int endHour;(граница по часу)

int endMinute;(граница по минуте)

Date date(дата из структуры), а так же различные методы: getSteps(), getStartHour(), getStartMinute(), getEndHour(), getEndMinute(), getDate() все они просто возращают установленные в private поля) и class Pedometr( поля: SingleCount startCount, vector<SingleCount> Counts (Вектор), методы: void setStartDate(Date d)(устанавливаем дату), Date getStartDate() (узнаем старт даты), void AddCount(Date d, int sm, int sh, int em, int eh, int c) (добавляем счетчик) , int GetAvierageCounts(int month)(узнаем среднее кол-во за месяц), int GetAvierageCounts(), int GetDayAvierageCounts(int DayOfWeek) (узнаем среднее в день) , int GetMaxStepsForDay(Date& d, int mount = -1)( максимальное кол-во шагов в день) , void saveFile() ( сохраняем файл), void openFile() (открываем файл) ), а так же с использованием структуры struct Date в которой int day, int month, int year.

**Лабораторная работа номер 5**

**“Банкомат”**

**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 вы должны ввести пинкод, если правильно, то в консоли выводится номер карты клиента, показывается состояние счета клиента. Так же вы можете ввести или снять деньги со своего счета. Потом банкомат отдает вам карту. Если пинкод 3 раза введен неправильно, то карточка блокируется.

**Руководство программиста**

Данная программа написана с использованием 3 классов, первый класс User, там есть пинкод, карднамбер, имя, деньги и состояние счета. Потом идет процессинговый центр ProcessingCenter, где хранится вся информация и мы используем класс User. Мы регистрируем пользователя в процессинговом центре, проверяем его счет и многое другое. Самый последний класс ATM банкомат, там мы используем класс ProcessingCenter, в нем мы загражаем карточку, ищем пользователя по карднамберу, запрашиваем пинкод и дальше вы можете узнать состояние счета, снять, вывести деньги и в конце возвращаем карточку.

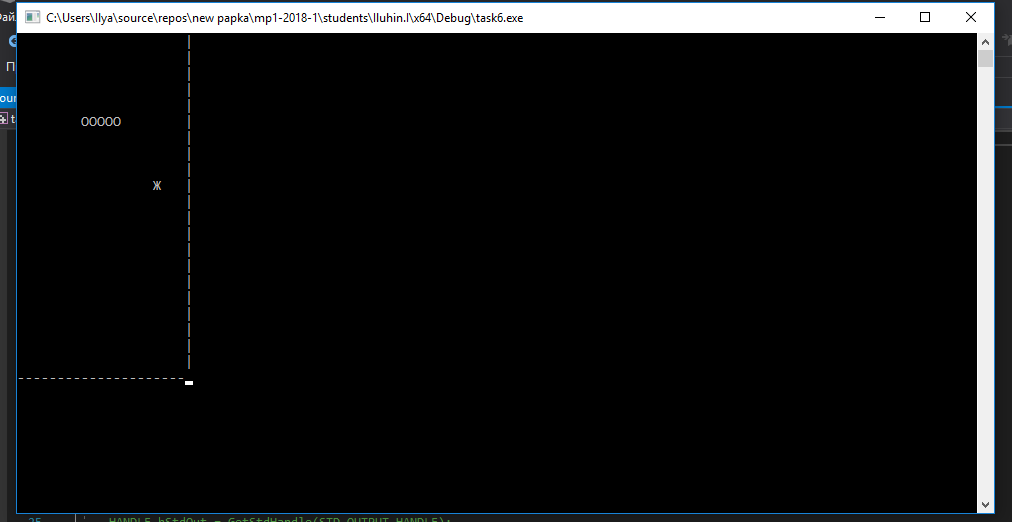
**Лабораторная работа номер 6**

**“Змейка”**

**Постановка задачи**

* 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**

При запуске программы пользователь сразу начинает играть в змейку, управление при помощи клавиш W A S D. Змейка двигается в заданном пространстве, цель играть, как можно больше собрать жуков, от которых ваша змейка будет расти, нельзя врезаться в стенки и в саму себя 

**Руководство программиста**

Данная программа использует структуру Point для координат, еще мы выписали enum Course {LEFT,RIGHT,UP,DOWN} для направлений, дальше у нас идет класс Zmeyka, для начала я написал куда изначально идет змейка(влево), ввел поле жук с использованием структуры, ввел вектор змей и размеры карты. Из методов сначала мы делаем инициализацию, делаем жука, потом мы делаем апдейт нашей змейки, фактически то, как двигается наша змейка, окрашиваем ее, дальше мы осуществляем проверку, чтобы не было никаких ошибок, дальше мы осуществляем первоначальное движение змейки влево, рисуем жука и змейку, очищаем мусор в консоли, еще для того, что запомнить цвета вводим enum где каждому цвету соответствует какое-то число.

**Заключение**

В ходе проектирования и реализации данных программ я приобрел важных навык в области объектно-ориентированного программирования и стал лучше ориентироваться в С++ и Visual Studio в целом.

**Список литературы**

* 1. Столлингс, В. Структурная организация и архитектура компьютерных систем, 5-е изд.: Пер. с англ. — М.: Издательский дом «Вильямс», 2002. — 896 с.: ил. — Парал. тит. англ.
  2. Брайан Керниган, Деннис Ритчи «Язык программирования Си».
  3. Герберт Шилдт - Полный справочник по C

1. Code Complete: Second Edition («Совершенный код. Мастер-класс»), Стив Макконнелл, ISBN 978-5-7502-0064-1

**Приложение**

Код Змейки:

#include <iostream>

#include <time.h>

#include <vector>

#include <Windows.h>

#include <conio.h>

using namespace std;

struct Point

{

int x;

int y;

Point() {};

Point(int \_x, int \_y)

{

x = \_x;

y = \_y;

}

};

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

/\*void SetColor(int text, int background)

{

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

SetConsoleTextAttribute(hStdOut, (WORD)((background << 4) | text));

}

\*/

class Zmeyka

{

private:

Course dir = LEFT;

vector<Point> zmey;

Point Beetle;

char map[21][21];

public:

Zmeyka()

{

srand(time(NULL));

Initialzation();

GenerateBeetle();

Update();

}

void Initialzation()

{

for (int i = 0; i < 5; i++)

{

Point a(15 + i, 5);

zmey.push\_back(a);

}

}

void Update()

{

while (zmey.size()<400 && Validation())

{

if (InsideZmey())

{

zmey.push\_back(zmey[zmey.size() - 1]);

GenerateBeetle();

}

Draw();

Move();

Point dif;

if (\_kbhit())

{

switch (\_getch())

{

case 'a':

dir = LEFT;

break;

case 'd':

dir = RIGHT;

break;

case 'w':

dir = UP;

break;

case 's':

dir = DOWN;

break;

}

}

switch (dir)

{

case LEFT: {dif.x = -1; dif.y = 0; break; }

case RIGHT: {dif.x = 1; dif.y = 0; break; }

case UP: {dif.x = 0; dif.y = -1; break; }

case DOWN: {dif.x = 0; dif.y = 1; break; }

}

zmey[0].x += dif.x;

zmey[0].y += dif.y;

Sleep(350);

}

//Clear();

cout << "\n\tGAME OVER";

}

bool Validation()

{

if (zmey[0].x == -1 || zmey[0].y == -1 || zmey[0].x == 21 || zmey[0].y == 21)

return false;

for (int i = 1; i < zmey.size(); i++)

if (zmey[0].x == zmey[i].x && zmey[0].y == zmey[i].y)

return false;

return true;

}

void Move()

{

for (int i = zmey.size()-1; i > 0; i--)

zmey[i] = zmey[i-1];

}

void Draw()

{

Clear();

map[Beetle.x][Beetle.y] = 'Ж';

for each (auto a in zmey)

{

map[a.x][a.y] = 'O';

}

for (int i = 0; i < 21; i++)

{

for (int j = 0; j < 21; j++)

cout << map[j][i];

cout << "|\n";

}

for (int i = 0; i < 21; i++)

cout << "-";

}

void Clear()

{

system("cls");

for (int i = 0; i < 21; i++)

for (int j = 0; j < 21; j++)

map[i][j] = ' ';

}

void GenerateBeetle()

{

Beetle.x = rand() % 20;

Beetle.y = rand() % 20;

if (InsideZmey())

GenerateBeetle();

}

bool InsideZmey()

{

for each (Point a in zmey)

{

if (a.x == Beetle.x && a.y == Beetle.y)

return true;

}

return false;

}

HANDLE hConsole;

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

enum ConsoleColor

{

Black = 0,

Blue = 1,

Green = 2,

Cyan = 3,

Red = 4,

Magenta = 5,

Brown = 6,

LightGray = 7,

DarkGray = 8,

LightBlue = 9,

LightGreen = 10,

LightCyan = 11,

LightRed = 12,

LightMagenta = 13,

Yellow = 14,

White = 15

};

void SetColor(ConsoleColor head, ConsoleColor background)

{

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

}

};

int main()

{

setlocale(LC\_ALL, "");

Zmeyka z;

cout << "\n";

system("pause");

return 0;

}