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

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Сизов И.И

Подпись

**Научный руководитель**:

Доцент каф.МОСТ

Кандидат технических наук

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

Подпись

Нижний Новгород

2017

**Содержание**

**Введение2**

**Лабораторная работа номер 1 “Длинное число”3**

Постановка задачи3

Руководство пользователя4

Руководство программиста5

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

Постановка задачи6

Руководство пользователя7

Руководство программиста8

**Лабораторная работа номер 3 “Пользовательское меню”9**

Постановка задачи9

Руководство пользователя10

Руководство программиста11

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

Постановка задачи12

Руководство пользователя13

Руководство программиста14

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

Постановка задачи16

Руководство пользователя18

Руководство программиста19

**Лабораторная работа номер 6 “Змейка”21**

Постановка задачи21

Руководство пользователя22

Руководство программиста23

Приложение26

**Заключение3**

**Список литературы3**

**Введение**

Объектно-ориентированное программирование (ООП) - это парадигма (совокупность понятий и идей) программирования, в рамках которой ставят понятия объектов и классов. Сейчас ООП так или иначе присутствует во всех языках, поэтому понимание его основ просто необходимо для всех, кто собирается заняться программированием.

**Лабораторная работа номер 1**

**“Длинное число”**

**Постановка задачи**

1. Разработать класс **Длинное число**.
2. Класс должен хранить знаковые целые числа длиной 64 бита, используя для этого целый тип int (unsigned int) длиной 32 бита.
3. Класс должен предоставлять 5 стандартных арифметических операций для целых чисел, сохраняя ту же семантику обработки переполнения, что и для типа int.
4. Класс должен содержать все необходимые конструкторы, оператор присваивания, а также «уметь» выводить себя на консоль.

**Руководство пользователя**

Для того чтобы создать переменную длинного числа напишите LongNumber далее “название вашей переменной” указав параметр число типа char.

**Пример:**

LongNumber a("5");

Класс использует стандартные перегруженные операторы “+ - \* / %”, поэтому для того что бы выполнять арифметические действия необходимо между двумя переменными использовать знак операции.

**Пример:**

Для того чтобы чтобы сложить два длинных числа используйте основной оператор сложения “+”.

LongNumber c = a + b;

Для того чтобы вывести длинное число на экран используйте функцию print(); обратившись через оператор “точка”.

**Пример:**

LongNumber a("5");

a.print();

**Руководство программиста**

Программа написана при помощи одного класса, который назван LongNumber

В модификаторе доступа private находятся следующие вспомогательные поля:

1. char arr[20] - Данный массив используется как основной для хранения и обработки информации.
2. bool limit - Вспомогательная переменная для логики.

В модификаторе доступа private находятся следующие вспомогательные методы:

1. void basis() – Функция реализует заполнение основного массива нулями, ничего не принимает, нет возвращаемого значения.
2. int atoi1(char elem1) – Функция реализует преобразование типа char к типу int, принимает символ типа char, возвращает соответственно число типа int.
3. char itoa1(int elem2) – Обратная функция к функции int atoi1(char elem1) см. выше.
4. void rank() – Функция увеличивает разряд длинного числа, нет принимающего и возвращаемого значения.

В модификаторе доступа паблик находятся следующие методы:

1. LongNumber() – Конструктор по умолчанию.
2. LongNumber(char \*str)-Конструктор принимаемый строку и записывающий задом на перед для удобства
3. LongNumber operator+(const LongNumber &cons) - Перегрузка оператора “+”
4. LongNumber operator-(const LongNumber &cons) - Перегрузка оператора “-”
5. LongNumber operator\*(const LongNumber &cons) - Перегрузка оператора “\*”
6. LongNumber operator/(const LongNumber &cons) - Перегрузка оператора “/”
7. LongNumber operator%(const LongNumber &cons) - Перегрузка оператора “%”
8. LongNumber& operator=(const LongNumber &cons) - Перегрузка оператора “=”
9. bool operator==(const LongNumber &cons) - Перегрузка оператора “==”
10. bool operator>(const LongNumber &cons) - Перегрузка оператора “>”
11. bool operator>=(const LongNumber &cons) - Перегрузка оператора “>=”
12. bool operator<(const LongNumber &cons) - Перегрузка оператора “<”
13. void print() - Функция выписывает в консоль длинное число, нет возвращаемого значения

**Лабораторная работа номер 2**

**“Вектор”**

**Постановка задачи**

Разработать класс **Вектор**.

Класс должен хранить вектор целых чисел заданного размера (от 1 до 20).

Класс должен предоставлять следующие операции:

1) задать размер вектора

2) узнать размер вектора

3) задать компоненту вектора по ее номеру,

4) узнать компоненту вектора по ее номеру

5) вычислить длину вектора

6) найти скалярное произведение двух векторов

7) найти сумму двух векторов одного размера.

Класс должен содержать все необходимые конструкторы, деструктор, оператор присваивания, а также «уметь» выводить себя на консоль.

**Руководство пользователя**

Переменная типа vector создается так же как и переменная стандартного типа без передачи параметра.

**Пример:**

vector t1;

Для того чтобы задать размера вектора вызовите функцию void SetDim(int i) передав в параметр Ваше значение.

**Пример:**

t1. SetDim(5);

Для того чтобы узнать размера вектора вызовите функцию void GetDim().

Для того чтобы задать компоненту по номеру вызовите функцию

void SetCoordinate(int i, int value) передав в первый параметр номер ячейки, а во второй ее значение.

Для того чтобы узнать компоненту вектора по ее номеру вызовите функцию

int GetCoordinate(int i), передав в параметр номер ячейки.

Для того чтобы высчитать длину вектора вызовите функцию float GetLength()

Для того чтобы вычитать скалярное произведение двух векторов вызовите функцию

int Scalar(const vector &number)

Для того чтобы вычислить сумму двух векторов используйте оператор “+”.

**Руководство программиста**

Программа написана при помощи одного класса, который назван vector

В модификаторе доступа private находятся следующие вспомогательные поля:

1. int Coordinates[size] – Основной массив, хранящий информацию компонент вектора.
2. int dim = 0 – Переменная отвечающая за размерность

В модификаторе доступа public находятся следующие вспомогательные поля и методы:

1. vector() – Конструктор по умолчанию
2. vector operator+(const vector &number) – Перегруженный оператор “+”
3. int Scalar(const vector &number)- Функция высчитывает скалярное произведение двух векторов одинаковой длинны, если длинна не совпадает программа выбрасывает исключение, в качестве аргумента принимает вектор, возвращает целое число.
4. float GetLength() – Функция высчитывает длину вектора, элементы массива возводятся в квадрат и складываются, возвращает квадратный корень из полученного результата.
5. void SetDim(int i) – Функция задает размерность принимая в качестве аргумента целое число.
6. int GetDim() – Функция возвращает переменную dim.
7. void SetCoordinate(int i, int value)- Данная функция изменяет значение компоненты вектора по номеру, принимает два целых аргумента, первый из который номер ячейки, второй значение ячейки, нет возвращаемого значения.
8. int GetCoordinate(int i) – Функция возвращает значение компоненты вектора по номеру
9. friend ostream& operator<< (ostream &os, vector &c)- Перегруженный оператор вывода для класса

|  |
| --- |
|  |

**Лабораторная работа номер 3**

**“Пользовательское меню”**

**Постановка задачи**

Разработать класс **Пользовательское меню**.

Класс должен предоставлять одноуровневое меню с заданным числом команд в консольном режиме экрана.

Класс должен содержать необходимые служебные методы (конструкторы, деструктор и пр.).

Класс должен предоставлять следующие операции:

1) задать число команд меню

2) узнать число команд меню

3) задать название пункта меню с указанным номером

4) вывести меню на экран в выбранной позиции окна консоли

5) обеспечить выбор пользователем пункта меню с выдачей выбранного номера

6) выдать номер последнего выбранного пользователем пункта меню.

**Руководство пользователя**

Переменная типа UserMenu создается так же как и переменная стандартного типа без передачи параметра.

Для того, чтобы задать количество команд меню вызовите метод

void SetValueCommands(int \_commands) передав число команд.

Для того, чтобы узнать число команд меню вызовите метод int GetValueCommads()

Для того, задать название пункта меню с указанным номером используйте метод

void SetCommand(int num, string str) передав в параметры номер и название пункта.

Для того, чтобы установить конкретную позицию окна консоли вызовите метод

void SetPositionManual(int \_x, int \_y).

Для того, чтобы установить позицию окна консоли по типу расположения вызовите метод void SetPosition(string str) передав строчку

**Пример:**

menu1.SetPosition("middle");

Для того, чтобы выбрать пункт меню используйте стрелочки на клавиатуре, при нажатии клавиши “Enter” произойдет выбор самого пункта, куда указывает ползунок.

**Руководство программиста**

Программа написано на основе одного класса, который назван UserMenu

В модификаторе доступа private находятся следующие поля и вспомогательные методы:

1. string menu[size]; - Основной массив содержащий само меню.
2. bool trigger - Вспомогательная переменная отвечающая за логику выбора пункта меню.
3. int commands –Число команд меню
4. int push – Переменная для switch
5. int x – Координата по x
6. int y – Координата по y
7. int ammount – Переменная отвечающая за перемещение ползунка
8. void base() Функция заполняющая массив string menu[size] нулевыми элементами.

В модификаторе доступа public находятся следующие вспомогательные поля и методы:

1. UserMenu() – Конструктор по умолчанию, содержит в себе функцию void base().
2. void SetValueCommands(int \_commands)- Устанавливает число команд меню.
3. int GetValueCommads() – Возвращает число команд меню.
4. void SetCommand(int num, string str) – Устанавливает название пункта меню, по указанному номеру, заполняя массив string menu[size].
5. void SetPositionManual(int \_x, int \_y)- Устанавливает расположения меню, если пользователь передаст координаты в консоли.
6. void SetPosition(string str) – Так же устанавливает расположение меню, но при этом нужно передать строку, готовые типы расположения указаны в скобках("default", "left+down", "right+up", "right+down", "middle")
7. void PrintMenu() - Выписывает заполненный массив string menu[size] и ползунок для выбора пунктов.
8. void CallMenu() - Функция обрабатывает саму логику и меню и переключение между пунктами с помощью стрелочек на клавиатуре.
9. int GetCallMenu() - Возвращает последний выбранный пункт меню.

**Лабораторная работа номер 4**

**“Фильмотека”**

**Постановка задачи**

Разработать класс Фильмотека.

Класс должен хранить информацию о фильмах. Каждый фильм описывается следующими данными: название, режиссер, сценарист, композитор, дата выхода в прокат (день, месяц, год), сборы (в рублях). Фильмы хранятся упорядоченно по названию и годам. Данные вводятся на русском языке.

Класс должен содержать необходимые служебные методы.

Класс должен предоставлять следующие операции:

1) добавить фильм

2) изменить данные выбранного фильма

3) найти фильм по названию и году

4) выдать все фильмы заданного режиссера

5) выдать все фильмы, вышедшие в прокат в выбранном году

6) выдать заданное число фильмов с наибольшими сборами

7) выдать заданное число фильмов с наибольшими сборами в выбранном году

8) узнать текущее число фильмов

9) удалить фильм

10) сохранить фильмотеку в файл

11) считать фильмотеку из файла.

**Руководство пользователя**

Для того чтобы создать переменную типа “Фильмотека” создайте переменную типа FilmLibrary.

Для того чтобы добавить в фильмотеку фильм, предварительно создайте переменную типа

Film и вызовите функцию void AddFilm(Film film) передав в параметр ваш фильм.

Для того, чтобы внести изменения фильма в вашей фильмотеке, используйте функцию Film SetChanges(Film film, string str, unsigned long int tmp) или Film SetChanges(Film film, string str, string tmp).

Чтобы выдать все фильмы заданного режиссера вызовите функцию

void SetFilmProducer(string producer).(см. примечание 1)

Чтобы выдать все фильмы заданного года вызовите функцию void SetFilmYear(int \_year).

(см. примечание 1)

Чтобы выдать все фильмы с наибольшими сборами вызовите функцию

void SetFilmFees(int k).(см примечание 1)

Чтобы выдать все фильмы с наибольшими сборами и выбранном году вызовите функцию

void SetFilmFees(int k, int \_year).(см примечание 1)

Чтобы узнать кол-во фильмов в фильмотеке используйте int GetValueFilms().

Чтобы удалить фильм используйте void DeleteFilm(Film tmp = {})

Чтобы сохранить фильмотеку в файл используйте функцию void SaveInFile(string name)

Чтобы считать фильмотеку из файла используйте функцию void GetOutFile(string name)

**Примечание 1:**

Данные методы работают вместе с функцией void PrintFilmTune() которая выписывает фильмы, в зависимости от выбранного параметра

**Руководство программиста**

Программа “Фильмотека” написана при помощи одного класса, названного FilmLibrary, а так же трех вспомогательных структур Date, Facts, Film.

Структура Date состоит из полей int day, int month, int year, обозначающие дату выхода фильма в прокат.

Структура Facts состоит из полей string name, string producer, string screenwriter, string composer, unsigned long int fees. Данная структура хранит все параметры фильма.

Структура Film содержит в себе две выше перечисленные структуры и служит объединением информации, содержит в себе так же перегруженные оператор присваивания, сравнения, неравенства.

В модификаторе доступа private находятся следующие поля:

1. Date date[size] = {}; - массив дат, для всех фильмов
2. Facts stock[size] = {}; - массив информации фильмов для всей фильмотеки
3. Film film1[size] = { \*date,\*stock };- массив фильмов для фильмотеки принимающий в аргументы дату и информацию фильма.(является основным архивом)
4. Film tmp[size] = {}; Неосновной массив фильмов, используется для обработки информации

В модификаторе доступа public находятся следующие поля:

1. void AddFilm(Film film)- Функция добавляет конкретный фильм в фильмотеку
2. Film GetFilmName(string name, int \_year) –Функция выдает фильм по выбранному названию или году, проверок на название или год нету.
3. Film SetChanges(Film film, string str, unsigned long int tmp)-Данная функция изменяет параметры фильма
4. Film SetChanges(Film film, string str, string tmp)-Данная функция изменяет параметры
5. void SetFilmProducer(string producer) - Функция выдает фильмы заданного режиссера, работает в паре с функцией void PrintFilmTune().
6. void SetFilmYear(int \_year) - Функция выдает фильмы заданного года, работает в паре с функцией void PrintFilmTune().
7. void SetFilmFees(int k)- Функция выдает фильмы с наибольшими сборами работает в паре с функцией void PrintFilmTune().
8. void SetFilmFees(int k, int \_year) Функция выдает фильмы с наибольшими сборами и в выбранном году,работает в паре с функцией void PrintFilmTune().
9. int GetValueFilms() – Выдает кол-во фильмов в фильмотеке.
10. void PrintFilmTune()- Выписывает фильмы выбранного типа, см. выше.
11. void DeleteFilm(Film tmp = {})-Удаляет фильм из фильмотеке, оставляя пустую ячейку в основном массиве фильмов.
12. void SaveInFile(string name)-Сохраняет фильмотеку в файл txt, в параметр передается ваше название файла.
13. void GetOutFile(string name)- Считывает фильмотеку из файла, в параметр передается название вашего файла, где содержится фильмотека.

По мимо основных методов, класс использует перегрузки оператора вывода, которые названы как

ostream &operator<<(ostream &os, const Film &film) – Выписывает фильм в консоль

ostream &operator<<(ostream &os, const FilmLibrary &tmp)- Выписывает полностью всю фильмотеку, заполненную фильмами, без пустых элементов.

**Лабораторная работа номер 5**

**“Депозит”**

**Постановка задачи**

Разработать классы Депозит и Процессинговый центр.

Класс Депозит должен имитировать работу интернет-банка в части управления депозитом для зарплатных клиентов банка (тех, чья зарплата перечисляется работодателем на счета работников, открытые в данном банке). Депозит может открываться клиентом на выбранный срок. Возможные варианты: 3 месяца, 6 месяцев, 1 год, 2 года, 3 года. Считать, что процентные ставки по депозитам зависят только от срока депозита и начальной суммы. Диапазоны начальных сумм: до 100 тыс. рублей, от 100 до 500 тыс. рублей, от 500 тыс. до 1 млн рублей, свыше 1 млн рублей. Процентные ставки указываются для срока 1 год (например, 6% годовых по вкладу на 6 месяцев от 100 до 500 тыс. рублей или 6.6% годовых по вкладу на 1 год от 500 тыс. до 1 млн рублей). Считать, что проценты начисляются раз в месяц на начальную сумму депозита и могут быть сняты клиентом в любой момент. Депозит может быть закрыт только по истечении срока, на который он был открыт.

База клиентов хранится в классе Процессинговый центр. Считать, что информация о клиенте состоит из номера зарплатного счета (для упрощения – номер счета от «0001» до «9999»), ФИО владельца, суммы на зарплатном счету, (для упрощения – без копеек), информации о депозите, пароля (произвольная строка, выбранная пользователем, с длиной больше 3 символов). Считать, что авторизация клиента в интернет-банке происходит по номеру зарплатного счета и паролю.

Класс Депозит должен предоставлять следующие операции:

1) авторизовать клиента

2) показать информацию о доступных клиенту депозитах, исходя из суммы на его зарплатном счету

3) проверить наличие открытого депозита

4) открыть депозит (переведя указанную сумму с зарплатного счета клиента на депозит)

5) показать состояние депозита

6) снять проценты (переведя их на зарплатный счет клиента)

7) закрыть депозит (переведя всю накопленную сумму на зарплатный счет клиента).

Все операции должны сопровождаться необходимыми проверками на указанные выше ограничения.

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

**Руководство пользователя**

При запуске программы, пользователю предлагается ввести логин и пароль, для входа в систему, при этом аккаунт пользователя должен находится в базе процессингового центра. Если аккаунта не существует, создайте переменную типа Client и заполните все необходимые поля, затем вызовите метод класса процессинговый центр для добавления в базу данных.  
Далее если все условия соблюдены, логин и пароль введены верны, появится диалоговое окно, показывающее зарплатный счет клиента, и список возможностей, которые ему предоставлены см. рис. 1.

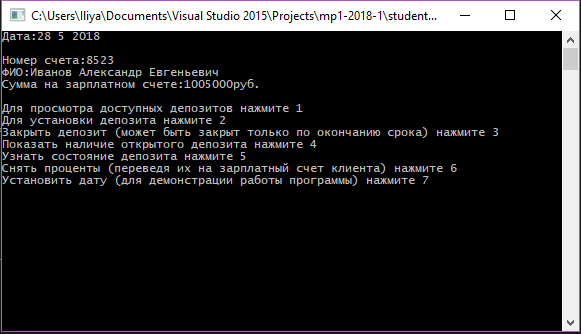


Рис. 1 Пользовательский интерфейс

На рисунке 1 предоставлен весь список возможностей, для выбора конкретного пункта, необходимо нажать соответствующую клавишу.

1. Произойдет показ доступных депозитов, в соответствии со средствами на вашем зарплатном счете.
2. Вы откроете депозит, на выбранную сумму и предложенный срок.
3. Закроет ваш депозит, если таковой есть.
4. Показывает есть ли на вашем счету открытый депозит.
5. Если у вас есть открытый депозит, покажет ваше накопления по вкладу.
6. Снимает накопленные проценты на зарплатный счет клиента.
7. Устанавливает дату, для работы с программой.

**Руководство программиста**

Программа “Депозит” написана на основе дружественных классов, которые названы как

ProcessingCentr и Bank, а так же некоторых структур которые названы deposit и Client

Структура deposit представляет собой набор данных о депозите, а именно статус(открыт/закрыт), дату открытия, дату закрытия.

Структура Client представляет собой набор основных данных клиента, его пароль, логин, инициалы, зарплатный счет, а так же информацию о депозите

Класс ProcessingCentr представляет собой архив данных клиентов и разработан в минимальном необходимом варианте, содержит в себе вектор клиентов и метод AddClient(Client tmp) который добавляет в массив данных нового клиента.

Класс Bank является основным классом и выполняет все основные вычисления, а также обработку информации клиентов.

В модификаторе доступа private класса Bank находятся следующие поля и методы:

1. ProcessingCentr box – Является переменной процессингового центра
2. Client clt = {} – Переменная для основной работы клиента с программой
3. bool login – Проверка на вход
4. bool end – Проверка на наличие закрытия депозита
5. unsigned long long int savings – Накопления депозита
6. string srok[5] – Массив содержащий сроки по депозитам
7. double percent[20] Массив содержащий проценты по депозитам
8. unsigned long long int sum[3] Массив содержащий ограничения по суммам
9. void rewrite(Client tmp) – Перезапись и сохранение изменений информации о клиенте.
10. unsigned long long int Calculation() – Функция выполняющая основные вычисления с процентами, сравнивает даты и выполняет необходимые подсчеты, в зависимости от случая.

В модификаторе доступа public класса Bank находятся следующие поля и методы:

1. Bank(ProcessingCentr \_box) – Конструктор по умолчанию, связывающий процессинговый центр и сам класс Bank.
2. Client SignIn(unsigned int account, string str) – Функция отвечающая за вход в систему, содержит в себе все необходимые проверки на авторизацию.
3. void ShowAvaiDeposits() – Функция показывает доступные депозиты для пользователя, проверяет зарплатный счет клиента и выдает на основаниях этого соответствующую информацию из массивов данных.
4. void OpenDeposit(unsigned long long int money, string time) – Функция, позволяющая открыть депозит на определенный срок и сумму, так же содержит в себе проверку на состояние счета клиента
5. void StateDep() – Проверяет состояние открытого депозита, просматривая переменную, которая содержится в структуре deposit.
6. void ShowStatusOfDeposit() – Функция проверяет наличие депозита, выполняет вычисления если депозит открыт и выдает информацию в зависимости от параметров даты, срока, процентной ставки.
7. void GetPercent() – Функция обналичивает проценты по вкладу, переводя их на зарплатный счет клиента.
8. void CloseDeposit() –Позволяет закрыть депозит, но только по окончанию срока.

Также для каждой структуры и класса написана перегрузка оператора “<<”

**Лабораторная работа номер 6**

**“Змейка”**

**Постановка задачи**

Разработать систему классов и реализовать с ее помощью игру Змейка.

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, перед началом игры, пользователя спрашивают, какую максимальную длину будет иметь змейка в конце игры. Пользователь задав длину переходит к следующему пункту, выбора скорости игры. Правила просты, если пользователь наберет хвост длинны, равной той, которую он указал вначале, игра будет считаться выигранной.

Когда пользователь ввел все параметры игры, появится карта, на которой будут показаны сама змейка, а так же игровая карта. Для начала игры нажмите клавишу “Enter”. При нажатии данной клавиши произойдет движение змейки влево, для управления используйте клавиши стрелок на клавиатуре. Цель игры заключается в поедании фрукта, съев один фрукт, хвост увеличится на одну позицию. Снизу указан ваш счет.

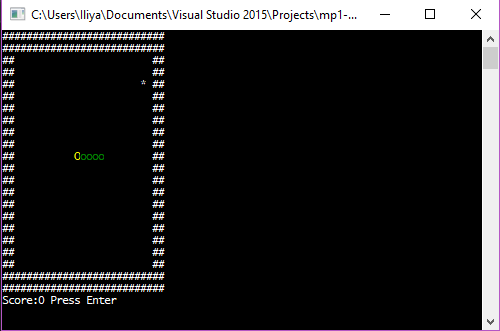


Рис. 1 Игра

**Руководство программиста**

Программа змейка написана с помощью систем нескольких классов, а именно Snake, Interface, Food.

В модификаторе доступа private класса Snake расположены следующие поля:

1. int maxtail – переменная отвечающая за максимальный хвост, выбранный пользователем
2. int tail – Размер хвоста по умолчанию
3. enum Moving { Stop, Left, Right, Up, Down }dir – Переменная перечислений, служит для управления змейкой
4. struct body - состоит из двух переменных, координаты по x и по y, по сути отвечает за хранение координат хвоста.

В модификаторе доступа public класса Snake расположены следующие поля:

1. Snake() – Конструктор по умолчанию, задает начальное расположение змейки, а именно по середине карты
2. void SetMaxTail(int \_maxtail) – Устанавливает максимальный размер хвоста змейки
3. void DrawSnake() – Рисует змейку, функция настроена так, что прорисовка головы змейки выделена отличным цветом от туловища змеи, постоянно очищает экран в зависимости от передвижения.
4. void crawl() – Вспомогательная функция для логичного передвижения змеи, по сути заставляет хвост полсти за головой змеи.
5. void control() – Функция написана с помощью оператора switch и представляет собой итерацию параметров переменной dir.
6. void kb() – Отвечает за контролирование змейки, а так же осуществляет старт игры
7. void Check() – Функция каждый раз проверяет проиграл или выиграл игрок.

Класс Interface отвечает за прорисовку игрового поля.

В модификаторе доступа private класса Interface расположены следующие поля:

1. int x – Координата прорисовки карты по x
2. int y - Координата прорисовки карты по y
3. int score – Переменная выводящая текущий счет игрока

В модификаторе доступа public класса Interface расположены следующие поля:

1. void ShowMap() – Единственный метод класса, который прорисовывает карту, а так же показывает счет, набранный в течении игрового процесса.

Класс Food отвечает за создание фрукта для змейки, а так же для рисования его на экране.

В модификаторе доступа private класса Food расположены следующие поля и методы:

1. int eatx – Координата фрукта по x
2. int eaty – Координата фрукта по y
3. bool did – Вспомогательная переменная логики
4. void create() – Функция отвечающая за генерацию фруктов, причем генерация осуществялется только в том месте, где не присутствует само тело змейки.

В модификаторе доступа public класса Food расположены следующие методы:

1. void ShowEAT() – Метод который отслеживает состояние созданного фрукта.

Помимо созданных классов в программе присутствуют вспомогательные функции:

1. void DisableShowConsoleCursor() – Отключает мигающий ползунок в консоли
2. void cls(HANDLE hConsole) – Очищает экран, замена стандартной функции system(“cls”)
3. void setposition(int x, int y) – Перемещает ползунок консоли в указанное место
4. void GameSpeed(int speed) – Задает скорость игры

**Приложение**

#include <iostream>

#include <string>

#include <Windows.h>

#include <cstdlib>

#include <conio.h>

#include <random>

using namespace std;

#define n 25//ширина

#define m 20//высота

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

COORD position;

class Interface;

class Food;

class Snake;

class Game;

bool Gameover = false;

void DisableShowConsoleCursor()//отключение мигающей штуки

{

CONSOLE\_CURSOR\_INFO showCursor;

GetConsoleCursorInfo(hConsole, &showCursor);

showCursor.bVisible = false;

SetConsoleCursorInfo(hConsole, &showCursor);

}

void cls(HANDLE hConsole)//Альтернатива system("cls") чтобы не моргал экран

{

COORD coordScreen = { 0, 0 };

DWORD cCharsWritten;

CONSOLE\_SCREEN\_BUFFER\_INFO csbi;

DWORD dwConSize;

if (!GetConsoleScreenBufferInfo(hConsole, &csbi))

return;

dwConSize = csbi.dwSize.X \* csbi.dwSize.Y;

if (!FillConsoleOutputCharacter(hConsole, (TCHAR) ' ',

dwConSize, coordScreen, &cCharsWritten))

return;

if (!GetConsoleScreenBufferInfo(hConsole, &csbi))

return;

if (!FillConsoleOutputAttribute(hConsole, csbi.wAttributes,

dwConSize, coordScreen, &cCharsWritten))

return;

SetConsoleCursorPosition(hConsole, coordScreen);

}

void setposition(int x, int y)

{

position.X = x;

position.Y = y;

SetConsoleCursorPosition(hConsole, position);

}

void GameSpeed(int speed)

{

Sleep(speed);

}

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

};

class Snake

{

int maxtail;

int tail = 5;

enum Moving { Stop, Left, Right, Up, Down }dir;

struct body

{

int x;

int y;

}snake[(n - 2)\*(m - 2) + 1];

public:

Snake()

{

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

{

dir = Stop;

snake[i].x = n / 2 + i;

snake[i].y = m / 2;

}

}

void SetMaxTail(int \_maxtail)

{

if (\_maxtail>tail)

maxtail = \_maxtail;

else throw;

}

void DrawSnake()//рисуем змейку

{

cls(hConsole);

setposition(snake[0].x, snake[0].y);

SetConsoleTextAttribute(hConsole, (WORD)((Black) | Yellow));

cout << "O";

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

{

setposition(snake[i].x, snake[i].y);

SetConsoleTextAttribute(hConsole, (WORD)((Black) | Green));

cout << "o";

}

}

void crawl()//хвост ползет за головой

{

if (dir != Stop)

{

for (int i = tail; i > 0; i--)

{

snake[i].x = snake[i - 1].x;

snake[i].y = snake[i - 1].y;

}

}

}

void control()//функция управления

{

switch (dir)

{

case Left:snake[0].x--; break;

case Right:snake[0].x++; break;

case Up:snake[0].y--; break;

case Down:snake[0].y++; break;

}

}

void kb()

{

if (\_kbhit())

{

switch (\_getch())

{

case 72:if (dir != Down&&dir != Stop)dir = Up; break;

case 80:if (dir != Up&&dir != Stop)dir = Down; break;

case 75:if (dir != Right&&dir != Stop)dir = Left; break;

case 77:if (dir != Left && dir != Stop)dir = Right; break;

case 13:if (dir == Stop)dir = Left; break;

}

}

}

void Check()

{

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

{

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

{

Gameover = true;

cls(hConsole);

cout << "YOU LOSE";

}

}

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

{

if (snake[0].x == i && (snake[0].y == 1 || snake[0].y == m))

{

Gameover = true;

cls(hConsole);

cout << "YOU LOSE";

}

}

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

{

if (snake[0].y == i && (snake[0].x == 1 || snake[0].x == n))

{

Gameover = true;

cls(hConsole);

cout << "YOU LOSE";

}

}

if (tail == maxtail)

{

Gameover = true;

cls(hConsole);

cout << "YOU WIN";

}

}

friend class Food;

friend class Interface;

}s;

class Interface

{

int x;

int y;

int score;

public:

void ShowMap()//рисуем карту

{

SetConsoleTextAttribute(hConsole, (WORD)((Black) | White));

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

{

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

{

setposition(i, j);

cout << "#";

}

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

{

setposition(n + j, i);

cout << "#";

}

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

{

setposition(j, i);

cout << "#";

}

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

{

setposition(i, m + j);

cout << "#";

}

}

setposition(0, m + 2);

cout << "Score:" << score << " ";

if (s.dir == s.Stop)

cout << "Press Enter" << endl;

}

friend class Food;

}A;

class Food

{

private:

int eatx;

int eaty;

bool did = true;

void create()

{

random\_device random\_device;

mt19937 generator(random\_device());

uniform\_int\_distribution<int> randx(2, n - 2);

uniform\_int\_distribution<int> randy(2, m - 2);

eatx = randx(generator);

eaty = randy(generator);

for (int i = 0; i < s.tail; i++)

{

if (eatx == s.snake[i].x&&eaty == s.snake[i].y)

create();

}

}

public:

void ShowEAT()

{

if (did)

{

create();

did = false;

}

else

{

setposition(eatx, eaty);

cout << "\*";

if (s.snake[0].x == eatx&&s.snake[0].y == eaty)

{

A.score += 10;

s.tail++;

did = true;

}

}

}

}eat;

void mainfunction()

{

int lenth;

int speed;

cout << "Enter lenth" << endl;

cin >> lenth;

s.SetMaxTail(lenth);

cout << "Enter speed(recommended speed 100)" << endl;

cin >> speed;

while (!Gameover)

{

s.DrawSnake();//рисуем змею

A.ShowMap();//рисуем карту

eat.ShowEAT();

s.crawl();

GameSpeed(speed);

s.kb();

s.control();

s.Check();

}

}

int main()

{

DisableShowConsoleCursor();

mainfunction();

system("pause>>NULL");

}

**Заключение**

Объектно ориентированное программирование является важным аспектом в современном программировании.

Программа представляет собой набор объектов, имеющих состояние и поведение. Объекты взаимодействуют посредством сообщений.

Несмотря на отдельные критические замечания в адрес ООП, в настоящее время именно эта парадигма используется в подавляющем большинстве промышленных проектов. Однако нельзя считать, что ООП является наилучшей из методик программирования во всех случаях.

**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