**СОДЕРЖАНИЕ**

ВВЕДЕНИЕ………………………………………………………………………………………5

1ПОСТАНОВКА ЗАДАЧИ……...………………………………………………………………6

2СТРУКТУРА ВХОДНЫХ И ВЫХОДНЫХ ДАННЫХ……………….……………….…….7

3РАЗРАБОТКА АЛГОРИТМОВ……………………….………………………………………8

3.1Схема алгоритмов…...………………………………..………………………………………8

3.2Алгоритмы по шагам……………………………………….………………………………...8

4 ДИАГРАММА КЛАССОВ…………………………………………………………10

5 ОПИСАНИЕ КЛАССОВ………………………………………………………………...…...11

6 КОД ПРОГРАММЫ………………………………………………………..…………………17

7 РЕЗУЛЬТАТЫ РАБОТЫ ПРОГРАММЫ…………...………………………………….…...18

ЗАКЛЮЧЕНИЕ………………...…………………………………………………….…..……..19

ПРИЛОЖЕНИЕ A(схемы алгоритмов)………………………… …………………….…..…..20

ПРИЛОЖЕНИЕ Б(диаграмма классов)…………………… ……………………………….....21

ПРИЛОЖЕНИЕ В(код программы)…………… …………………………….….…....22

ПРИЛОЖЕНИЕ Г(скриншоты работы программы)…………… ……………….……...24

СПИСОК ЛИТЕРАТУРЫ…………………………….……………………….……..27

**ВВЕДЕНИЕ**

C++ — компилируемый статически типизированный язык программирования общего назначения. Несмотря на то, что данный язык программирования возник в 1980-м году, он остается актуальным и востребованным до сих пор. Поддерживает процедурное, объектно-ориентированное и обобщенное программирование, обеспечивает модульность, раздельную компиляцию, обработку исключений, абстракцию данных, объявление типов (классов) объектов, виртуальные функции. Стандартная библиотека включает, в том числе, общеупотребительные контейнеры и алгоритмы. C++ сочетает свойства как высокоуровневых, так и низкоуровневых языков.

С++ широко используется для разработки прикладных программ, операционных систем, приложений для встраиваемых систем, драйверов устройств и игр. На нем реализованы такие популярные программы, как CCleaner, Google Chrome, Havok Vision Engine, Steam и многие другие. Синтаксис унаследован от языка С, что в свою очередь облегчает освоение данного языка.

**1 ПОСТАНОВКА ЗАДАЧИ**

Программа должна иметь удобный пользовательский интерфейс с необходимыми пунктами меню. Информация должна храниться в различных файлах, связанных определенным образом. Реализовать функции: добавление, редактирование, удаление данных о доходах и расходах пользователя, вывод на экран.

Разработать иерархию классов с использованием наследования. Разработать и использовать в программе классы контейнеров и итераторов (свои и STL). Производить обработку исключительных ситуаций. При реализации операции редактирования, добавления, удаления информации необходимо предусмотреть операцию отмены последних действий.

Для реализации игры используется объектно-ориентированный язык программирования C++, среда разработки Microsoft Visual Studio 2017. Ограничения на использование других операционных систем нет. Приложение написано на ОС Windows 10.

**2** **СТРУКТУРА ВХОДНЫХ И ВЫХОДНЫХ ДАННЫХ**

В программе используется 3 текстовых файла:

-Файл user.txt. Хранит данные о текущем пользователе (логин и изначальный баланс).

|  |
| --- |
| CourseTestUser1 256 |

-Файл income.txt. Хранит данные о доходах (сумма дохода и его источник).

|  |
| --- |
| 340 1  15 2  60 4 |

-Файл consum.txt. Хранит данные о расходах (сумма расхода, тип расхода и является ли этот расход дачей денег в долг).

|  |
| --- |
| 15 1 1  150 5 1  35 8 1 |

Также в ходе работы программы для хранения промежуточных данных используются STL-контейнеры и контейнер List, написанный вручную.

При изменении данных в программе, происходит изменение в файле после закрытия программы.

**3 РАЗРАБОТКА АЛГОРИТМОВ**

**3.1 Схема Алгоритмов**

3.1.1 Алгоритм функции **FileSaveConsum(List<X> &tmp, const char \*file)**.Позволяет сохранять данные о расходах в файл;

Представлен в Приложении A.

3.1.2 Алгоритм функции **getBalance (List <Income> &income, List <Consum> &consum)**. Позволяет получить текущий баланс пользователя;

Представлен в Приложении A.

**3.2 Алгоритмы по шагам**

Для алгоритма по шагам рассмотрим метод FileSaveConsum(List<X> &tmp, const char \*file)шаблонного класса File.

Шаг 1: Начало;

Шаг 2: Открываем текстовый файл on для записи данных. Если файл не открылся, выводим сообщение “Невозможно открыть файл!” и переходим к шагу 5;

Шаг 3: С помощью цикла записываем в текстовый файл значения суммы расхода, данные о типе расхода и данные о флажке долга с самого начала списка до его конца;

Шаг 4: Записываем в файл “-1”, являющуюся условным окончанием записи и предназначенную для повторного считывания данных из файла;

Шаг 5: Конец.

Для алгоритма по шагам рассмотрим метод **getBalance (List <Income> &income, List <Consum> &consum, double tempBalance = 0)** шаблонного класса List.

Шаг 1: Начало;

Шаг 2: Если размер списка доходов больше нуля, то в цикле от начала списка и до конца увеличиваем переменную tempBalance на величину sum. Иначе пропускаем этот шаг;

Шаг 3: Если размер списка расходов больше нуля, то в цикле от начала списка и до конца уменьшаем переменную tempBalance на величину sum. Иначе пропускаем этот шаг;

Шаг 4: Возвращаем из функции переменную tempBalance;

Шаг 5: Конец.

**4 ДИАГРАММА КЛАССОВ**

Предоставляется в Приложении Б.

**5 ОПИСАНИЕ КЛАССОВ**

1. Класс файлов предназначен для работы с файлами;

template <class X>

class File

{

public:

File() {}; - конструктор по умолчанию

~File() {}; - деструктор

void FileSaveIncome (List<X> &tmp, const char \*file); - метод записи доходов

void FileSaveConsum(List<X> &tmp, const char \*file); - метод записи расходов

void FileOpenIncome(List<X> &tmp, const char \*file); - метод чтения доходов

void FileOpenConsum(List<X> &tmp, const char \*file); - метод чтения расходов

};

1. Базовый класс изменения баланса;

class BalanceChange

{

protected:

double sum; - сумма изменения баланса

public:

BalanceChange(); - конструктор по умолчанию

~BalanceChange(); - деструктор

double getSum(); - метод получения суммы

void setSum(double \_sum); - метод присвоения суммы

};

1. Класс расхода

class Consum : public BalanceChange

{

int type; - тип дохода

bool debt; - флажок долга

public:

Consum(); - конструктор по умолчанию

Consum(const Consum &tmp); - конструктор копирования

Consum(int \_type, double \_sum, bool \_debt); - конструктор с параметрами

~Consum(); - деструктор

double getSum(); - метод получения суммы расхода

int getType(); - метод получения типа расхода

bool getDebt(); - метод получения флажка долга

void setSum(double \_sum); - метод присвоения суммы расхода

void setType(int \_type); - метод присвоения типа расхода

void setDebt(int \_debt); - метод присвоения значения флажка долга

friend void operator >> (istream & is, Consum & tmp); - перегруженный оператор ввода

friend void operator << (ostream & os, Consum & tmp); - перегруженный оператор вывода

friend bool operator== (Consum tmp1, Consum tmp2); - перегруженный оператор сравнения

void operator = (const Consum & tmp); - перегруженный оператор присвоения

};

1. Класс дохода

class Income : public BalanceChange

{

int type; - тип дохода

public:

Income(); - конструктор по умолчанию

Income(const Income &tmp); - конструктор копирования

Income(int \_type, double \_sum); - конструктор с параметрами

~Income(); - деструктор

double getSum(); - метод получения суммы дохода

int getType(); - метод получения типа дохода

void setSum(double \_sum); - метод присвоения суммы дохода

void setType(int \_type); - метод присвоения типа дохода

friend void operator >> (std::istream & is, Income & tmp); - перегруженный оператор ввода

friend void operator << (std::ostream & os, Income & tmp); - перегруженный оператор вывода

friend bool operator== (Income tmp1, Income tmp2); - перегруженный оператор сравнения

void operator = (const Income & tmp); - перегруженный оператор присвоения

};

1. Класс пользователя

class User

{

string username; - имя пользователя

double balance; - баланс пользователя

public:

User(); - конструктор по умолчанию

User(char\* currentName, double currentMoney); - конструктор с параметрами

User(const User &tmp); - конструктор копирования

~User(); - деструктор

string getUsername(); - метод получения имени пользователя

double getBalance(); - метод получения баланса пользователя

void setUsername(string tmp); - метод присвоения имени пользователю

void setBalance(double\_balance); - метод присвоения баланса пользователю

string InputCharCheck(string input); - метод ввода строки с проверкой

bool CheckCorrect(string input, bool fl); - метод проверки правильности ввода строки

friend void operator >> (std::istream & is, User & tmp); - перегрузка оператора ввода

};

1. Базовый класс исключений

class Exception

{

protected:

int error; - номер ошибки

public:

Exception(); - конструктор по умолчанию

Exception(int n); - конструктор копирования

};

1. Класс исключений ввода

class InputException :public Exception

{

public:

InputException():Exception(); - конструктор по умолчанию

InputException(int n):Exception(n); - конструктор копирования

void NumError(); - метод вывода описания ошибки

};

1. Класс исключений ошибок в списке

class ListException :public Exception

{

public:

ListException() : Exception(); - конструктор по умолчанию

ListException(int n) : Exception(n); - конструктор копирования

void NumError(); - метод вывода описания ошибки

};

1. Шаблонный класс двусвязного списка

template<typename T>

class List

{

protected:

template<typename T> friend class Iterator;

Node<T> \*head; - указатель на голову списка

Node<T> \*tail; - указатель на хвост списка

long amount; - длина списка

public:

List(); - конструктор по умолчанию

~List(); - деструктор

long size(); - метод получения длины списка

void push\_head(T input\_object); - метод добавления элемента в список со стороны головы

void push\_tail(T input\_object); - метод добавления элемента в список со стороны хвоста

T pop\_head(); - метод удаления элемента списка со стороны головы

T pop\_tail(); - метод удаления элемента списка со стороны хвоста

T &operator[](long num); - метод доступа к произвольному элементу списка

void delete\_element(Node<T> \*node); - метод удаления элемента списка

double getBalance(List<Income> &income, List<Consum> &consum); - метод получения текущего баланса пользователя

Node<T> \*begin(); - метод для получения головы списка

Node<T> \*end(); - метод для получения хвоста списка

void List<T>::output(); - метод вывода списка на экран

void List<T>::output\_for\_type(int \_type); - метод вывода списка на экран по типу

};

1. Класс итератора

template <typename T>

class Iterator

{

Node<T> \*current; - звено

public:

Iterator(); - конструктор по умолчанию

Iterator(List<T> &container); - конструктор копирования

Iterator(Iterator<T> &iter); - перегруженный конструктор копирования

Iterator(Node<T> \*node); - перегруженный конструктор копирования

~Iterator(); - деструктор

Node<T> \*get\_node(); - метод получения узла из итератора

bool operator++(int i); - перегрузка инкремента (сдвиг вперёд)

bool operator--(int i); - перегрузка декремента (сдвиг назад)

T &operator\*(); - получение значения из итератора

bool operator==(Iterator &iter); - перегрузка сравнения

bool operator!=(Iterator &iter); - перегрузка неравенства

};

**6 КОД ПРОГРАММЫ**

Предоставляется в Приложении В.

**7 РЕЗУЛЬТАТЫ РАБОТЫ ПРОГРАММЫ**

Предоставляется в Приложении Г.

**ЗАКЛЮЧЕНИЕ**

В результате выполнения курсового проекта было разработано приложение для учёта личных доходов и расходов, позволяющее пользователю вводить различную информацию, выводить ее на экран, изменять, удалять, добавлять, а также, в случае допущения ошибки пользователем, отменять последние действия. Были закреплены знания в области ООП. Использовались среда разработки Visual Studio 2017 и операционная система Windows 10. К достоинствам программы можно отнести простой и понятный интерфейс, что в свою очередь обеспечивает удобство пользования для обычных пользователей. В дальнейшем планируется усовершенствование программы, а именно усовершенствование интерфейса и добавление новых функций, поддержка многопользовательности. После освоения в ходе учебного курса основ и принципов работы с базами данных планируется переход с системы хранения информации в текстовых файлах на хранение в базе данных, что предоставит возможность добавления информации нескольким пользователям одновременно и позволит увеличить скорость работы программы.

Системные требования:

-Операционная система Windows 98, XP, Vista,7, 8.1, 10;

-Процессор Intel Pentium III 1 GHz;

-Оперативная память 2 гб;

-Свободное место на жестком диске: 5 Мб;

**ПРИЛОЖЕНИЕ A**

**(схемы алгоритмов)**

**ПРИЛОЖЕНИЕ Б**

**(диаграмма классов)**

**ПРИЛОЖЕНИЕ В**

**(код программы)**

**ПРИЛОЖЕНИЕ Г**

**(скриншоты работы программы)**

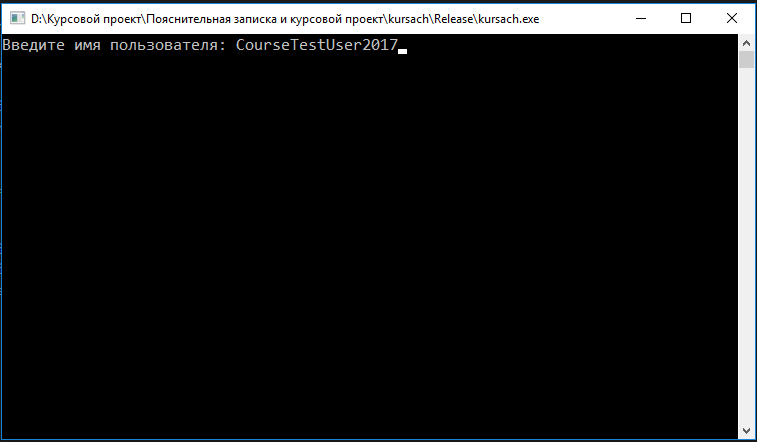


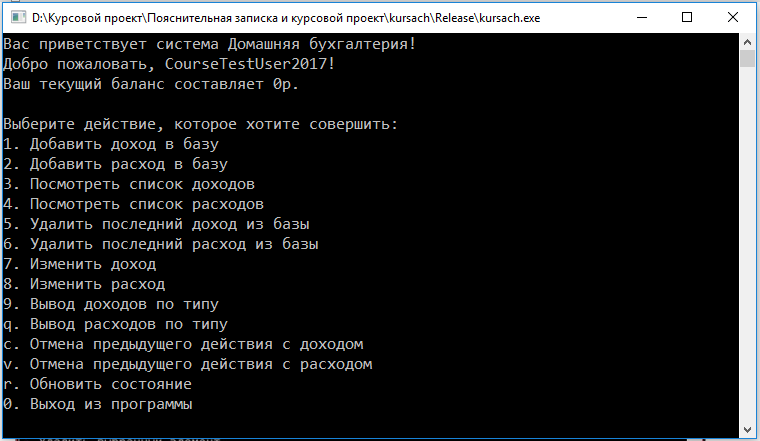
Рисунок 1 – Окно ввода имени пользователя. 

Рисунок 2 – Главное меню.

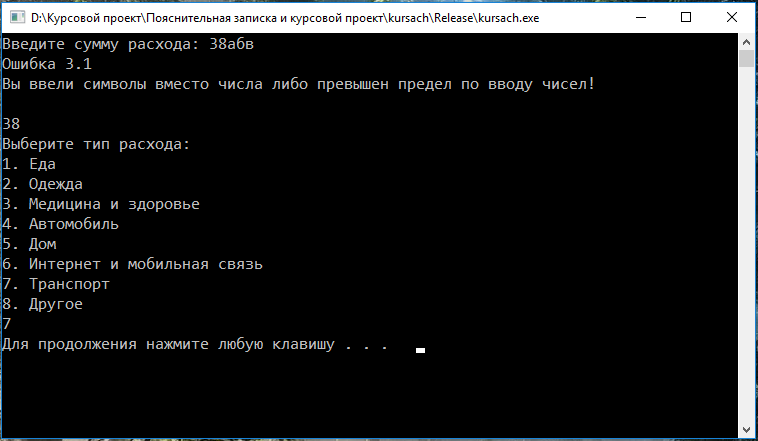


Рисунок 3 - Добавление данных о расходе (с обработкой исключительных ситуаций).

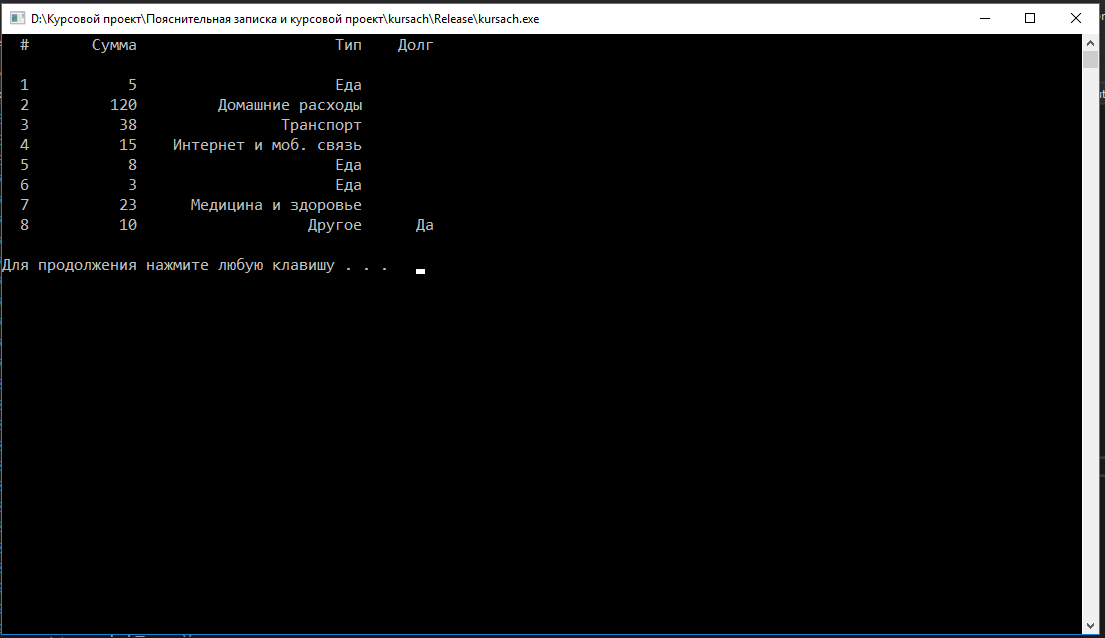


Рисунок 4 - Отображение информации о расходах на экран.

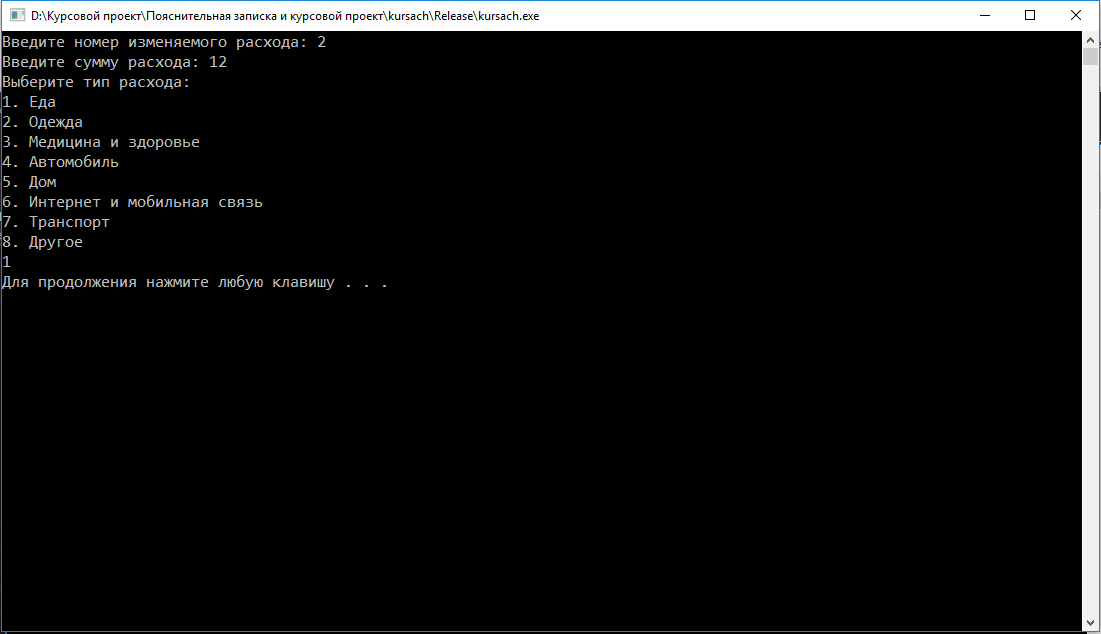


Рисунок 5 – Изменение информации.

**СПИСОК ЛИТЕРАТУРЫ**

**1.** Луцик Ю. А. Объектно-ориентированное программирование на языке С++: учеб. пособие /Ю. А. Луцик, В. Н. Комличенко. – Минск: БГУИР, 2008. –   
266 с.  
**2.** Герберт, Ш. Самоучитель C++/Ш. Герберт. Санкт-Петербург 2003г.   
**3.** Страуструп, Б. Язык программирования С++/Б. Страуструп.; 3-е изд. – СПб.; 1999 – 991 с.  
**4.** Стефенс Д. Р. С++. Сборник рецептов/Д.Р. Стефенс – КУДИЦ-ПРЕСС, 2007. – 624 с.