Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Петрозаводский государственный университет»

Физико-технический институт

Кафедра информационно-измерительных систем и физической электроники

КОМАНДНОЕ ЗАДАНИЕ ПО КУРСУ “ТЕХНОЛОГИЯ ПРОГРАММИРОВАНИЯ”

Командная работа

Авторы работы:

студенты группы 21316

Кошкин В.В.

Коваль В.С.

Куксин В.В.

Научный руководитель:

канд. физ.-мат. наук, доцент

Бульба А.В.

Петрозаводск 2021

**Цель работы:** в рамках “Унифицированного процесса разработки ПО” смоделировать ситуацию выполнения заказа на разработку программного обеспечения. В нашем случае был выбран такой сюжет для ПО, как прачечная.

**Среда разработки:** Visual Studio 2019

**Язык программирования:** С++

**Заголовочные и исполнительные файлы:**

1. UserInterface.h является главным заголовочным файлом, на котором держится вся программа
   1. UserInterface.cpp описывает единственный метод, который отвечает за взаимодействие пользователя с ПО (иначе говоря, предоставляет меню программы)
2. Client.h отвечает за поля (свойства) класса клиента
   1. Client.cpp описывает геттеры (получение ФИО клиента и прочих полей)
3. ClientInputScreen.h отвечает за добавления клиента
   1. ClientInputScreen.cpp описывает единственный метод, который как раз-таки отвечает за добавление клиента (ввод ФИО клиента и прочее)
4. ListOfClients.h отвечает за хранение клиентов (в нашем случае они хранятся в STL-контейнере, как список)
   1. ListOfClients.h описывает методы, которые отвечают за добавление клиента в список, удаление клиента из этого списка, а также отвечает за вывод таблицы клиентов с их данными
5. Expense.h отвечает за поля(свойства) класса расходы
   1. Expense.cpp описывает геттеры (получение категории расходов и прочих полей)
6. ExpenseInputScreen.h отвечает за добавление расходов
   1. ExpenseInputScreen.cpp описывает единственный метод, который как раз-таки отвечает за добавление расходов (ввод категории расходов и прочее)
7. ExpenseRecord.h отвечает за хранение расходов (в нашем случае они хранятся в STL-контейнере, как вектор)
   1. ExpenseRecord.cpp описывает методы, которые отвечают за вывод таблиц расходов и годового отчёта, добавление расходов в вектор.
8. Income.h отвечает за подсчёт прибыли
   1. Income.cpp содержит метод, который отвечает за вывод таблицы доходов, а также есть сеттер и геттер, с помощью которых производится подсчёт прибыли по каждому месяцу, а также итоговую прибыль
9. Laundry.cpp содержит функцию main, с которой начинается запуск самой программы

**Процесс разработки:**

1. К нам обратился частный предприниматель с просьбой создания программного обеспечения с целью облегчения управления его бизнеса. Сам бизнес представляет собой прачечную. По словам заказчика программное обеспечение должно ввести статистику клиентов, эта статистика должна содержать такие данные как ФИО клиента, номер телефона клиента (для того, чтобы можно было связаться с ним), тип услуги (стирка/химчистка белья, одежды, игрушек, …, а также продажа специальных средств, такие как стиральный порошок, отбеливатель и т.п.), дата взятия заказа, а также стоимость самой услуги. Также это программное обеспечение должно ввести статистику доходов и расходов бизнеса.
2. Материалы (данные), которые предоставил нам заказчик:

*Табл.1 Данные клиента.*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ФИО клиента | Номер телефона | Тип услуги | Дата взятия заказа | Стоимость услуги |

*Табл.2 Доход.*

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Месяц | Янв | Фев | Март | Апр | Май | Июнь | Июль | Авг | Сент | Окт | Ноя | Дек |
| Доход |  |  |  |  |  |  |  |  |  |  |  |  |

*Табл. 3 Расходы*

|  |  |  |  |
| --- | --- | --- | --- |
| Дата | Получатель | Сумма | Категория |
|  |  |  |  |

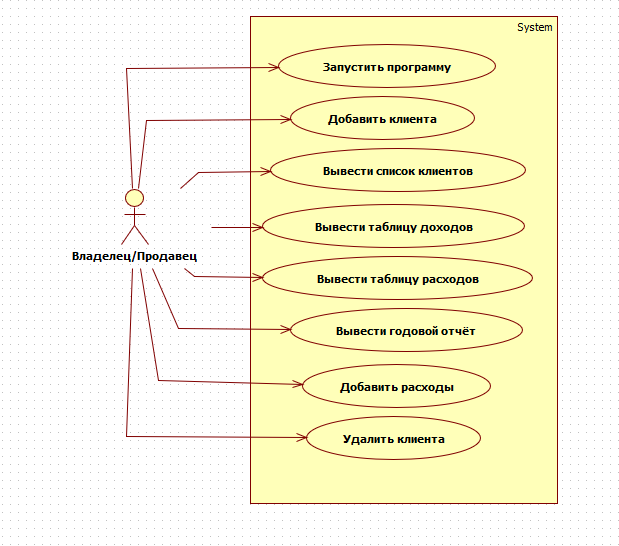
*Таблица.4 Годовой отчет*

|  |  |  |
| --- | --- | --- |
| **Доходы** | | |
|  | Прибыль |  |
| **Расходы** | | |
|  | Коммунальные услуги |  |
|  | Реклама |  |
|  | Ремонт |  |
|  | Снабжение |  |
|  | Страховка |  |
|  | Аренда помещения |  |
| **Чистая прибыль** |  | |

1. Действующий субъект - Владелец

В обязанности роли “Владелец” входит:

1. Вносить клиента в базу, а также заказ этого клиента
2. Вносить расходы в базу
3. Список вариантов использования:



1. Документации к вариантам использования:
2. “Запустить программу”

Запуск программы осуществляет вывод на экран меню, из которого Владелец/Продавец может выбрать нужное действие (выбрать пункт меню).

1. “Добавить клиента”

На экране должно отобразиться сообщение, в котором программа просит пользователя ввести ФИО клиента, его номер телефона, тип услуги, дату взятия заказа, а также саму стоимость услуги. Эта информация будет занесена в таблицу.

При неправильном вводе даты или месяц программа попросит сделать повторный ввод этих параметров.

1. “Вывести список клиентов”

Программа выведет на экран таблицу клиентов, состоящую из пяти столбцов: ФИО клиента, его номер телефона, тип услуги, дату взятия заказа, а также саму стоимость услуги.

1. “Вывести таблицу доходов”

Программа выведет на экран таблицу доходов, состоящую из строк: Месяц, доход. Там будет отображен доход от прачечной за каждый месяц.

1. “Вывести таблицу расходов”

Программа выведет на экран таблицу, состоящую из четырех столбцов: дата и месяц, получатель, сумма, категория.

1. “Вывести годовой отчёт”

Программа выведет на экран таблицу, в которой будет отображена прибыль за год, без вычета расходов. Расходы, где расписаны все затраты, куда и сколько средств было затрачено. А также будет выведена чистая прибыль

1. “Добавить расходы”

На экране должно отобразиться сообщение, в котором программа просит пользователя ввести дату и месяц, получателя, сумму выплаты, категорию расходов. Эта информация будет занесена в таблицу.

При неправильном вводе даты или месяц программа попросит сделать повторный ввод этих параметров.

1. “Удалить клиента”

На экране должно отобразиться сообщение, в котором программа просит пользователя ввести ФИО клиента, которого следует удалить из базы клиентов.

1. Диаграммы деятельности:

Диаграмма деятельности варианта использования “Добавить клиента”

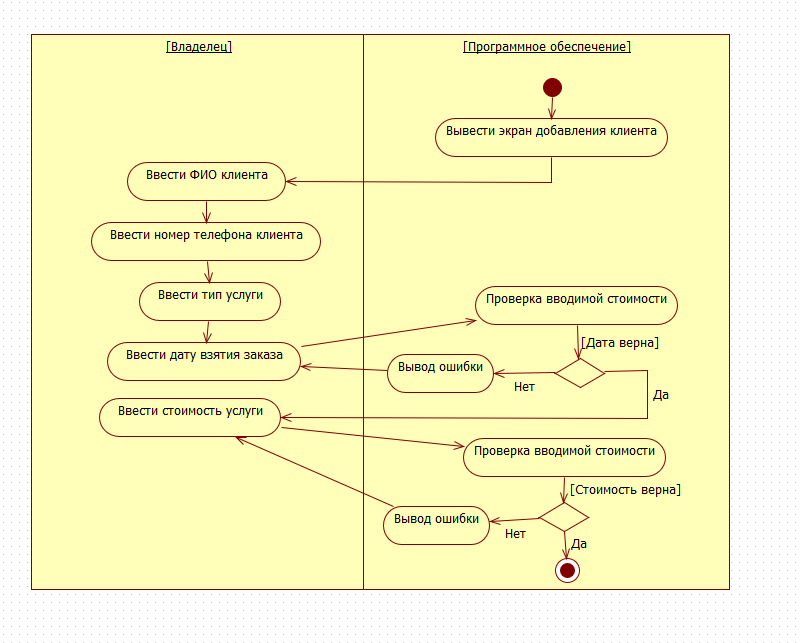
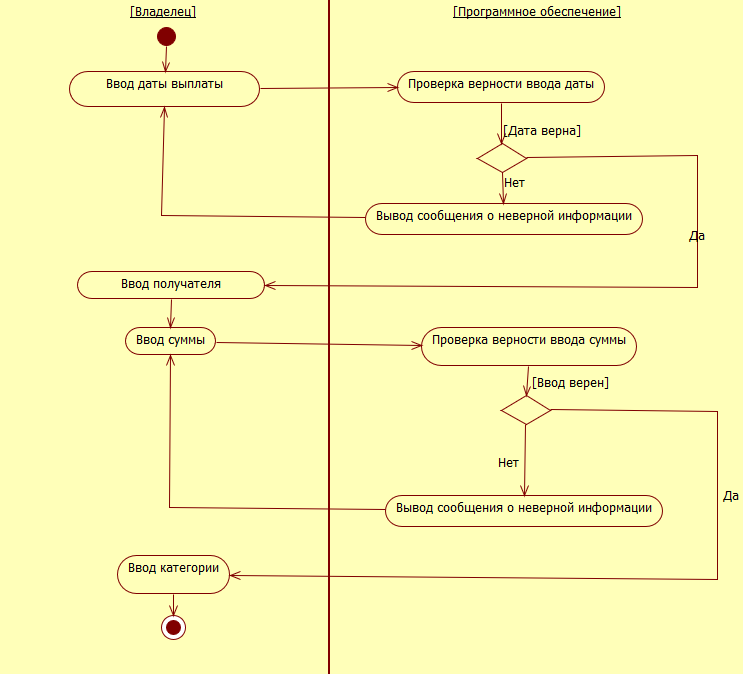
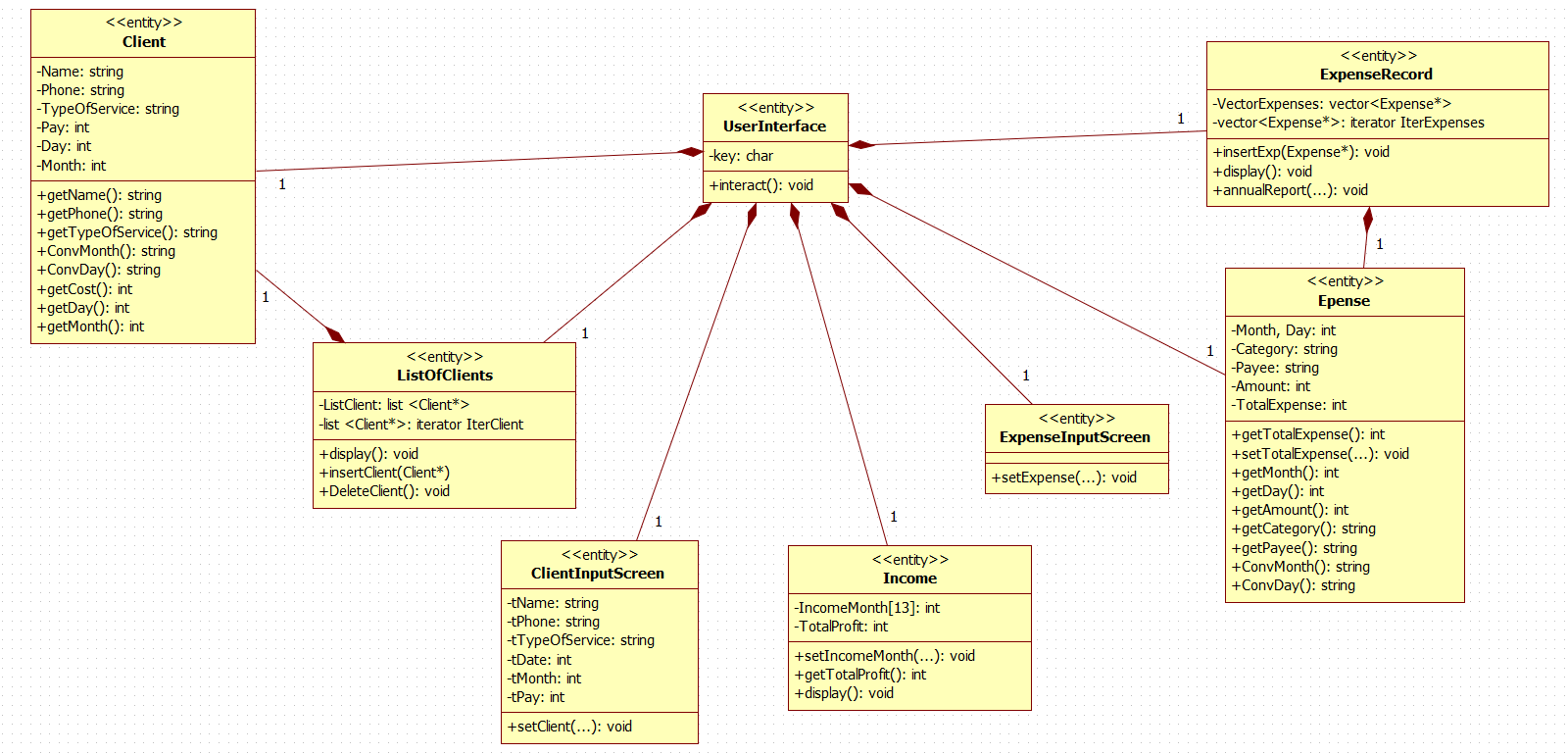


Диаграмма деятельности варианта использования “Добавить расходы”



1. **Список существительных**
2. Экран интерфейса пользователя
3. Клиент
4. Экран ввода клиента
5. Имя клиента
6. Номер телефона клиента
7. Услуга
8. Список клиентов
9. Месяц
10. Расходы
11. Стоимость услуги
12. Экран ввода расходов
13. День
14. Год
15. Таблица доходов
16. Таблица расходов
17. Доход
18. Годовой отчёт
19. Получатель
20. Категория
21. Чистая прибыль
22. Тип услуги
23. Сумма
24. Доходы
25. Прибыль
26. Список классов:
27. Экран пользовательского интерфейса
28. Клиент
29. Экран ввода клиента
30. Список клиентов
31. Экран ввода расходов
32. Доходы
33. Таблица расходов
34. Расходы
35. Атрибуты классов:
36. Экран пользовательского интерфейса
    1. Key – отвечает за нажатую клавишу
37. Клиент
    1. Name – ФИО клиента
    2. Phone – номер телефона клиента
    3. TypeOfService - тип услуги
    4. Pay - Стоимость услуги
    5. Day, Month - дата взятия заказа
38. Экран ввода клиента
    1. tName – ФИО клиента
    2. tPhone – номер телефона клиента
    3. tTypeOfService - тип услуги
    4. tPay - стомность услуги
    5. tDay, Month - дата взятия заказа
39. Список клиентов
    1. ListClient – список, который хранит информацию об каждом клиенте
    2. IterClient – итератор списка ListClient
40. Экран ввода расходов
    1. -
41. Доходы
    1. IncomeMonth – прибыль по каждому месяцу
    2. TotalProfit – общая прибыль
42. Таблица расходов
    1. VectorExpenses – вектор, который хранит информацию об расходах
    2. IterExpenses – итератор вектора VectorExpenses
43. Расходы
    1. TotalExpense – общая сумма расходов
    2. Month, Day – число оплаты расхода
    3. Amount – сколько заплатили
    4. Category – категория расходов
    5. Payee – кому заплатили
44. Список сообщений(методов) классов:
45. Экран пользовательского интерфейса
    1. метод interact - запускает главное меню программы
46. Клиент
    1. метод getName - возвращает ФИО клиента
    2. метод getPhone - возвращает номер телефона клиента
    3. метод getTypeOfService - возвращает тип услуги
    4. метод getCost - возвращает стоимость услуги
    5. метод getDay - возвращает дату принятия заказа
    6. метод getMonth - возвращает месяц принятия заказа
    7. метод ConvDay - конвертирует дату типа int в string
    8. метод ConvMonth - конвертирует месяц типа int в string
47. Экран ввода клиента
    1. метод setClien - добавляет клиента
48. Список клиентов
    1. метод insertClient - добавляет клиента в список
    2. метод display - выводит таблицу клиентов
    3. метод DeleteClient – удаление клиента
49. Экран ввода расходов
    1. метод setExpense - добавляет расходы
50. Доходы
    1. метод display - выводит таблицу доходов
    2. метод getTotalProfit - возвращает общую прибыль
    3. метод setIncomeMonth – вводит прибыль по каждому месяцу
51. Таблица расходов
    1. метод display - выводит таблицу расходов
    2. метод annualReport – вывод таблицу с годовым отчётом
    3. метод insertExp - добавляет расход в список
52. Расходы
    1. Метод getTotalExpense – возвращает общую сумму расходов
    2. Метод setTotalExpense – вводит общую сумму расходов
    3. метод ConvDay - конвертирует дату типа int в string
    4. метод ConvMonth - конвертирует месяц типа int в string
53. Диаграмма классов



1. Диаграммы последовательности

Диаграмма последовательности варианта использования «Добавить расходы»

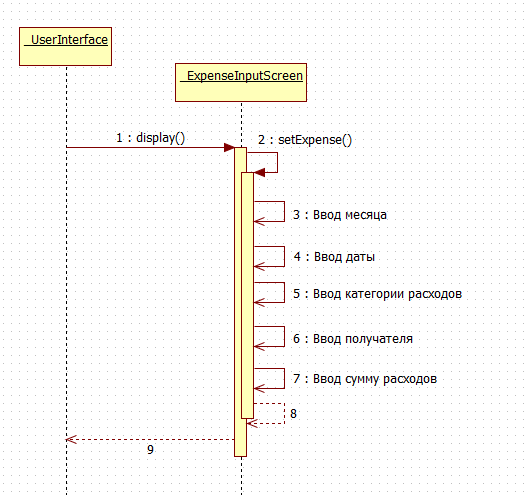


Диаграмма последовательности варианта использования «Удалить клиента»

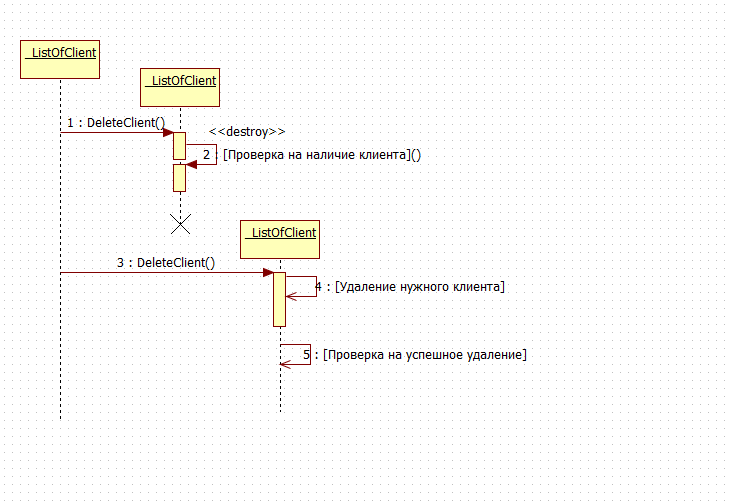
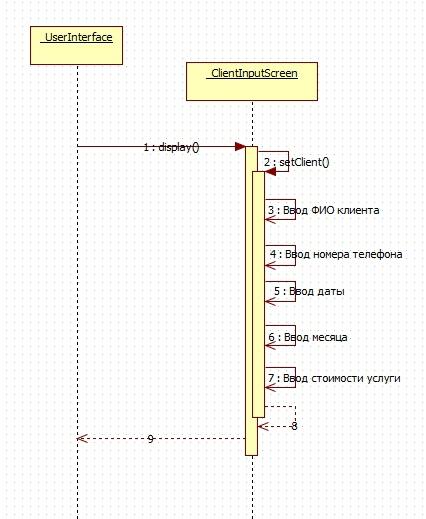


Диаграмма последовательности варианта использования «Добавить клиента»



1. Код заголовочных файлов

**Client.h**

#ifndef Client\_H

#define Client\_H

#include <string>

#include <iostream>

using namespace std;

//////////////////Класс Client//////////////////////

//этот класс содержит свойста(поля) клиента, а также методы для работы с ними

class Client

{

private:

string Name; // имя клиента

string Phone; // номер телефона клиента

string TypeOfService;

int Date;

int Month;

int Pay;

public:

Client();

Client(string name, string phone, string typeofservice, int date, int month, int pay);

~Client();

string getName(); //возвращает ФИО клиента

string getPhone(); //возвращает номер телефона клиента

string getTypeOfService();

string ConvDate();

string ConvMonth();

int getDate();

int getMonth();

int getPay();

}; //конец класса Client

#endif

**ClientInputScreen.h**

#ifndef ClientInputScreen\_H

#define ClientInputScreen\_H

using namespace std;

#include <string>

#include <iostream>

#include "ListOfClients.h"

#include "Client.h"

#include "Income.h"

////////////////класс ClientInputScreen////////////////////

//класс ClientInputScreen. Это класс, отвечающий за отображение «экрана»,

class ClientInputScreen

{

private:

string tName;

string tPhone;

int tDate;

int tMonth;

int tPay;

string tTypeOfService;

public:

void setClient(ListOfClients& \_ListOfClients, Income& \_Income); // добавить данные о клиенте

};//конец класса ClientInputScreen

#endif

**Expense.h**

#ifndef Expense\_H

#define Expense\_H

#include <iostream>

#include <string>

using namespace std;

//////////////////Класс Expense//////////////////////

//этот класс содержит свойста(поля) расходов, а также методы для работы с ними

class Expense

{

private:

int Month = 0, Day = 0; // месяц и день уплаты расходов

string Category; // категория расходов (газ, свет, ремонт и тд)

string Payee; // кому платим (поставщики газа, света, сервисный центр...)

int Amount = 0; // сколько платим

int TotalExpense = 0;

public:

Expense()

{ }

Expense(int m, int d, string c, string p, int a) :

Month(m), Day(d), Category(c), Payee(p), Amount(a)

{

}

void setTotalExpense(int expens);

int getTotalExpense();

int getMonth();

int getDay();

int getAmount();

string getCategory();

string getPayee();

string ConvDay();

string ConvMonth();

};//конец класса Expense

#endif

**ExpenseInputScreen.h**

#ifndef ExpenseInputScreen\_H

#define ExpenseInputScreen\_H

#include <iostream>

#include "ExpenseRecord.h"

//////////////////Класс ExpenseInputScreen//////////////////////

//Класс для ввода расходов

class ExpenseInputScreen

{

private:

public:

void setExpense(ExpenseRecord &ExpenseRecord, Expense &\_Expense);

}; //конец класса ExpenseInputScree

#endif

**ExpenseRecord.h**

#ifndef ExpenseRecord\_H

#define ExpenseRecord\_H

#include <iostream>

#include <vector>

#include <iomanip>

#include "Expense.h"

#include "Income.h"

using namespace std;

//////////////////Класс ExpenseRecord//////////////////////

//Класс записей о затратах

class ExpenseRecord

{

private:

vector<Expense\*> VectorExpenses; //вектор указателей на расходы

vector<Expense\*>::iterator IterExpenses;

public:

~ExpenseRecord();

void insertExp(Expense\*);

void display();

void annualReport(Income& \_Income, Expense &\_Expense);

};// конец класса ExpenseRecord

#endif

**Income.h**

#ifndef Income\_H

#define Income\_H

#include <iostream>

#include <iomanip>

using namespace std;

//////////////////Income//////////////////////

//Класс для работы с доходами

class Income

{

private:

int IncomeMonth[13] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };

int TotalProfit = 0;

public:

void setIncomeMonth(int i, int income);

int getTotalProfit();

void display();

};// конец класса Income

#endif

**ListOfClients.h**

#ifndef ListOfClients\_H

#define ListOfClients\_H

#include <list>

#include "Client.h"

#include <iomanip>

//////////////////Класс ListOfClients//////////////////////

//Класс записей о клиентах

class ListOfClients

{

private:

// установить указатели на клиентов

list <Client\*> ListClient; // указатели на класс клиент

list <Client\*>::iterator IterClient; //итератор

public:

~ListOfClients(); // деструктор (удаление клиентов)

void insertClient(Client\*); // добавить клиента в список

void display(); // вывод списка клиентов

void DeleteClient();

}; // конец класса ListOfClients

#endif

**UserInterface.h**

#include "Client.h"

#include "ListOfClients.h"

#include "ClientInputScreen.h"

#include "Expense.h"

#include "ExpenseRecord.h"

#include "ExpenseInputScreen.h"

#include "Income.h"

using namespace std;

//////////////////Класс UserInterface//////////////////////

//Главный класс для запуска приложения:

//этот класс определяет взаимодействие юзера с программой.

class UserInterface

{

private:

char key;

public:

void interact();

}; // конец класса userInterface

#endif

1. Код исполняемых файлов

**Laundry.cpp**

#include "UserInterface.h"

#include <windows.h>

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

UserInterface theUserInterface;

theUserInterface.interact();

return 0;

}

**Client.cpp**

#include "Client.h"

string Client::getName() //геттер возвращает ФИО клиента

{

return Name;

}

string Client::getPhone() //геттер возвращает ФИО клиента

{

return Phone;

}

string Client::ConvDate()

{

string sDate;

if (Date < 10)

{

sDate = "0" + std::to\_string(Date);

return sDate;

}

else return std::to\_string(Date);

}

int Client::getDate()

{

return Date;

}

string Client::ConvMonth()

{

string sMonth;

if (Month < 10)

{

sMonth = "0" + std::to\_string(Month);

return sMonth;

}

else return std::to\_string(Month);

}

int Client::getMonth()

{

return Month;

}

int Client::getPay()

{

return Pay;

}

string Client::getTypeOfService()

{

return TypeOfService;

}

Client::Client(string name, string phone, string typeofservice, int date, int month, int pay) : Name(name), Phone(phone), TypeOfService(typeofservice), Date(date), Month(month), Pay(pay)

{

}

Client::Client()

{

}

Client::~Client()

{

}

**ClientInputScreen.cpp**

#include "ClientInputScreen.h"

void ClientInputScreen::setClient(ListOfClients& \_ListOfClients, Income& \_Income) // добавить данные о клиенте

{

cout << "Введите ФИО клиента: " << endl;

tName = "";

while (tName == "")

{

getline(cin, tName);

}

tName = tName.substr(0, 30);//ограничение ввода

system("cls");

do

{

cout << "\n";

cout << "Введите номер телефона:" << endl;

cout << "+";

getline(cin, tPhone);

if (tPhone.size() < 11 || tPhone.size() > 11)// Слишком длинная ИЛИ слишком короткая

{

cout << "\nОшибка,повторите ввод\n";

system("pause");

}

system("cls");

} while (tPhone.size() < 11 || (tPhone.size() > 11));

system("cls");

cout << "\n";

cout << "Тип услуги:" << endl;

getline(cin, tTypeOfService);

tTypeOfService = tTypeOfService.substr(0, 15); //ограничение ввода

system("cls");

do {

system("cls");

cout << "\n";

cout << "Введите дату" << endl;

cin >> tDate;

if (tDate > 31 || tDate <= 0)

{

cout << "Неправильный формат даты" << endl;

cout << "\n";

system("pause");

}

else break;

} while (true);

do

{

system("cls");

cout << "\n";

cout << "Введите месяц:" << endl;

cin >> tMonth;

if (tMonth > 12 || tMonth <= 0)

{

cout << "Введён неправльный формат месяца" << endl;

cout << "\n";

system("pause");

}

else break;

} while (true);

do

{

system("cls");

cout << "\n";

cout << "Введите стоимость услуги:" << endl;

cin >> tPay;

if (tPay < 0)

{

cout << "Введён неправльный формат стоимости услуги" << endl;

cout << "\n";

system("pause");

}

else break;

} while (true);

system("cls");

\_Income.setIncomeMonth(tMonth, tPay);

Client\* ptrClient = new Client(tName, tPhone, tTypeOfService, tDate, tMonth, tPay); // создать клиента

\_ListOfClients.insertClient(ptrClient); // занести в список клиента

}

**Expense.cpp**

#include "Expense.h"

int Expense::getTotalExpense()

{

return TotalExpense;

}

int Expense::getAmount()

{

return Amount;

}

string Expense::getCategory()

{

return Category;

}

string Expense::ConvDay()

{

string sDay;

if (Day < 10)

{

sDay = "0" + std::to\_string(Day);

return sDay;

}

else return std::to\_string(Day);

}

int Expense::getDay()

{

return Day;

}

string Expense::ConvMonth()

{

string sMonth;

if (Month < 10)

{

sMonth = "0" + std::to\_string(Month);

return sMonth;

}

else return std::to\_string(Month);

}

int Expense::getMonth()

{

return Month;

}

string Expense::getPayee()

{

return Payee;

}

void Expense::setTotalExpense(int expens)

{

TotalExpense = TotalExpense + expens;

}

**ExpenseInputScreen.cpp**

#include "ExpenseInputScreen.h"

void ExpenseInputScreen::setExpense(ExpenseRecord& ExpenseRecord, Expense & \_Expense)

{

int month, day;

string category, payee;

float amount;

do {

system("cls");

cout << "\n";

cout << "Введите месяц:" << endl;

cin >> month;

if (month > 12 || month <= 0)

{

cout << "Неправильный формат месяца!" << endl;

cout << "\n";

system("pause");

}

else break;

} while (true);

system("cls");

cin.ignore(80, '\n');

do {

system("cls");

cout << "\n";

cout << "Введите день:" << endl;

cin >> day;

if (day > 31 || day <= 0)

{

cout << "Неправильный формат дня!" << endl;

cout << "\n";

system("pause");

}

else break;

} while (true);

system("cls");

cout << "\n";

cin.ignore(80, '\n');

cout << "Введите категорию расходов (Ремонт, Налоги): ";

getline(cin, category);

cout << "Введите получателя: ";

getline(cin, payee);

cout << "Введите сумму в рублях: ";

cin >> amount;

\_Expense.setTotalExpense(amount);

cin.ignore(80, '\n');

// создаем новый расход

Expense\* ptrExpense = new Expense(month, day, category, payee, amount);

// вставляем расход в список всех расходов

ExpenseRecord.insertExp(ptrExpense);

cout << "\n";

}

**ExpenseRecord.cpp**

#include "ExpenseRecord.h"

void ExpenseRecord::display() // распечатываем все расходы

{

cout << setw(8) << "Дата" << setw(5) << "||" << setw(23) << "Получатель" << setw(15) << "||" << setw(15) << "Сумма" << setw(10) << "||" << setw(15) << "Категория" << endl;

if (VectorExpenses.size() == 0) // В контейнере нет расходов

{

cout << "" << endl;

cout << setw(60) << "\*\*\*Расходов нет\*\*\*\n" << endl;

}

else

{

IterExpenses = VectorExpenses.begin();

while (IterExpenses != VectorExpenses.end())

{ // распечатываем все расходы

cout << setw(5) << (\*IterExpenses)->ConvMonth() << '/' << (\*IterExpenses)->ConvDay()<< setw(5) << "||" << setw(26) << (\*IterExpenses)->getPayee() << setw(12) << "||" << "";

cout << setw(15) <<(\*IterExpenses)->getAmount() << setw(10) << "||" << setw(15) << (\*IterExpenses)->getCategory() << endl;

IterExpenses++;

}

cout << endl;

}

}

void ExpenseRecord:: annualReport(Income& \_Income, Expense& \_Expense)

{

cout << "Годовой отчет\n--------------\n" << endl;

cout << "Доходы\n" << endl;

cout << "\tПрибыль:\t\t";

cout << \_Income.getTotalProfit() << endl;

cout << "Расходы\n" << endl;

IterExpenses=VectorExpenses.begin();

while (IterExpenses != VectorExpenses.end())

{

//выводим на экран категории расходов

cout << '\t' << ((\*IterExpenses)->getCategory()) << '\t' << ((\*IterExpenses)->getAmount()) << endl;

IterExpenses++;

}

cout << "\tРасходы всего:\t\t";

cout << \_Expense.getTotalExpense() << endl;

// вычисляем прибыльность

cout << "\nБаланс:\t\t" << \_Income.getTotalProfit()-\_Expense.getTotalExpense() << endl;

cout << "\n";

}

void ExpenseRecord::insertExp(Expense\* ptrExp)

{

VectorExpenses.push\_back(ptrExp);

}

ExpenseRecord::~ExpenseRecord() // деструктор

{ // удалить объекты expense

// удалить указатели на вектор

while (!VectorExpenses.empty())

{

IterExpenses = VectorExpenses.begin();

delete\* IterExpenses;

VectorExpenses.erase(IterExpenses);

}

}

**Income.cpp**

#include "Income.h"

void Income::setIncomeMonth(int i, int income)

{

IncomeMonth[i] = IncomeMonth[i] + income;

TotalProfit = TotalProfit + income;

}

int Income::getTotalProfit()

{

return TotalProfit;

}

void Income::display()

{

cout << setw(95) << "Месяцы" << endl;

cout << "" << endl;

cout << setw(8) << "Янв" << setw(8) << " || " << setw(8) << "Фев" << setw(8) << " || " << setw(8) << "Мар" << setw(8) << " || "<< setw (8) << "Апр" << setw(8) << " || " << setw(8) << "Май" << setw(8) << " || " << setw(8) << "Июн" << setw(8) << " || " << setw(8) << "Июл" << setw(8) << " || " << setw(8) << "Авг" << setw(8) << " || " << setw (8) << "Сен" << setw(8) << " || " << setw (8) << "Окт" << setw(8) << "|| " << setw (8) << "Ноя" << setw(8) << " || " << setw (8) << "Дек" << setw(8) << " || " << setw (8) << endl;

if (TotalProfit == 0)

{

cout << "" << endl;

cout << setw(110) << "\*\*\*Нет платежей!\*\*\*\n" << endl;

}

else

{

for (int i = 1; i <= 12; i++)

{

if (IncomeMonth[i] == 0)

{

cout << setw(8) << " 0 " << setw(8) << " ";

}

else

cout << setw(10) << IncomeMonth[i] << setw(6) << " ";

}

}

cout << "\n";

cout << "\n";

cout << "Суммарная прибыль:" << TotalProfit << endl;

cout << "\n";

}

**ListOfClients.cpp**

#include "ListOfClients.h"

using namespace std;

void ListOfClients::DeleteClient()

{

string delFIO = "";

if (ListClient.empty())

{

cout << "------------- Нечего удалять, так как нет клиентов -------------" << endl;

cout << "\n";

}

else {

cout << "Введите ФИО клиента, которого хотите удалить:" << endl;

while (delFIO == "")

{

getline(cin, delFIO);

}

system("cls");

IterClient = ListClient.begin();

while (IterClient != ListClient.end()) // распечатываем всех жильцов

{

if ((\*IterClient)->getName() == delFIO)

{

cout << "\n";

delete\* IterClient;

ListClient.erase(IterClient);

cout << "Клиент успешно удалён!" << endl;

cout << "\n";

IterClient = ListClient.end();

break;

}

else

{

\*IterClient++;

}

if (IterClient == ListClient.end())

{

cout << "\n";

cout << "Клиента с таким ФИО не сущетсвует!" << endl;

cout << "\n";

break;

}

}

}

system("pause");

system("cls");

}

ListOfClients::~ListOfClients() // деструктор

{ // удалить объекты ListOfClients

//удалить указатели на вектор

while (!ListClient.empty())

{

IterClient = ListClient.begin();

delete\* IterClient; // Удаляет объект из памяти

ListClient.erase(IterClient); // Удаляет объект из листа

}

}

void ListOfClients::insertClient(Client\* ptrT)

{

ListClient.push\_back(ptrT); // вставка нового клиента в список

}

void ListOfClients::display() // вывод списка клиентов

{

cout << setw(20) << "ФИО клиента "<< setw(14) << " || " <<setw(20) << "Номер телефона" << setw(8) << " || " <<setw(14) << "Тип услуги " << setw(5) << " || " << setw(9) << "Дата" << setw(9) << " || " << "Стоимость услуги" << endl;

if (ListClient.empty()) // если список клиентов пуст

cout << "\*\*\*Клиентов нет\*\*\*\n" << endl; // выводим запись, что он пуст)

else

{

IterClient = ListClient.begin();

while (IterClient != ListClient.end()) // распечатываем всех жильцов

{

cout << setw(30) << (\*IterClient)->getName() << " || " << setw(13) << "+" <<(\*IterClient)->getPhone() << " || "<< setw(15) << (\*IterClient)->getTypeOfService() << " || " << setw(8) << (\*IterClient)->ConvDate() << "/" << (\*IterClient)->ConvMonth() << setw(7) << " || " << setw(7)<< (\*IterClient)->getPay() << endl;

\*IterClient++;

}

}

cout << "\n";

}

**UserInterface.cpp**

#include <iostream>

#include <conio.h>

#include "UserInterface.h"

void UserInterface::interact()

{

Client \_Client;

ListOfClients \_ListOfClients;

ClientInputScreen \_ClientInputScreen;

Expense \_Expense;

ExpenseRecord \_ExpenseRecord;

ExpenseInputScreen \_ExpenseInputScreen;

Income \_Income;

while (true)

{

cout << "\n";

cout << " Для ввода данных нажмите '1' \n" << " Для вывода данных нажмите '2' \n" << " Для удаления клиента нажмите '3' \n" << " Для выхода 'q' \n";

key = \_getch();

system("cls");

if (key == '1') // ввод данных

{

cout << "\n";

cout << " Для добавление клиента нажмите '1' \n" << " Для записи расходов нажмите '2' \n" << " Вернуться назад '3'\n";

key = \_getch();

system("cls");

switch (key)

{

case '1':

cout << "\n";

\_ClientInputScreen.setClient(\_ListOfClients, \_Income);

break;

case '2':

cout << "\n";

\_ExpenseInputScreen.setExpense(\_ExpenseRecord, \_Expense);

system("pause");

system("cls");

break;

case '3':

system("cls");

break;

default:

cout << "Неизвестная функция\n";

cout << "\n";

system("pause");

system("cls");

break;

} // конец секции switch

} // конец условия if

else if (key == '2') // вывод данных

{

cout << "\n";

cout << " Для вывода клиентов нажмите '1' \n" << " Для вывода дохода нажмите '2' \n" << " Для вывода расходов нажмите '3' \n" << " Для вывода годового отчета нажмите '4' \n" << " Вернуться назад '5' \n";

key = \_getch();

system("cls");

switch (key)

{

case '1':

system("cls");

cout << "\n";

\_ListOfClients.display();

system("pause");

system("cls");

break;

case '2':

cout << "\n";

\_Income.display();

system("pause");

system("cls");

break;

case '3':

cout << "\n";

\_ExpenseRecord.display();

system("pause");

system("cls");

break;

case '4':

cout << "\n";

\_ExpenseRecord.annualReport(\_Income, \_Expense);

system("pause");

system("cls");

break;

case '5':

cout << "\n";

system("cls");

break;

default:

cout << "Неизвестная функция вывода\n";

system("pause");

break;

} // конец switch

} // конец elseif

else if (key == '3')

{

\_ListOfClients.DeleteClient();

}

else if (key == 'q' || key == 'й')

return; // выход

else

{

cout << "--------------------------------------------------\n";

cout << "!!!!!!!!!!!!! Неизвестная функция !!!!!!!!!!!!!!!!\n";

cout << "--------------------------------------------------\n";

cout << "\n";

system("pause");

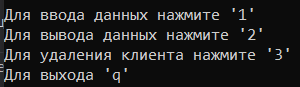
system("cls");

}

} // конец while

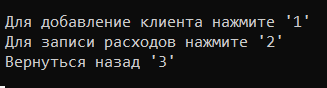
} // конец метода interact()

1. Руководство пользователя

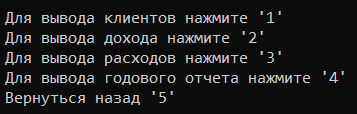


При запуске программы появляется главное меню, через которое можно

1. ввести данные (добавить клиента и ввести расходы)



1. вывести данные (вывод таблиц: клиентов, доходов, расходов, годового отчёта)



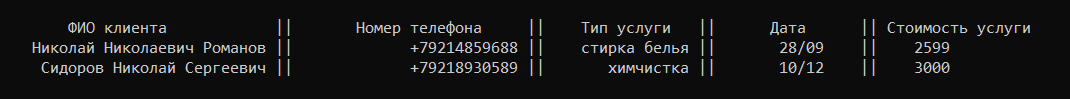
1. удалить клиента из базы
2. выйти из программы

Рассмотрим эти пункты более детально:

1. При добавлении клиента от пользователя ПО потребуется ввести ФИО клиента (или же инициалы), его номер телефона, тип услуги (например: стирка белья, продажа порошка для стирки), дату и месяц взятия заказа, а также его стоимость.

При неправильной набранной дате или месяца будет происходить повторный ввод.

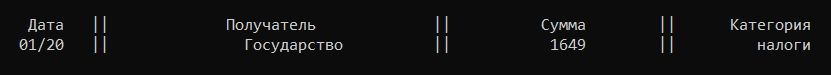
При выводе таблицы с клиентами можно увидеть примерно такую таблицу:



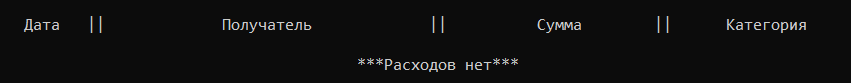
1. При добавлении расходов от пользователя ПО потребуется ввести дату и месяц покрытия данного расхода, саму категорию расходов (например: ремонт, налоги), кому платим эти расходу (иначе говоря получатель), а также сумму, которую заплатили.

При неправильной набранной дате или месяца будет происходить повторный ввод.

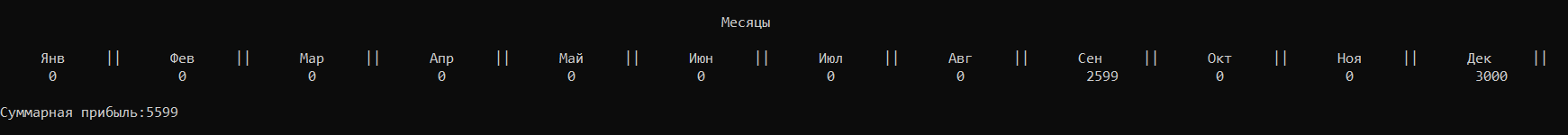
При выводе таблицы с расходами можно увидеть примерно такую таблицу:



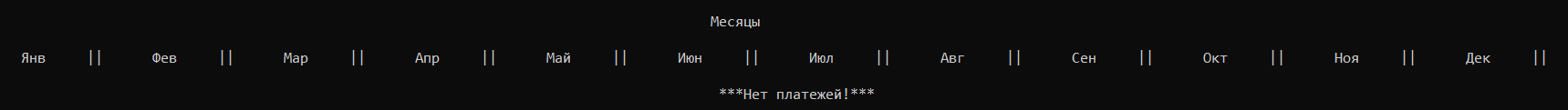
*Если же расходов нет, то таблица будет выглядеть так:*



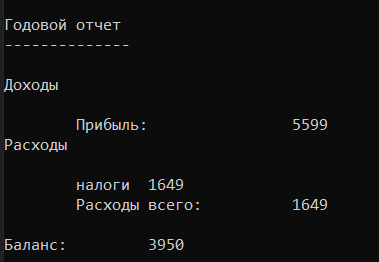
1. Таблица доходов будет выгляд*еть* примерно так:



*Если же клиентов нет, то таблица будет выглядеть так:*



1. Таблица годового отчёта будет выглядеть примерно так:



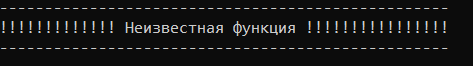
1. При удалении клиента пользователю ПО потребуется ввести только ФИО клиента (в точности такое же, какое оно было введено ранее)

Если же клиентов нет, то выведется такая подсказка:

**

Как можно было увидеть ранее, для переключения по меню используются такие клавиши, как «1», «2», «3», «4» и «q» (или же «й»).

При нажатии иной клавиши выскочит предупреждение:

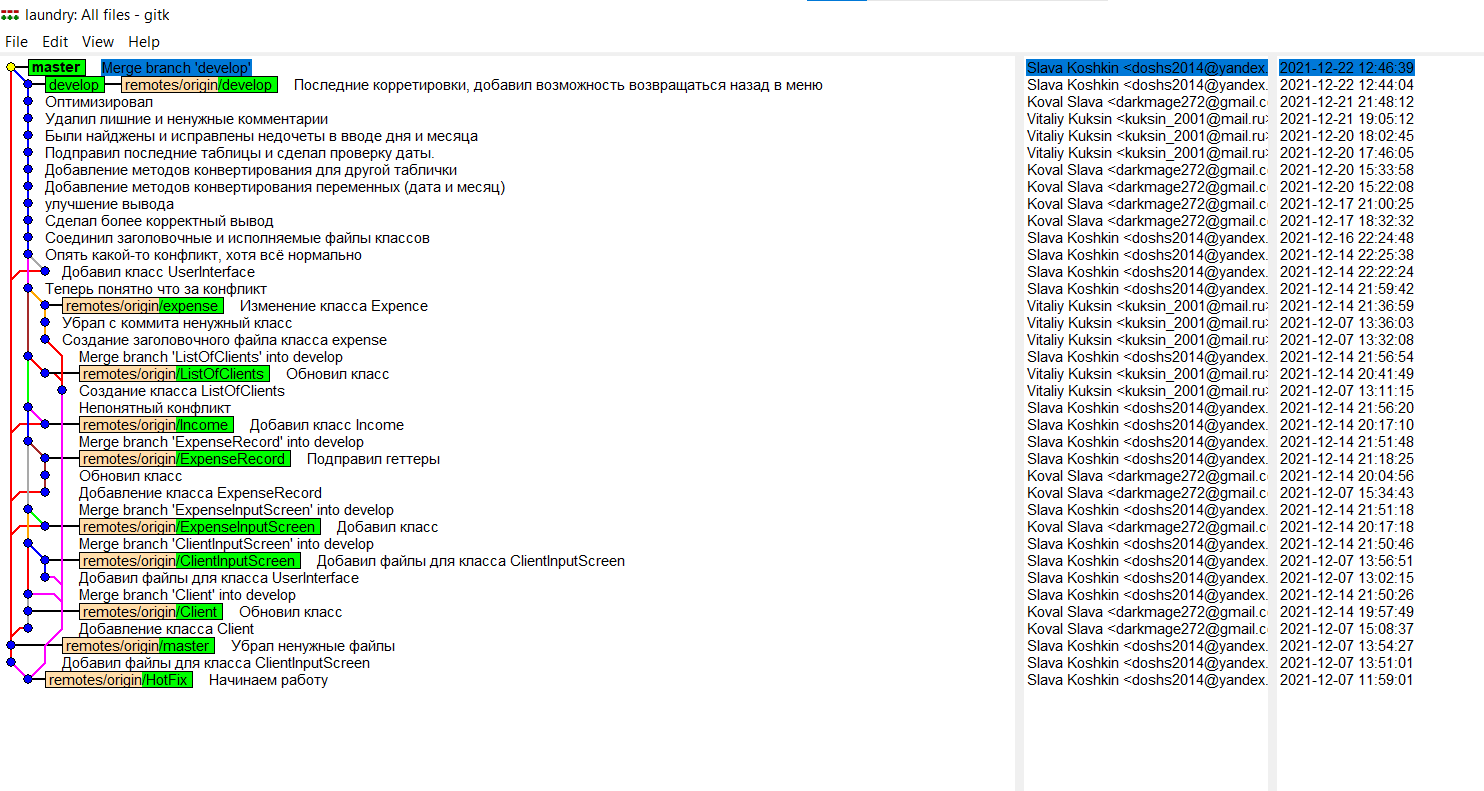


**Система контроля версий**

Удалённый репозиторий:

<https://github.com/StrayLucifer/laundry>

Графическая интерпретация истории репозитория через команду gitk:



**Пояснения к коммитам**

**Кошкин В.В.:**

“Merge branch “develop” ” - сделал слияние веток: master с develop

“Последние корректировки, добавил возможность возвращаться назад в меню” - добавил возможность возвращаться назад в меню (кнопка “Вернуться назад”).

“Соединил заголовочные и исполняемые файлы классов” - соединил все заговочный и исполняемые между собой (добавил #include, где пропустили их, а также добавили директивы #ifndef)

“Опять какой-то конфликт хотя всё нормально” - аналогично коммиту “Непонятный конфликт

“Добавил класс UserInterface” - изменил код в заголовочном и исполняемом файлах

“Теперь понятно что за конфликты” - аналогично коммиту “Непонятный конфликт”

“Merge branch “ListOfClients” into develop” - сделал слияние веток: develop c ListOfClients.

“Непонятный конфликт” - при слиянии веток (если не ошибаюсь develop и UserInterface), выводилось сообщение, что произошёл конфликт слияния, однако, тщательно проверив все файлы, предложение git-а исправить что-либо не было обнаружено. Предположительно это связано из-за того, что пару веток были сделаны на основе веток классов, а не на основе ветки master

“Добавил класс Income” - создал ветку Income и добавил в неё заголовочный и исполняемый файлы.

“Merge branch “ExpenseRecord” into develop” - сделал слияние веток: develop c ExpenseRecord

“Подправил геттеры” - добавил геттеры для класса ExpenseRecord (как-то вообще не обратил внимание, что поля класса были публичными, что есть очень нехорошо)

“Merge branch “ExpenseInputScreen” into develop” - сделал слияние веток: develop c ExpenseInputScreen

“Merge branch “ClientInputScreen” into develop” - сделал слияние веток: develop c ClientInputScreen

“Добавил файлы для класса ClientInputScreen” - создал ветку ClientInputScreen и добавил в неё заголовочный и исполняемый файлы.

“Добавил файлы для класса UserInterface” - добавил ветку UserInterface и добавил в неё заголовочный и исполняемый файлы.

“Merge branch “Client” into develop” - сделал слияние веток: develop c Client

“Убрал ненужные файлы” - случайно создал ветку Client, находясь на ветке ClientInputScreen, а не на ветке master, поэтому убрал с новой ветки файлы ветки ClientInputScreen.

“Добавил файлы для класса ClientInputScreen” - добавил ветку ClientInputScreen с заголовочным и исполняемым файлов одноименного класса.

“Начинаем работу” - создал проект, добавил .gitignore

**Коваль В. С.:**

“Добавление класса Client” - добавлены файлы Client.h и Client.cpp. В Client.h были описаны геттеры и сеттеры. В Client.cpp были объявление методов и переменных.

“Обновил класс” - добавление комментариев, а также исправление ошибок.

“Добавил класс” - добавление файлов ExpenseInputScreen.h и ExpenseInputScreen.срр. В ExpenseInputScreen.срр расположен метод ExpenseInputScreen::setExpense, который хранит в себе день, месяц, категорию расходов, получателя и сумму. В ExpenseInputScreen. h описан одноименный класс.

“Добавление класса ExpenseRecord” - Добавление файлов ExpenseRecord.срр и ExpenseRecord.h. В ExpenseRecord.срр расположены методы для реализации годового отчета. В ExpenseRecord.h находится описание одноименного класса.

“Обновил класс” - Исправление слетевшей кодировки русских символов.

“Сделал более корректный вывод” - создание конечной таблички в файлах ListOfClients.cpp, ClientInputScreen.cpp, путем добавления в ListOfClients.h директивы #include <iomanip>.

“улучшение вывода” - поправки к прошлому пункту в тех же файлах.

“Добавление методов конвертирования переменных (дата и месяц)” - добавление методов конвертирования в Client.срр, а также поправки в выводе файла ExpenseRecord.cpp и ListOfClients.cpp для более корректного вывода.

“Добавление методов конвертирования переменных для другой таблички” - добавление методов конвертирования в Expense.срр, а также поправки в выводе файла ExpenseRecord.cpp для более корректного вывода.

“Оптимизировал” - удалил ненужные строки кода, а также добавление или удаление комментариев.

**Куксин В.В.:**

“Удалил лишние и ненужные комментарии” - в программе оставались ненужные комментарии и закомментированные части кода, которые были не нужны.

“Были найдены и исправлены недочеты в вводе дня и месяца”- были небольшие недоработки в вводе дня и месяца. Они были обнаружены исправлены.

“Подправил последние таблицы и сделал проверку даты”-Сделал вывод таблиц более красивым, чтобы цифры и слова четко попадали в столбцы. Также сделал проверку ввода даты в классе ввода клиента и расходов.

“Изменение класса Expense” - доработал класс Expense, сделал вывод таблицы.

“Убрал с коммита ненужный класс”- случайно была создана ветка,, пришлось убрать с нее файлы.

“Создание заголовочного файла класса Expense”- был создан .h файл для класса Expense, а также заготовка для .cpp файла.

“Обновил класс”-доработал класс List of Clients. Прописал функционал.  
“Создание класса List of Clients”-создал .h файл для класса List of Clients. Начал работу с .cpp файлом.

**Заключение**

В качестве среды разработки использовалась Visual Studio 2019, язык программирования - C++. Мы использовали систему контроля версий Git, был создан репозиторий, на который каждый из участников команды загружал свои наработки и корректировки. Также в UML были сделаны имеющиеся диаграммы, с которыми каждый участник команды работал. Все из запланированных прецедентов были реализованы, сбои и зависания не наблюдаются. Использованы принципы раздельной компиляции, а также реализована очистка памяти. Неиспользуемые переменные отсутствуют, а также нет конструкций, без которых можно обойтись. Приложены следующие диаграммы: диаграмма прецедентов, диаграммы вариантов использования, диаграмма классов и диаграммы последовательности. Цель данной работы выполнена, так как смоделированный заказ был выполнен с использованием всех необходимых инструментов.