Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

«Петрозаводский государственный университет»

Кафедра информационно-измерительных систем и физической электроники

ЧАСТНАЯ КЛИНИКА

командное задание по дисциплине: «Технология программирования»

 Авторы работы:

 студенты группы 21312

\_\_\_\_\_\_\_\_\_\_\_\_ С. А. Семенов

\_\_\_\_\_\_\_\_\_\_\_\_ В. Р. Бояркин

\_\_\_\_\_\_\_\_\_\_\_\_ А. Д. Барановский

\_\_\_\_\_\_\_\_\_\_\_\_ С. И. Смирнов

«\_\_\_» \_\_\_\_\_\_\_\_\_ 2021 г.

 Научный руководитель:

 канд. физ.-мат. наук, доцент:

\_\_\_\_\_\_\_\_\_\_\_\_\_ А. В. Бульба

«\_\_\_» \_\_\_\_\_\_\_\_\_ 2021 г.

Петрозаводск 2021

СОДЕРЖАНИЕ

Введение . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .3

1 О программной реализации . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4   
2 Описание процесса разработки . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

2.1 Краткое словесное описание сюжета . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6

2.2 Описание имеющихся у заказчика материалов (данных). . . . . . . . . . . . . . . . 7

2.3 Действующие субъекты (actors) и их функциональные обязанности . . . . . . 8

2.4 Список вариантов использования . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .9

2.5 Отдельное описание вариантов использования . . . . . . . . . . . . . . . . . . . . . . . .10

2.6 Диаграммы действий . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .12

2.7 Список существительных .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .14

2.8 Уточненный список существительных . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .15

2.9 Список атрибутов классов с пояснением предназначения . . . . . . . . . . . . . . .16

2.10 Список сообщений с пояснениями . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .20

2.11 Диаграмма классов . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .21

2.12 Диаграмма последовательностей UML. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .22

2.13 Коды программы (.h) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .24

2.14 Коды программы (.cpp) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 30

2.15 Руководство пользователя . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 54   
3 История проекта на GitHub . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .59

Заключение . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .61

Введение

Целью данной работы является укрепление навыков и обобщение полученных ранее знаний путём разработки программного обеспечения на определённую тематику.

1 О программной реализации

Среда разработки: Microsoft Visual Studio 2019

Язык программирования: C++

Созданные единицы компиляции:

1. Client.cpp   
   (Описания методов класса Client возвращения данных)
2. Client.h   
   (Хранит данные клиента)
3. ClientInputScreen.cpp  
    (Ввод данных о клиенте)
4. ClientInputScreen.h   
   (Хранит прототип метода ввода данных о клиенте)
5. ClientList.cpp   
   (Список клиентов)
6. ClientList.h  
    (Хранит прототип метода вывода таблицы клиентов)
7. Clinic.cpp   
   (Реализация методов Clinic)
8. Clinic.h  
    (Хранит прототипы методов ввода и его проверки)
9. ClinicMain.cpp   
   (Хранит функцию main())
10. Expenses.h   
    (Хранит данные о расходах)
11. ExpensesInputScreen.cpp  
     (Ввод расходов)
12. ExpensesInputScreen.h   
    (Хранит прототип метода ввода расходов)
13. ExpenseRecord.cpp  
     (Реализация метода вывода расхода)
14. ExpenseRecord.h   
    (Хранит прототип метода вывода расходов)
15. Menu.cpp  
     (Реализация вывода меню)
16. Menu.h  
     (Хранит прототип метода вывода меню)
17. SeeDoctor.cpp  
    (Реализация методов возвращения данных о записи к врачу)
18. SeeDoctor.h  
    (Хранит данные о записи к врачу)
19. SeeDoctorInputScreen.cpp   
    (Реализация метода записи к врачу)
20. SeeDoctorInputScreen.h  
    (Хранит прототип метода записи к врачу)
21. SeeDoctorRecord.cpp  
    (Реализация методов ввод и вывода записи к врачу)
22. SeeDoctorRecord.h  
    (Хранит прототипы методов ввода и вывода записи к врачу)
23. Income.h  
    (Хранит данные о доходах)
24. IncomeRecord.cpp  
    (Реализация методы вывода доходов)
25. IncomeRecord.h  
    (Список доходов)
26. AnnualReportScreen.cpp  
    (Реализация методов вывода годового отчёта)
27. AnnualReportScreen.h  
    (Хранит прототипы методов вывода годового отчёта)
28. ProlongInsScreen.cpp  
    (Реализация методов продления страховки)
29. ProlongInsScreen.h

(Хранит прототипы методов продления страховки)

2 Описание процесса разработки

2.1 Краткое словесное описание сюжета

К команде разработчиков (авторы работы) обратился директор небольшой частной клиники с просьбой разработать программу, которая бы упростила учёт клиентов клиники и финансовую деятельность.Заказчик сообщил о том, что в данный момент он ведёт учёт в таблицах Excel. Изначально заказчик считал таблицы Excel самым удобным способом вести учёт, но в конце каждого года возникали трудности с подготовкой годового отчёта. Помимо этого, на заполнение таблиц уходило очень много времени, несмотря на всё удобство таблиц. Также использование приложения для работы с электронными таблицами (Excel) накладывало дополнительные расходы в виде покупки лицензии продукта.

Требования заказчика просты, в программном обеспечении необходимо реализовать быстрый и удобный учёт клиентов, расходов, доходов и запись клиентов к врачам. А также программа должна иметь похожие структуру и вид таблиц. В любой момент времени, должна иметься возможность вывести список клиентов, расходов, доходов и клиентов, записанных к врачам. Помимо всего прочего, в системе должна присутствовать функция вывода годового отчёта в один клик.

Программа не должна выдавать сбоев и ошибок. Интерфейс должен быть прост и интуитивно понятен.

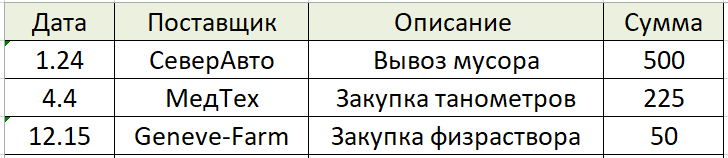
2.2 Описание имеющихся у заказчика материалов (данных)

Заказчик ведёт учёт клиентов и финансовой деятельности в приложении для работы с электронными таблицами (Excel).  
 Имеется несколько таблиц:

1. Таблица клиентов

Используется для учёта актуальной информации о клиентах. Содержит имя клиента, его номер страхования, тип страховки и период её действия.   

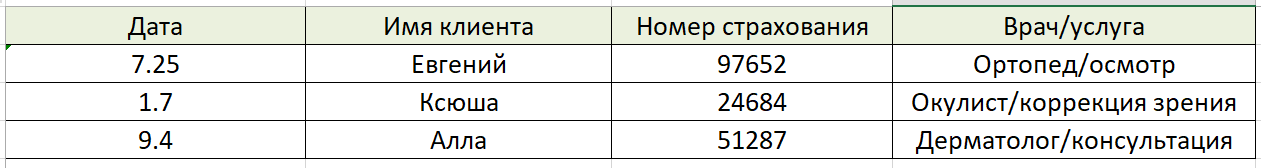

1. Таблица расходов

Ведение учёта расходов клиники. Содержит дату, поставщика услуг или материалов, описание и суммы расхода.  


1. Таблица доходов

Ведение учёта доходов клиники. Содержит дату, тип дохода, номер страхования и сумму дохода с клиента.  


1. Таблица записи клиентов к врачам  
   Используется для записи клиентов к врачам. Содержит дату записи, имя клиента, номер страхования, врача/услуги



1. Таблица годового отчёта

Используется для статистики. Содержит в себе доходы, итог по доходу, расходы, итого по расходу, описания к финансовой деятельности, результирующий годовой баланс и чистую прибыль.



2.3 Действующие субъекты (actors) и их функциональные обязанности

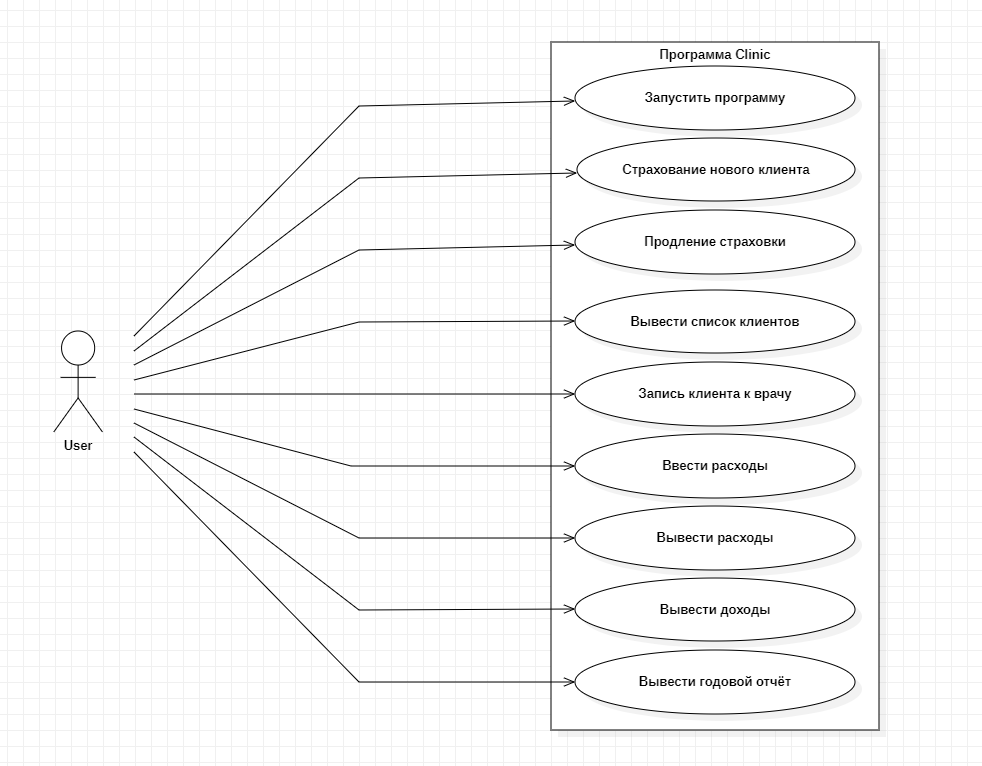
Единственным действующим лицом в системе является сам заказчик – директор клиники.

Ему доступны все возможности программы:

1. Добавление новых клиентов.
2. Продление страховки.
3. Запись клиентов к врачу.
4. Получение списка клиентов
5. Получение информации о расходах и доходах, оформление годового отчёта.

2.4 Список вариантов использования

1. “Запустить программу”
2. “Страхование нового клиента”
3. “Продление страховки”
4. “Вывести список клиентов”
5. “Запись клиента к врачу”
6. “Ввести расходы”
7. “Вывести расходы”
8. “Вывести доходы”
9. “Вывести годовой отчёт”



2.5 Отдельное описание вариантов использования

1. “Запустить программу”

На экран выводится меню (экран пользовательского интерфейса), из которого пользователь может выбрать нужное действие.

1. “Страхование нового клиента”

На экране отображается сообщение, в котором программа просит пользователя ввести имя нового клиента клиники, номер страхования, сегодняшнюю дату, а также указать тип страховки и период страхования. За каждый тип страховки установлена фиксированная цена. Вся информация заносится в таблицу. Расчетная стоимость страховки отправляется в доходы.

Тип страховки (standart, gold, gold+) предполагает различный доступ к спектру услуг.

1. “Продление страховки”

Экран продления страхования содержит сообщение, из которого пользователь узнает, что ему необходимо ввести номер страховки клиента, сегодняшнюю дату, период страховки, на который клиент хочет продлить страховку. Программа просматривает таблицу клиентов клиники, по номеру страхования находит соответствующую запись, и продлевает указанному клиенту период страхования на нужное количество месяцев.

Расчетная сумма страхования заносится в доходы.

1. “Вывести список клиентов”

Программа выводит на экран список клиентов клиники, каждая строка списка состоит из трёх полей: номер страхования, имя жильца и тип страховки.

1. “Запись клиента к врачу”

Программа запрашивает у пользователя информацию для записи:

-Номер страхования

-Дату приема

Просит выбрать врача и услугу.

Для клиентов с страховкой standart доступен только обычный прием, остальные страховки подразумевают дополнительные услуги.

Программа выведет ошибку если услуга не соответствует страховке клиента.

1. “Ввести расходы”

Экран ввода расхода содержат приглашение пользователю на ввод дня, месяца, поставщика, описания расходов, их суммы.

Затем программа создает новую строку, содержащую эту информацию, и вставляет ее в таблицу расходов.

1. “Вывести расходы”

Программа выводит на экран таблицу, каждая строка которой, состоит из значений месяца, дня, поставщика, описания и суммы.

1. “Вывести доходы”  
   Программа выводит на экран таблицу, каждая строка которой, состоит из типа дохода (страхования или продления страховки), даты, номера страхования и суммы дохода.
2. “Вывести годовой отчёт”

Программа выводит годовой отчет, состоящий из:

-списка всех доходов, суммы всех доходов за год;

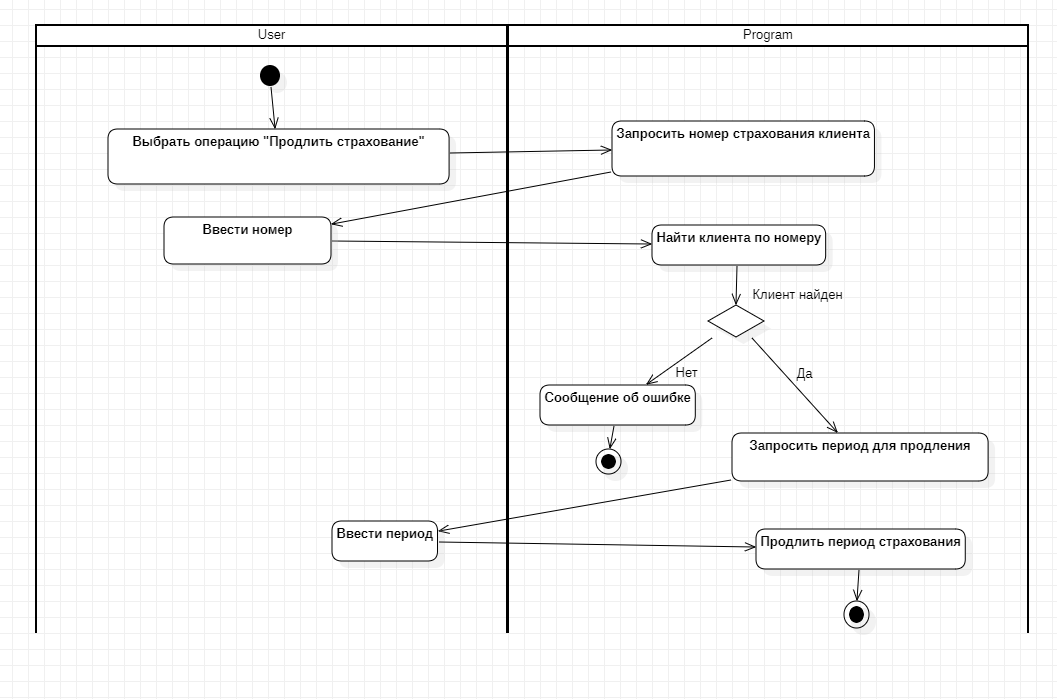
-списка всех расходов, суммы всех расходов за год;

-результирующего годового баланса (доходы/убытки);

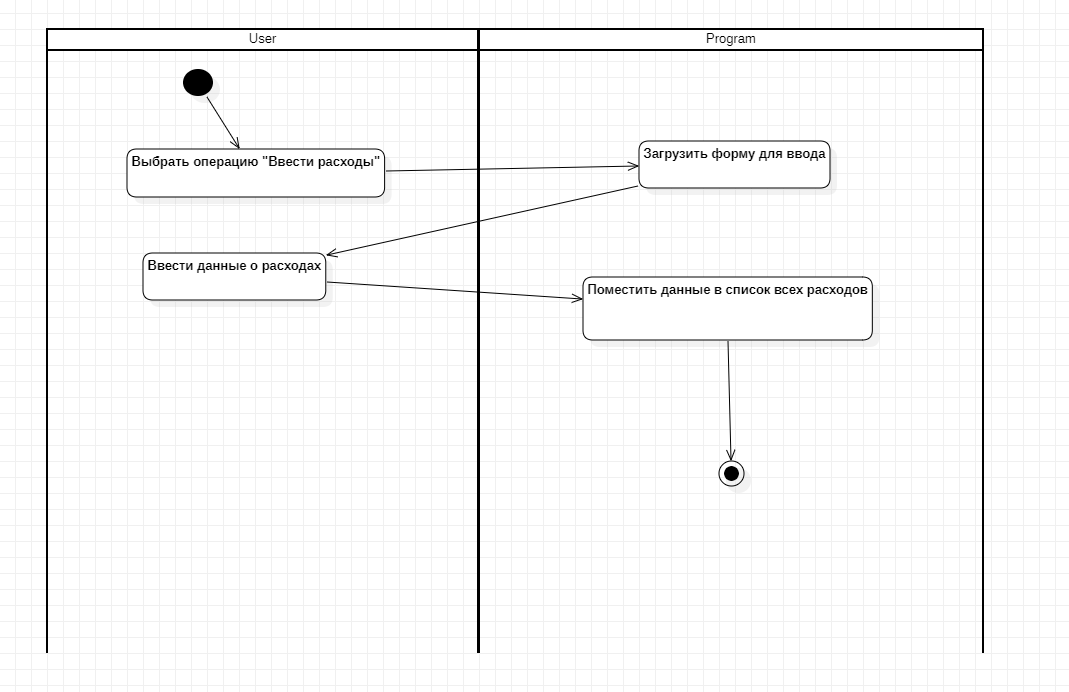
-чистой годовой прибыли (доходы-убытки);

2.6 Диаграммы действий

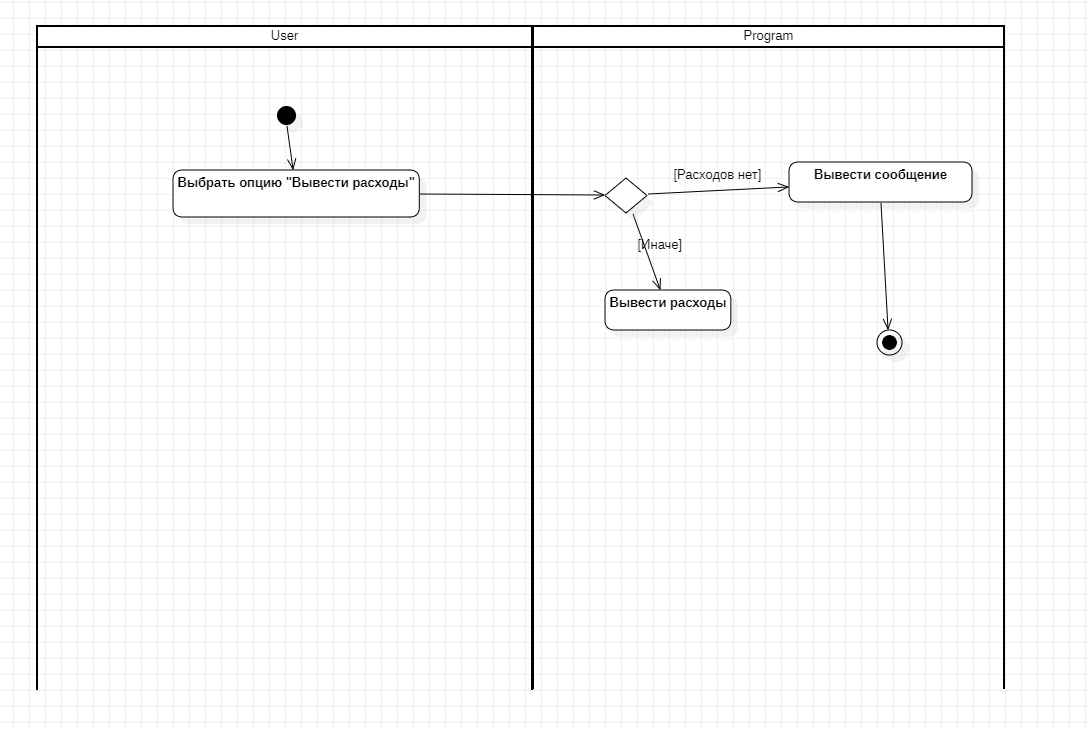
“Продление страховки”



“Ввести расходы”



“Вывести расходы”



2.7 Список существительных

1.Экран интерфейса пользователя.

2. Экран оформления страховки.

3. Экран продления страховки.

4. Клиент.

5. Имя клиента.

6. Строка клиента.

7. Список клиентов.

8. Номер страховки.

9. Вид страховки.

10. Плата за страховку.

11. Период страхования.

12. Врач\услуга.

13. Дата.

14. Доход

15. Таблица доходов.

16. Тип дохода.

17. Строка доходов.

18. Сумма дохода.

19. Суммарный доход.

20. Расход.

21. Поставщик.

22. Размер платежа.

23. Строка в таблице расходов.

24. Таблица расходов.

25. Описание расхода.

26. Сумма расхода

27. Суммарные расход.

28. Годовой отчет.

29. Результирующего годового баланса

30. Чистая прибыль

31. Клиника

32.Экран занесения расходов.

33.Экран записи к врачу.

34.Запись к врачу

35.Таблица записи к врачу

36.Продление страховки

2.8 Уточненный список существительных

1. Интерфейс пользователя

Главный класс для запуска приложения: этот класс определяет взаимодействие юзера с программой.

1. Экран оформления страховки

Экран для оформления новоиспечённого клиента. Добавление платы за страховку.

1. Экран продления страховки

Экран для оформления зарегистрированного клиента. Добавление платы за страховку.

1. Клиенты

Класс хранящий данные о клиентах.

1. Список клиентов

Хранит список клиентов.

1. Расходы

Хранит данные о расходах клиники.

1. Экран занесения расходов

Экран для занесения расходов клиник и описания к ним.

1. Таблица расходов

Хранит форму вывода расходов.

1. Запись к врачу

Хранит данные о записях к врачу.

1. Экран записи к врачу

Экран для записи к врачу, требует дату, номер страховки, врача и услугу.

1. Таблица записи к врачу

Хранит форму вывода записей к врачу.

1. Доход

Хранит данные о доходах клиники.

1. Таблица доходов

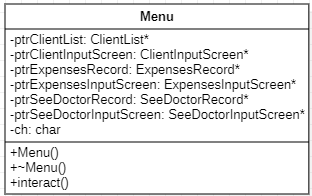
Хранит форму вывода доходов

1. Экран годового отчёта

Экран вывода всех данных для статистики

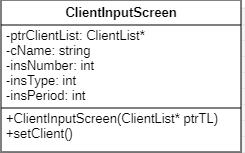
2.9 Список атрибутов классов с пояснением предназначения

1. Интерфейс пользователя



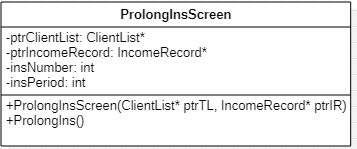
* Содержит указатели на классы с видимостью private.
* Атрибут ch c видимостью private, хранит в себе введённый пользователем символ.

1. Экран оформления страховки



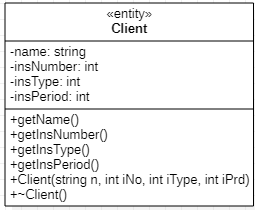
* ptrClientList - указатель на класс ClientList с видимостью private.
* Атрибут cName предназначен для хранения имени клиента
* Атрибут insNumber предназначен для хранения номера страховки клиента
* Атрибут insType предназначен для хранения типа страховки клиента
* Атрибут insPeriod предназначен для хранения оплаченного периода страховки клиента

1. Экран продления страховки



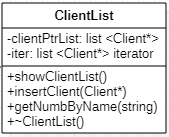
* ptrClientList - указатель на класс ClientList с видимостью private.
* ptrIncomeRecord -указатель на класс IncomeRecord с видимостью private.
* Атрибут insNumber предназначен для хранения номера страховки клиента
* Атрибут insPeriod предназначен для хранения оплаченного периода страховки клиента

1. Клиенты



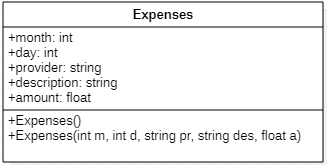
* Атрибут name предназначен для хранения имени клиента
* Атрибут insNumber предназначен для хранения номера страховки клиента
* Атрибут insType предназначен для хранения типа страховки клиента
* Атрибут insPeriod предназначен для хранения оплаченного периода страховки клиента

1. Список клиентов



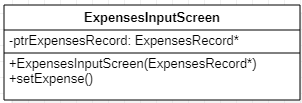
* clientPtrList - список указателей на класс Client с видимостью private
* iter – итератор списка

1. Расходы



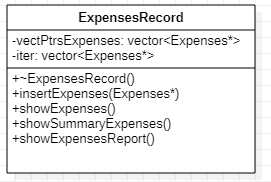
* Атрибуты month, day предназначены для хранения даты расхода
* Атрибут provider предназначен для хранения фирмы поставщика
* Атрибут description предназначен для хранения описания расхода
* Атрибут amount предназначен для хранения суммы расхода

1. Экран занесения расходов



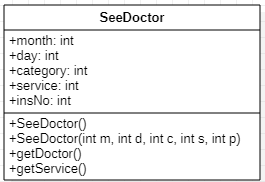
* ptrExpensesRecord – указатель на класс ExpensesRecord с видимостью private

1. Таблица расходов



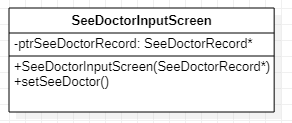
* vectPtrExpenses – вектор указателей на расходы с видимостью private
* iter – итератор вектора

1. Запись к врачу



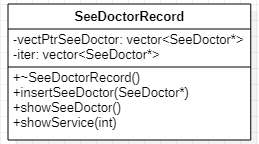
* Атрибуты month, day предназначены для хранения даты записи
* Атрибут category предназначен для хранения категории врача
* Атрибут service предназначен для хранения описания расхода
* Атрибут insNo предназначен для хранения номера страховки

1. Экран записи к врачу



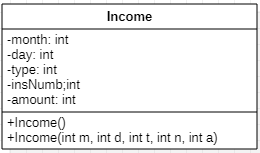
* ptrSeeDoctorRecord – указатель на класс SeeDoctorRecord с видимостью private

1. Таблица записи к врачу



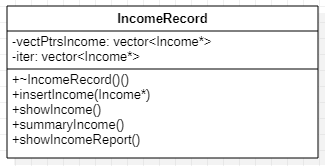
* vectPtrSeeDoctor – вектор указателей на запись к врачу с видимостью private
* iter – итератор вектора

1. Доход



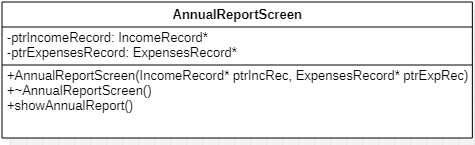
* Атрибуты month, day предназначены для хранения даты дохода
* Атрибут type предназначен для хранения типа дохода
* Атрибут insNumb предназначен для хранения номера страховки
* Атрибут amount предназначен для хранения суммы дохода

1. Таблица доходов



* vectPtrsIncome – вектор указателей на доходы с видимостью private
* iter – итератор вектора

1. Экран годового отчёта



* ptrIncomeRecord - указатель на класс IncomeRecord с видимостью private.
* ptrExpensesRecord -указатель на класс ExpensesRecord с видимостью private.

2.10 Список сообщений с пояснениями

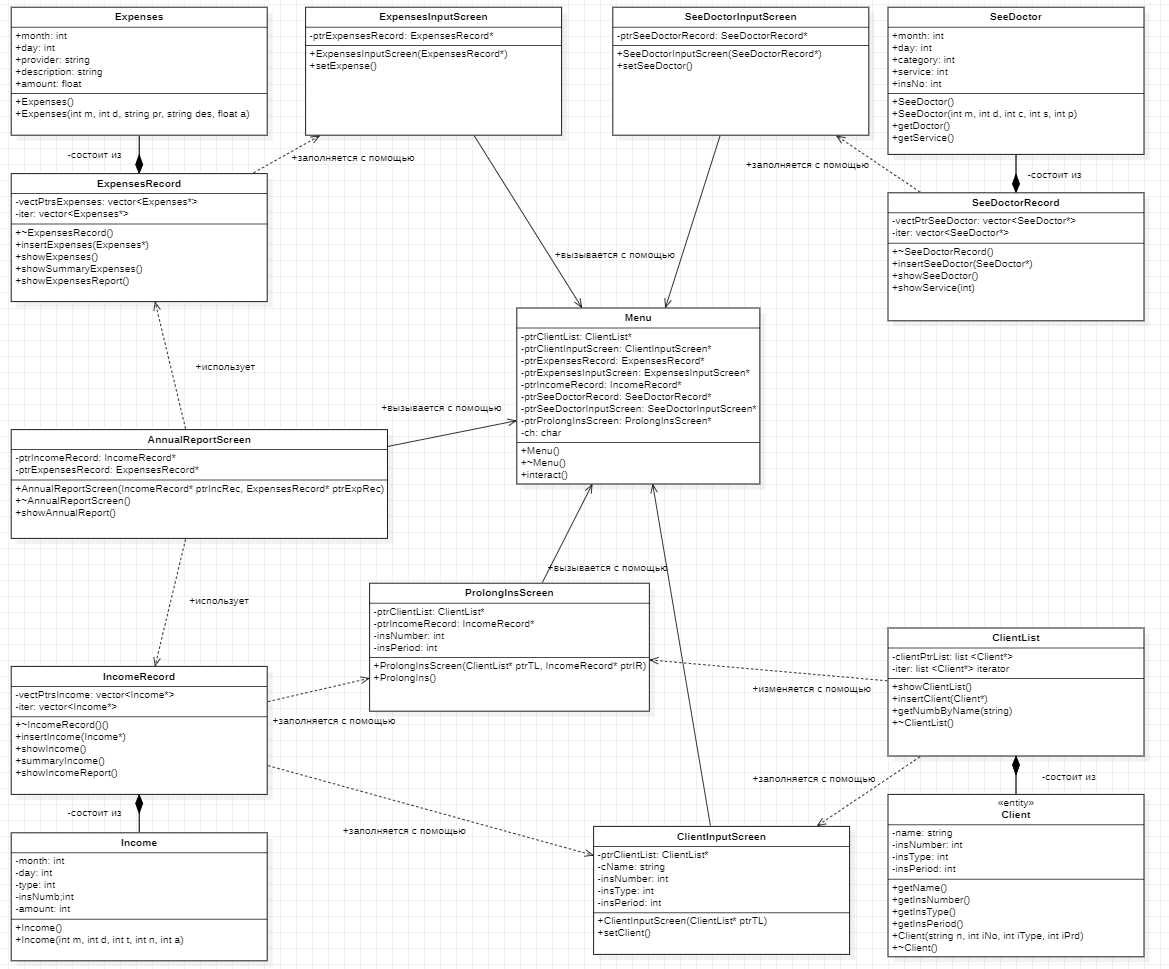
Класс AnnualReportScreen выводит годовой отчет, используя методы showExpensesReport и showIncomeReport, классов ExpensesRecord и IncomeRecord.

Классы ClientInputScreen, SeeDoctorInputScreen, ExpensesInputScreen, ProlongInsScreen заполняют списки клиентов, записей к врачам, расходов, используя методы insert…() классов ClientList, SeeDoctorRecord, ExpensesRecord, IncomeRecord соответственно.

Класс Menu вызывает формы ввода SeeDoctorInputScreen, ClientInputScreen, AnnualReportScreen, ExpensesInputScreen, ProlongInsScreen, используя конструкторы этих классов.

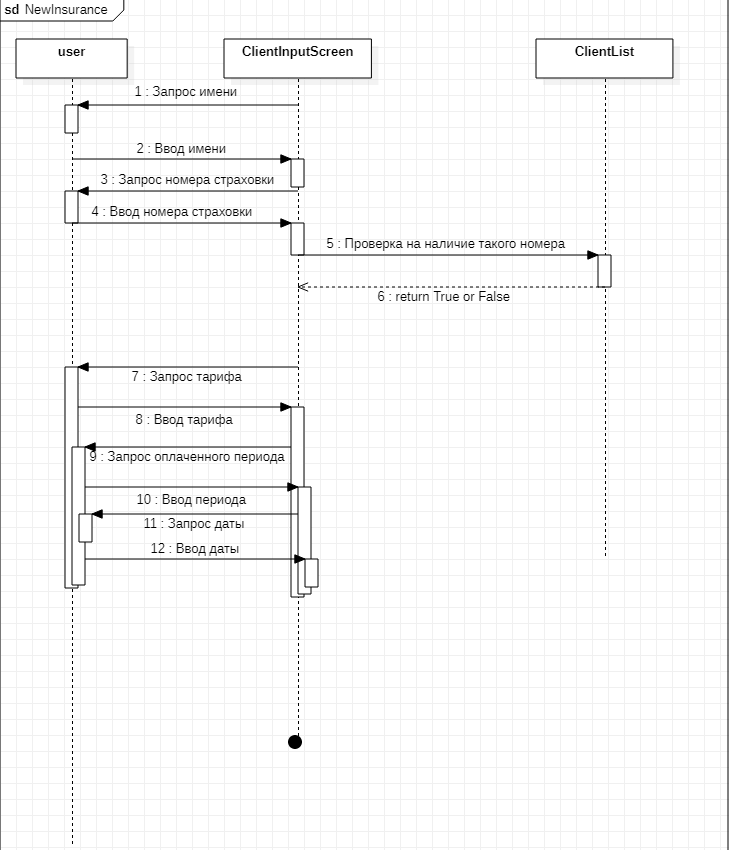
Класс Menu выводит таблицы клиентов, доходов, расходов, записей к врачу, используя методы show…() классов ClientList, IncomeRecord, ExpensesRecord, SeeDoctorRecord.

2.11 Диаграмма классов

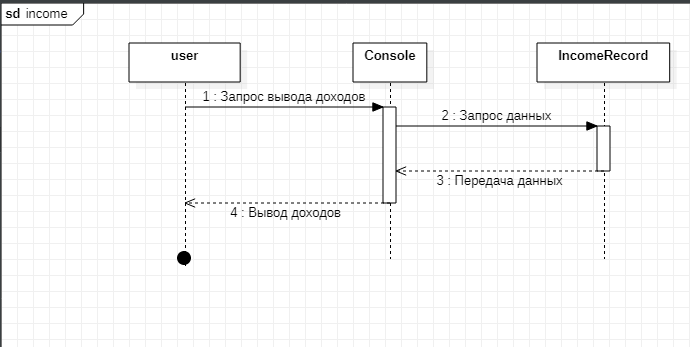


2.12 Диаграмма последовательностей UML

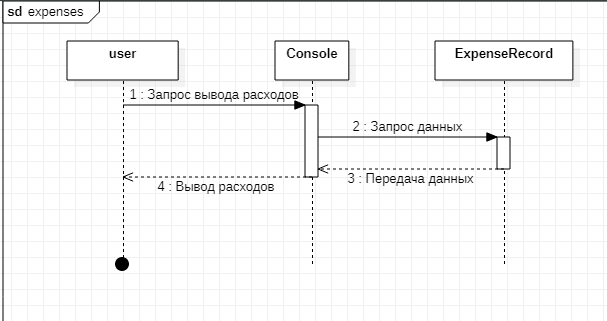
1. Страхование нового клиента



1. Вывести доходы



1. Вывести расходы



2.13 Коды программы (.h)

***AnnualReportScreen.h:***

#pragma once

#include "Clinic.h"

#include "Client.h"

#include "Income.h"

#include "IncomeRecord.h"

#include "Expenses.h"

#include "ExpensesRecord.h"

class AnnualReportScreen

{

private:

IncomeRecord\* ptrIncomeRecord;

ExpensesRecord\* ptrExpensesRecord;

public:

AnnualReportScreen(IncomeRecord\* ptrIncRec, ExpensesRecord\* ptrExpRec) : ptrIncomeRecord(ptrIncRec), ptrExpensesRecord(ptrExpRec)

{

}

~AnnualReportScreen();

void showAnnualReport();

};

***Client.h:***

#pragma once

#include "Clinic.h"

// Класс Client, хранит данные клиента

class Client

{

private:

string name; // имя клиента

int insNumber; // номер страховки

int insType; // тип страховки (1-standart/2-gold/3-goldPlus)

int insPeriod; // период страхования

public:

Client(string n, int iNo, int iType, int iPrd); // создание нового клиента через конструктор с параметрами

~Client(); // деструктор

string getName(); //возвращает имя клиента

int getInsNumber(); //возвращает номер страховки

int getInsType(); // возвращает тип страховки

int getInsPeriod(); // возвращает период страхования

void changeInsPeriod(int newPeriod); // редактирует поле периода страхования

};

***ClientInputScreen.h:***

#pragma once

#include "Clinic.h"

#include "Client.h"

#include "ClientList.h"

#include "Income.h"

#include "IncomeRecord.h"

class ClientInputScreen

{

private:

ClientList\* ptrClientList;

IncomeRecord\* ptrIncomeRecord;

string cName;

int insNumber;

int insType;

int insPeriod;

public:

ClientInputScreen(ClientList\* ptrTL, IncomeRecord\* ptrIR) : ptrClientList(ptrTL), ptrIncomeRecord(ptrIR)

{

}

void setClient();

};

***ClientList.h:***

#pragma once

#include "Clinic.h"

#include "Client.h"

class ClientList

{

private:

list <Client\*> clientPtrList;

list <Client\*>::iterator iter;

public:

~ClientList();

void insertClient(Client\*);

int getNumbByName(string);

int getTypeByNumb(int insNo);

int checkNumb(int insNo);

void changeTypeByNumb(int insNo, int newType);

void prolongPeriodByNumb(int insNo, int newPeriod);

void showClientList();

};

***Clinic.h:***

#pragma once

#include <iostream>

#include <vector>

#include <list>

#include <string>

#include <numeric> //для accumulate()

using namespace std;

static int standartPay = 500;

static int goldPay = 1000;

static int goldPlusPay = 2000;

string getaLine(int n); // получение строки текста

char getaChar(); // получение символа

int getaNumb(int min, int max);

***Expenses.h:***

#pragma once

#include "Clinic.h"

#include "Client.h"

class Expenses

{

public:

int month, day;

string provider;

string description;

float amount;

Expenses()

{

}

Expenses(int m, int d, string pr, string des, float a) : month(m), day(d), provider(pr), description(des), amount(a)

{

}

};

***ExpensesInputScreen.h:***

#pragma once

#include "Clinic.h"

#include "Client.h"

#include "Expenses.h"

#include "ExpensesRecord.h"

class ExpensesInputScreen

{

private:

ExpensesRecord\* ptrExpensesRecord;

public:

ExpensesInputScreen(ExpensesRecord\*);

void setExpense();

};

***ExpensesRecord.h:***

#pragma once

#include "Clinic.h"

#include "Client.h"

#include "Expenses.h"

class ExpensesRecord

{

private:

vector<Expenses\*> vectPtrsExpenses;

vector<Expenses\*>::iterator iter;

public:

~ExpensesRecord();

void insertExpenses(Expenses\*);

void showExpenses();

int summaryExpenses();

void showExpensesReport();

};

***Income.h:***

#pragma once

#include "Clinic.h"

#include "Client.h"

class Income

{

public:

int month, day, type, insNumb, amount;

Income()

{

}

Income(int m, int d, int t, int n, int a) : month(m), day(d), type(t), insNumb(n), amount(a)

{

}

};

***IncomeRecord.h:***

#pragma once

#include "Clinic.h"

#include "Client.h"

#include "Income.h"

class IncomeRecord

{

private:

vector<Income\*> vectPtrsIncome;

vector<Income\*>::iterator iter;

public:

~IncomeRecord();

void insertIncome(Income\*);

void showIncome();

int summaryIncome();

void showIncomeReport();

};

***Menu.h:***

#pragma once

#include "Clinic.h"

#include "Client.h"

#include "ClientInputScreen.h"

#include "ExpensesRecord.h"

#include "IncomeRecord.h"

#include "ExpensesInputScreen.h"

#include "SeeDoctorRecord.h"

#include "SeeDoctorInputScreen.h"

#include "prolongInsScreen.h"

#include "AnnualReportScreen.h"

class Menu

{

private:

ClientList\* ptrClientList;

ClientInputScreen\* ptrClientInputScreen;

ExpensesRecord\* ptrExpensesRecord;

IncomeRecord\* ptrIncomeRecord;

ExpensesInputScreen\* ptrExpensesInputScreen;

SeeDoctorRecord\* ptrSeeDoctorRecord;

SeeDoctorInputScreen\* ptrSeeDoctorInputScreen;

ProlongInsScreen\* ptrProlongInsScreen;

AnnualReportScreen\* ptrAnnualReportScreen;

char ch;

public:

Menu();

~Menu();

void interact();

***ProlongInsScreen.h:***

#pragma once

#include "Clinic.h"

#include "Client.h"

#include "ClientList.h"

#include "Income.h"

#include "IncomeRecord.h"

class ProlongInsScreen

{

private:

ClientList\* ptrClientList;

IncomeRecord\* ptrIncomeRecord;

int insNumber;

int insPeriod;

public:

ProlongInsScreen(ClientList\* ptrTL, IncomeRecord\* ptrIR) : ptrClientList(ptrTL), ptrIncomeRecord(ptrIR)

{

}

void ProlongIns();

};

***seeDoctor.h:***

#pragma once

#include "Clinic.h"

#include "Client.h"

class SeeDoctor

{

public:

int month, day;

int category, service;

int insNo;

SeeDoctor()

{

}

SeeDoctor(int m, int d, int c, int s, int p) : month(m), day(d), category(c),service(s), insNo(p)

{

}

int getDoctor(); // возвращает тип страховки

int getService(); // возвращает тип страховки

};

***SeeDoctorInputScreen.h:***

#pragma once

#include "Clinic.h"

#include "Client.h"

#include "SeeDoctor.h"

#include "SeeDoctorRecord.h"

class SeeDoctorInputScreen

{

private:

SeeDoctorRecord\* ptrSeeDoctorRecord;

ClientList\* ptrClientList;

public:

SeeDoctorInputScreen(SeeDoctorRecord\*, ClientList\*);

void setSeeDoctor();

};

***SeeDoctorRecord.h:***

#pragma once

#include "Clinic.h"

#include "Client.h"

#include "SeeDoctor.h"

class SeeDoctorRecord

{

private:

vector<SeeDoctor\*> vectPtrSeeDoctor;

vector<SeeDoctor\*>::iterator iter;

public:

~SeeDoctorRecord();

void insertSeeDoctor(SeeDoctor\*);

void showSeeDoctor();

void showService(int);

};

2.14 Коды программы (.cpp)

***AnnualReportScreen.cpp:***

#include "Clinic.h"

#include "Client.h"

#include "AnnualReportScreen.h"

void AnnualReportScreen::showAnnualReport()

{

system("cls");

cout << "|| Дата || Тип операции || Источник || Сумма ||" << endl;

cout << "||\_\_\_\_\_\_\_||\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_||\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_||\_\_\_\_\_\_\_\_\_\_\_||" << endl;

ptrIncomeRecord->showIncomeReport();

ptrExpensesRecord->showExpensesReport();

float sumInc = ptrIncomeRecord->summaryIncome();

float sumExp = ptrExpensesRecord->summaryExpenses();

if (sumExp == 0)

{

cout << "\nРезультирующий годовой баланс: -";

}

else

{

float resultCount = sumInc / sumExp;

cout << "\nРезультирующий годовой баланс: " << resultCount;

}

int cleanCount = sumInc - sumExp;

cout << "\nЧистая годовая прибыль: " << cleanCount;

cout << endl << endl;

system("pause");

}

***Client.cpp:***

#include <iostream>

#include "Clinic.h"

#include "Client.h"

using namespace std;

// Методы класса Client

Client::Client(string n, int iNo, int iType, int iPrd) : name(n), insNumber(iNo), insType(iType), insPeriod(iPrd)

{

}

Client::~Client()

{

}

string Client::getName()

{

return name;

}

int Client::getInsNumber()

{

return insNumber;

}

int Client::getInsType()

{

return insType;

}

int Client::getInsPeriod()

{

return insPeriod;

}

void Client::changeInsPeriod(int newPeriod)

{

insPeriod = newPeriod;

}

***ClientInputScreen.cpp:***

#include "Clinic.h"

#include "Client.h"

#include "ClientList.h"

#include "ClientInputScreen.h"

#include "IncomeRecord.h"

void ClientInputScreen::setClient()

{

int month, day, type, amount;

system("cls");

cout << "Оформление страховки: ";

cout << "\n\nИмя клиента: ";

cName = getaLine(20);

cout << "\nНомер страхования (ХХХХХХ): ";

insNumber = getaNumb(111111, 999999);

while (true)

{

if (ptrClientList->checkNumb(insNumber) == 1)

{

cout << "\nКлиент с таким номером уже зарегистрирован, повторите ввод: ";

insNumber = getaNumb(111111, 999999);

}

else

{

break;

}

}

cout << "\nДоступные тарифы страхования:";

cout << "\n1 - Standart";

cout << "\n2 - Gold ";

cout << "\n3 - GoldPlus ";

cout << "\n\nТип страхования: ";

insType = getaNumb(1, 3);

cout << "\nОплаченный период (м): ";

insPeriod = getaNumb(1, 36500);

cout << "\nМесяц (1-12): ";

month = getaNumb(1, 12);

cout << "\nДень (1-31): ";

day = getaNumb(1, 31);

type = 1;

switch (insType)

{

case 1:

amount = insPeriod \* standartPay;

break;

case 2:

amount = insPeriod \* goldPay;

break;

case 3:

amount = insPeriod \* goldPlusPay;

break;

}

cin.clear();

while (cin.get() != '\n');

Client\* ptrClient = new Client(cName, insNumber, insType, insPeriod);

ptrClientList->insertClient(ptrClient);

Income\* ptrIncome = new Income(month, day, type, insNumber, amount);

ptrIncomeRecord->insertIncome(ptrIncome);

}

***ClientList.cpp:***

#include <iostream>

#include "Clinic.h"

#include "ClientList.h"

ClientList::~ClientList()

{

while (!clientPtrList.empty())

{

iter = clientPtrList.begin();

delete\* iter;

clientPtrList.erase(iter);

}

}

void ClientList::insertClient(Client\* clPtr)

{

clientPtrList.push\_back(clPtr);

}

int ClientList::getNumbByName(string clName)

{

int insNo;

iter = clientPtrList.begin();

while (iter != clientPtrList.end())

{

insNo = (\*iter)->getInsNumber();

if (clName == ((\*iter)->getName()))

{

return insNo;

}

iter++;

}

return -1;

}

int ClientList::getTypeByNumb(int insNo)

{

int insType;

iter = clientPtrList.begin();

while (iter != clientPtrList.end())

{

if (insNo == ((\*iter)->getInsNumber()))

{

return (\*iter)->getInsType();

}

iter++;

}

}

void ClientList::changeTypeByNumb(int insNo, int newType)

{

}

void ClientList::prolongPeriodByNumb(int insNo, int newPeriod)

{

iter = clientPtrList.begin();

while (iter != clientPtrList.end())

{

if (insNo == ((\*iter)->getInsNumber()))

{

newPeriod = newPeriod + (\*iter)->getInsPeriod();

(\*iter)->changeInsPeriod(newPeriod);

}

iter++;

}

}

int ClientList::checkNumb(int insNo)

{

int check = -1;

if (clientPtrList.empty())

{

return 0;

}

else

{

iter = clientPtrList.begin();

while (iter != clientPtrList.end())

{

if ((\*iter)->getInsNumber() == insNo) {

check = 1;

}

iter++;

}

return check;

}

}

void ClientList::showClientList()

{

system("cls");

cout << "|| Имя клиента || Номер страховки || Тип страховки || Период (м) ||" << endl;

cout << "||----------------------||-----------------||---------------||------------||" << endl;

if (clientPtrList.empty())

{

cout << "|| ||" << endl;

cout << "|| НЕТ КЛИЕНТОВ ||" << endl;

cout << "||\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_||" << endl;

system("pause");

}

else

{

iter = clientPtrList.begin();

while (iter != clientPtrList.end())

{

cout.width(3);

cout << "|| ";

cout.width(20);

cout << (\*iter)->getName();

cout.width(4);

cout << " || ";

cout.width(15);

cout << (\*iter)->getInsNumber();

cout.width(4);

cout << " || ";

cout.width(13);

switch ((\*iter)->getInsType())

{

case 1:

cout << "Standart";

break;

case 2:

cout << "Gold";

break;

case 3:

cout << "GoldPlus";

break;

default:

cout << "none";

break;

}

cout.width(4);

cout << " || ";

cout.width(10);

cout << (\*iter)->getInsPeriod();

cout.width(3);

cout << " ||" << endl;

\*iter++;

}

cout << "||\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_||\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_||\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_||\_\_\_\_\_\_\_\_\_\_\_\_||" << endl;

cout << endl;

system("pause");

}

}

***Clinic.cpp:***

#include <iostream>

#include "Clinic.h"

string getaLine(int n)

{

string inStr;

char line[42];

cin.get(line, n+1, '\n');

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

inStr = line;

return inStr;

}

char getaChar()

{

char ch = cin.get();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

return ch;

}

int getaNumb(int min, int max)

{

int n;

for (;;)

{

if ((cin >> n).good() && (min <= n) && (n <= max))

return n;

if (cin.fail())

{

cin.clear();

cout << "\nНеверный ввод, повторите\n";

}

else

{

cout << "\nЧисло вне допустимого диапазона значений. Повторите ввод\n";

}

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

}

}

***ClinicMain.cpp:***

#include "Clinic.h"

#include "Client.h"

#include "ClientList.h"

#include "ClientInputScreen.h"

#include "Menu.h"

using namespace std;

int main()

{

setlocale(0, "rus");

Menu menu;

menu.interact();

return 0;

}

***ExpensesInputScreen.cpp:***

#include "Clinic.h"

#include "Client.h"

#include "ExpensesInputScreen.h"

#include "ExpensesRecord.h"

ExpensesInputScreen::ExpensesInputScreen(ExpensesRecord\* per) : ptrExpensesRecord(per)

{

}

void ExpensesInputScreen::setExpense()

{

int month, day;

string provider, description;

float amount;

system("cls");

cout << "Поставщик (фирма): ";

provider = getaLine(18);

cout << "\nОписание: ";

description = getaLine(26);

cout << "\nСумма ($): ";

amount = getaNumb(1, 50000);

cout << "\nМесяц (1-12): ";

month = getaNumb(1, 12);

cout << "\nДень (1-31): ";

day = getaNumb(1, 31);

cin.clear();

while (cin.get() != '\n');

Expenses\* ptrExpenses = new Expenses(month, day, provider, description, amount);

ptrExpensesRecord->insertExpenses(ptrExpenses);

}

***ExpensesRecord.cpp:***

#include "Clinic.h"

#include "Client.h"

#include "ExpensesRecord.h"

ExpensesRecord::~ExpensesRecord()

{

while (!vectPtrsExpenses.empty())

{

iter = vectPtrsExpenses.begin();

delete\* iter;

vectPtrsExpenses.erase(iter);

}

}

void ExpensesRecord::insertExpenses(Expenses\* ptrExp)

{

vectPtrsExpenses.push\_back(ptrExp);

}

void ExpensesRecord::showExpenses()

{

system("cls");

cout << "|| Дата || Получатель || Описание || Сумма ||" << endl;

cout << "||\_\_\_\_\_\_\_||\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_||\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_||\_\_\_\_\_\_\_\_\_\_\_||" << endl;

if (vectPtrsExpenses.size() == 0)

{

cout << "|| ||" << endl;

cout << "|| НЕТ ЗАПИСЕЙ ||" << endl;

cout << "||\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_||" << endl;

}

else

{

iter = vectPtrsExpenses.begin();

while (iter != vectPtrsExpenses.end())

{

cout.width(3);

cout << "|| ";

cout.width(2);

cout << (\*iter)->month;

cout << "/";

if ((\*iter)->day > 9)

{

cout.width(2);

cout << (\*iter)->day;

}

else

{

cout.width(1);

cout << (\*iter)->day << " ";

}

cout.width(4);

cout << " || ";

cout.width(18);

cout << (\*iter)->provider;

cout.width(4);

cout << " || ";

cout.width(26);

cout << (\*iter)->description;

cout.width(4);

cout << " || ";

cout.width(9);

cout << (\*iter)->amount;

cout.width(3);

cout << " ||" << endl;

\*iter++;

}

cout << "||\_\_\_\_\_\_\_||\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_||\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_||\_\_\_\_\_\_\_\_\_\_\_||" << endl;

cout.width(3);

cout << "|| ";

cout.width(17);

cout << "Итог по расходам:";

cout.width(53);

cout << summaryExpenses();

cout.width(3);

cout << " ||" << endl;

cout << "||\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_||" << endl;

}

cout << endl;

system("pause");

}

int ExpensesRecord::summaryExpenses()

{

float totalExpenses = 0;

if (vectPtrsExpenses.size() == 0)

{

return 0;

}

iter = vectPtrsExpenses.begin();

while (iter != vectPtrsExpenses.end())

{

totalExpenses += (\*iter)->amount;

iter++;

}

return totalExpenses;

}

void ExpensesRecord::showExpensesReport() {

if (vectPtrsExpenses.size() == 0)

{

cout << "|| НЕТ РАСХОДОВ ЗА ГОД ||" << endl;

cout << "||\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_||" << endl;

}

else

{

iter = vectPtrsExpenses.begin();

while (iter != vectPtrsExpenses.end())

{

cout.width(3);

cout << "|| ";

cout.width(2);

cout << (\*iter)->month;

cout << "/";

if ((\*iter)->day > 9)

{

cout.width(2);

cout << (\*iter)->day;

}

else

{

cout.width(1);

cout << (\*iter)->day << " ";

}

cout.width(4);

cout << " || ";

cout.width(22);

cout << (\*iter)->description;

cout.width(4);

cout << " || ";

cout.width(22);

cout << (\*iter)->provider;

cout.width(4);

cout << " || ";

cout.width(9);

cout << (\*iter)->amount;

cout.width(3);

cout << " ||" << endl;

\*iter++;

}

cout << "||\_\_\_\_\_\_\_||\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_||\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_||\_\_\_\_\_\_\_\_\_\_\_||" << endl;

cout.width(3);

cout << "|| ";

cout.width(17);

cout << "Итог по расходам:";

cout.width(53);

cout << summaryExpenses();

cout.width(3);

cout << " ||" << endl;

cout << "||\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_||" << endl;

}

}

***IncomeRecord.cpp:***

#include "Clinic.h"

#include "Client.h"

#include "IncomeRecord.h"

IncomeRecord::~IncomeRecord()

{

while (!vectPtrsIncome.empty())

{

iter = vectPtrsIncome.begin();

delete\* iter;

vectPtrsIncome.erase(iter);

}

}

void IncomeRecord::insertIncome(Income\* ptrInc)

{

vectPtrsIncome.push\_back(ptrInc);

}

void IncomeRecord::showIncome()

{

system("cls");

cout << "|| Дата || Тип дохода || Номер страхования || Сумма ||" << endl;

cout << "||\_\_\_\_\_\_\_||\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_||\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_||\_\_\_\_\_\_\_\_\_\_\_||" << endl;

if (vectPtrsIncome.size() == 0)

{

cout << "|| ||" << endl;

cout << "|| НЕТ ЗАПИСЕЙ ||" << endl;

cout << "||\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_||" << endl;

}

else

{

iter = vectPtrsIncome.begin();

while (iter != vectPtrsIncome.end())

{

cout.width(3);

cout << "|| ";

cout.width(2);

cout << (\*iter)->month;

cout << "/";

if ((\*iter)->day > 9)

{

cout.width(2);

cout << (\*iter)->day;

}

else

{

cout.width(1);

cout << (\*iter)->day << " ";

}

cout.width(4);

cout << " || ";

cout.width(27);

if ((\*iter)->type == 1)

{

cout << "Оформление страховки";

}

else

{

cout << "Продление страховки";

}

cout.width(4);

cout << " || ";

cout.width(17);

cout << (\*iter)->insNumb;

cout.width(4);

cout << " || ";

cout.width(9);

cout << (\*iter)->amount;

cout.width(3);

cout << " ||" << endl;

\*iter++;

}

cout << "||\_\_\_\_\_\_\_||\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_||\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_||\_\_\_\_\_\_\_\_\_\_\_||" << endl;

cout.width(3);

cout << "|| ";

cout.width(16);

cout << "Итог по доходам:";

cout.width(54);

cout << summaryIncome();

cout.width(3);

cout<< " ||" << endl;

cout << "||\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_||" << endl;

}

cout << endl;

system("pause");

}

int IncomeRecord::summaryIncome()

{

float totalExpenses = 0;

if (vectPtrsIncome.size() == 0)

{

return 0;

}

iter = vectPtrsIncome.begin();

while (iter != vectPtrsIncome.end())

{

totalExpenses += (\*iter)->amount;

iter++;

}

return totalExpenses;

}

void IncomeRecord::showIncomeReport() {

if (vectPtrsIncome.size() == 0)

{

cout << "|| НЕТ ДОХОДОВ ЗА ГОД ||" << endl;

cout << "||\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_||" << endl;

}

else

{

iter = vectPtrsIncome.begin();

while (iter != vectPtrsIncome.end())

{

cout.width(3);

cout << "|| ";

cout.width(2);

cout << (\*iter)->month;

cout << "/";

if ((\*iter)->day > 9)

{

cout.width(2);

cout << (\*iter)->day;

}

else

{

cout.width(1);

cout << (\*iter)->day << " ";

}

cout.width(4);

cout << " || ";

cout.width(22);

if ((\*iter)->type == 1)

{

cout << "Оформление страховки";

}

else

{

cout << "Продление страховки";

}

cout.width(4);

cout << " || ";

cout.width(15);

cout << "Клиент:";

cout.width(7);

cout << (\*iter)->insNumb;

cout.width(4);

cout << " || ";

cout.width(9);

cout << (\*iter)->amount;

cout.width(3);

cout << " ||" << endl;

\*iter++;

}

cout << "||\_\_\_\_\_\_\_||\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_||\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_||\_\_\_\_\_\_\_\_\_\_\_||" << endl;

cout.width(3);

cout << "|| ";

cout.width(16);

cout << "Итог по доходам:";

cout.width(54);

cout << summaryIncome();

cout.width(3);

cout << " ||" << endl;

cout << "||\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_||" << endl;

}

}

***Menu.cpp:***

#include "Clinic.h"

#include "Client.h"

#include "ClientInputScreen.h"

#include "ClientList.h"

#include "Menu.h"

#include "prolongInsScreen.h"

Menu::Menu()

{

ptrClientList = new ClientList;

ptrExpensesRecord = new ExpensesRecord;

ptrIncomeRecord = new IncomeRecord;

ptrSeeDoctorRecord = new SeeDoctorRecord;

}

Menu::~Menu()

{

delete ptrClientList;

delete ptrExpensesRecord;

delete ptrIncomeRecord;

delete ptrSeeDoctorRecord;

}

void Menu::interact()

{

while (true)

{

system("cls");

cout << " Меню" << endl << endl;

cout << " q) Страхование нового клиента" << endl;

cout << " w) Продление страховки" << endl;

cout << " e) Вывести список клиентов" << endl << endl;

cout << " a) Запись клиента к врачу" << endl;

cout << " s) Вывести записи к врачам" << endl << endl;

cout << " d) Ввести расходы" << endl;

cout << " z) Вывести расходы" << endl;

cout << " x) Вывести доходы" << endl;

cout << " c) Вывести годовой отчёт" << endl << endl;

cout << " i) EXIT" << endl;

ch = getaChar();

system("cls");

switch (ch)

{

case 'q':

ptrClientInputScreen = new ClientInputScreen(ptrClientList, ptrIncomeRecord);

ptrClientInputScreen->setClient();

delete ptrClientInputScreen;

break;

case 'w':

ptrProlongInsScreen = new ProlongInsScreen(ptrClientList, ptrIncomeRecord);

ptrProlongInsScreen->ProlongIns();

delete ptrProlongInsScreen;

break;

case 'e':

ptrClientList->showClientList();

break;

case 'a':

ptrSeeDoctorInputScreen = new SeeDoctorInputScreen(ptrSeeDoctorRecord, ptrClientList);

ptrSeeDoctorInputScreen->setSeeDoctor();

delete ptrSeeDoctorInputScreen;

break;

case 's':

ptrSeeDoctorRecord->showSeeDoctor();

break;

case 'd':

ptrExpensesInputScreen = new ExpensesInputScreen(ptrExpensesRecord);

ptrExpensesInputScreen->setExpense();

delete ptrExpensesInputScreen;

break;

case 'z':

ptrExpensesRecord->showExpenses();

break;

case 'x':

ptrIncomeRecord->showIncome();

break;

case 'c':

ptrAnnualReportScreen = new AnnualReportScreen(ptrIncomeRecord, ptrExpensesRecord);

ptrAnnualReportScreen->showAnnualReport();

delete ptrExpensesInputScreen;

break;

case 'i':

return;

default:

cout << "Неизвестная функция\n";

system("pause");

break;

}

}

}

***ProlongInsScreen.cpp:***

#include "Clinic.h"

#include "Client.h"

#include "ClientList.h"

#include "ProlongInsScreen.h"

void ProlongInsScreen::ProlongIns()

{

int error = 0;

int month, day, type, amount;

system("cls");

cout << "Продление страховки: ";

cout << "\n\nНомер страхования клиента: ";

insNumber = getaNumb(111111, 999999);

while (true)

{

if (ptrClientList->checkNumb(insNumber) == -1)

{

cout << "\nВ базе нет такого клиента, повторите ввод: ";

insNumber = getaNumb(111111, 999999);

}

else if (ptrClientList->checkNumb(insNumber) == 0)

{

error = 1;

cout << "\nВ базе нет клиентов\n\n";

break;

}

else

{

break;

}

}

if (error == 0)

{

cout << "\nОплаченный период (м): ";

insPeriod = getaNumb(1, 36500);

cout << "\nМесяц (1-12): ";

month = getaNumb(1, 12);

cout << "\nДень (1-31): ";

day = getaNumb(1, 31);

type = 2;

switch (ptrClientList->getTypeByNumb(insNumber))

{

case 1:

amount = insPeriod \* standartPay;

break;

case 2:

amount = insPeriod \* goldPay;

break;

case 3:

amount = insPeriod \* goldPlusPay;

break;

}

ptrClientList->prolongPeriodByNumb(insNumber, insPeriod);

cout << "\nПериод страхования продлен на " << insPeriod << " м\n\n";

Income\* ptrIncome = new Income(month, day, type, insNumber, amount);

ptrIncomeRecord->insertIncome(ptrIncome);

}

system("pause");

cin.clear();

while (cin.get() != '\n');

}

***SeeDoctor.cpp:***

#include "Clinic.h"

#include "Client.h"

#include "SeeDoctor.h"

int SeeDoctor::getDoctor()

{

return category;

}

int SeeDoctor::getService()

{

return service;

}

***SeeDoctorInputScreen.cpp:***

#include "Clinic.h"

#include "Client.h"

#include "ClientList.h"

#include "Menu.h"

#include "SeeDoctorInputScreen.h"

#include "SeeDoctorRecord.h"

SeeDoctorInputScreen::SeeDoctorInputScreen(SeeDoctorRecord\* per, ClientList\* clli) : ptrSeeDoctorRecord(per), ptrClientList(clli)

{

}

void SeeDoctorInputScreen::setSeeDoctor()

{

int month, day, insNo;

int category, service;

int error = 0, insType;

cout << "Запись к врачу:\n";

cout << "\nНомер страхования: ";

insNo = getaNumb(111111, 999999);

while (true)

{

if (ptrClientList->checkNumb(insNo) == -1)

{

cout << "\nВ базе нет такого клиента, повторите ввод: ";

insNo = getaNumb(111111, 999999);

}

else if (ptrClientList->checkNumb(insNo) == 0)

{

error = 1;

break;

}

else

{

break;

}

}

insType = ptrClientList->getTypeByNumb(insNo);

if (error == 0)

{

cout << "\nВыберите врача: ";

cout << "\n1 - Ортопед ";

cout << "\n2 - Дерматолог ";

cout << "\n3 - Стоматолог ";

cout << "\n4 - Врач спортивной медицины ";

cout << "\n5 - Окулист\n";

category = getaNumb(1, 5);

if (category == 1)

{

cout << "\nВыберите услугу: ";

cout << "\n1 - Консультация (S)";

cout << "\n2 - Повторный приём (S)";

cout << "\n3 - Профилактика и лечение грыж (G)";

cout << "\n4 - Изготовление стелек и фиксаторов (G+)\n";

service = getaNumb(1, 4);

if (((insType == 1) && ((service == 3) || (service == 4))) || ((insType == 2) && (service == 4)))

{

error = 1;

}

}

else if (category == 2)

{

cout << "\nВыберите услугу: ";

cout << "\n1 - Консультация (S)";

cout << "\n2 - Повторный приём (S)";

cout << "\n3 - Направление на анализы (S)";

cout << "\n4 - Чистка кожи (G)";

cout << "\n5 - Косметические дефекты (G+)\n";

service = getaNumb(1, 5);

if (((insType == 1) && ((service == 4) || (service == 5))) || ((insType == 2) && (service == 5)))

{

error = 1;

}

}

else if (category == 3)

{

cout << "\nВыберите услугу: ";

cout << "\n1 - Консультация (S)";

cout << "\n2 - Повторный приём (S)";

cout << "\n3 - Удаление зуба (S)";

cout << "\n4 - Установка имплантов (G)";

cout << "\n5 - Установка брекет-системы (G+)";

cout << "\n6 - Установка виниров (G+)\n";

service = getaNumb(1, 6);

if (((insType == 1) && ((service == 4) || (service == 5) || (service == 6))) || ((insType == 2) && ((service == 5) || (service == 6))))

{

error = 1;

}

}

else if (category == 4)

{

cout << "\nВыберите услугу: ";

cout << "\n1 - Консультация (S)";

cout << "\n2 - Повторный приём (S)";

cout << "\n3 - ЛФК процедуры (G)";

cout << "\n4 - Массаж/мануальные техники (G)";

cout << "\n5 - Реабилитация (G+)\n";

service = getaNumb(1, 5);

if (((insType == 1) && ((service == 3) || (service == 4) || (service == 5))) || ((insType == 2) && (service == 5)))

{

error = 1;

}

}

else if (category == 5)

{

cout << "\nВыберите услугу: ";

cout << "\n1 - Консультация (S)";

cout << "\n2 - Повторный приём (S)";

cout << "\n3 - Проверка зрения (G)";

cout << "\n4 - Коррекция зрения (G+)";

cout << "\n5 - Корреция линз очков (G+)\n";

service = getaNumb(1, 5);

if (((insType == 1) && ((service == 3) || (service == 4) || (service == 5))) || ((insType == 2) && ((service == 4) || (service == 5))))

{

error = 1;

}

}

if (error == 0)

{

cout << "\nВведите месяц (1-12): ";

month = getaNumb(1, 12);

cout << "\nВведите день (1-31): ";

day = getaNumb(1, 31);

cin.clear();

while (cin.get() != '\n');

SeeDoctor\* ptrSeeDoctor = new SeeDoctor(month, day, category, service, insNo);

ptrSeeDoctorRecord->insertSeeDoctor(ptrSeeDoctor);

}

else

{

cout << "\nЭта услуга недоступна при таком типе страхования\n\n";

system("pause");

cin.clear();

while (cin.get() != '\n');

}

}

else

{

cout << "\nВ базе нет клиентов\n\n";

system("pause");

cin.clear();

while (cin.get() != '\n');

}

}

***SeeDoctorRecord.cpp:***

#include "Clinic.h"

#include "Client.h"

#include "SeeDoctorRecord.h"

SeeDoctorRecord::~SeeDoctorRecord()

{

while (!vectPtrSeeDoctor.empty())

{

iter = vectPtrSeeDoctor.begin();

delete\* iter;

vectPtrSeeDoctor.erase(iter);

}

}

void SeeDoctorRecord::insertSeeDoctor(SeeDoctor\* ptrExp)

{

vectPtrSeeDoctor.push\_back(ptrExp);

}

void SeeDoctorRecord::showSeeDoctor()

{

system("cls");

cout << "|| Дата || Номер страховки || Врач || Услуга ||" << endl;

cout << "||\_\_\_\_\_\_\_||\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_||\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_||\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_||" << endl;

if (vectPtrSeeDoctor.size() == 0)

{

cout << "|| ||" << endl;

cout << "|| НЕТ ЗАПИСЕЙ ||" << endl;

cout << "||\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_||" << endl;

}

else

{

iter = vectPtrSeeDoctor.begin();

while (iter != vectPtrSeeDoctor.end())

{

cout.width(3);

cout << "|| ";

cout.width(2);

cout << (\*iter)->month;

cout << "/";

if ((\*iter)->day > 9)

{

cout.width(2);

cout << (\*iter)->day;

}

else

{

cout.width(1);

cout << (\*iter)->day << " ";

}

cout.width(4);

cout << " || ";

cout.width(15);

cout << (\*iter)->insNo;

cout.width(4);

cout << " || ";

cout.width(30);

switch ((\*iter)->getDoctor())

{

case 1:

cout << "Ортопед";

break;

case 2:

cout << "Дерматолог";

break;

case 3:

cout << "Стоматолог";

break;

case 4:

cout << "Врач спортивной медицины";

break;

case 5:

cout << "Окулист";

break;

default:

cout << "none";

break;

}

cout.width(4);

cout << " || ";

cout.width(30);

if((\*iter)->category == 1)

switch ((\*iter)->getService())

{

case 1:

cout << "Консультация";

break;

case 2:

cout << "Повторный приём";

break;

case 3:

cout << "Профилактика и лечение грыж";

break;

case 4:

cout << "Изготовление стелек";

break;

default:

cout << "none";

break;

}

if ((\*iter)->category == 2)

switch ((\*iter)->getService())

{

case 1:

cout << "Консультация";

break;

case 2:

cout << "Повторный приём";

break;

case 3:

cout << "Направление на анализы";

break;

case 4:

cout << "Чистка кожи";

break;

case 5:

cout << "Косметические дефекты";

break;

default:

cout << "none";

break;

}

if ((\*iter)->category == 3)

switch ((\*iter)->getService())

{

case 1:

cout << "Консультация";

break;

case 2:

cout << "Повторный приём";

break;

case 3:

cout << "Удаление зуба";

break;

case 4:

cout << "Установка имплантов";

break;

case 5:

cout << "Установка брекет-системы";

break;

case 6:

cout << "Установка виниров";

break;

default:

cout << "none";

break;

}

if ((\*iter)->category == 4)

switch ((\*iter)->getService())

{

case 1:

cout << "Консультация";

break;

case 2:

cout << "Повторный приём";

break;

case 3:

cout << "ЛФК процедуры";

break;

case 4:

cout << "Массаж/мануальные техники";

break;

case 5:

cout << "Реабилитация";

break;

default:

cout << "none";

break;

}

if ((\*iter)->category == 5)

switch ((\*iter)->getService())

{

case 1:

cout << "Консультация";

break;

case 2:

cout << "Повторный приём";

break;

case 3:

cout << "Проверка зрения";

break;

case 4:

cout << "Коррекция зрения";

break;

case 5:

cout << "Корреция линз очков";

break;

default:

cout << "none";

break;

}

cout.width(3);

cout << " ||" << endl;

iter++;

}

cout << "||\_\_\_\_\_\_\_||\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_||\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_||\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_||" << endl;

}

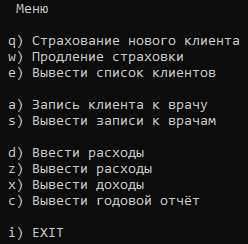
cout << endl;

system("pause");

}

2.15 Руководство пользователя

При запуске программы показывается меню. В нем указаны все варианты использования программы пользователем

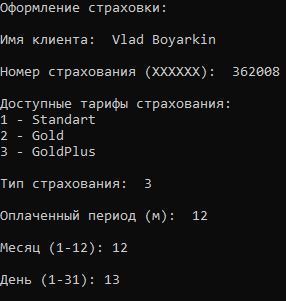


Для того чтобы запустить нужную функцию необходимо ввести в строку соответствующий ей символ и нажать «Enter».

q) Страхование нового клиента

При нажатии «q» выводится форма страхования нового клиента.

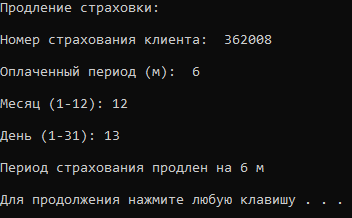
Здесь необходимо последовательно через «Enter» ввести: имя клиента, его номер страховки, тип страховки, уже оплаченный клиентом период, сегодняшнюю дату.



w) Продление страховки

При нажатии «w» выводится форма продления страховки клиенту.

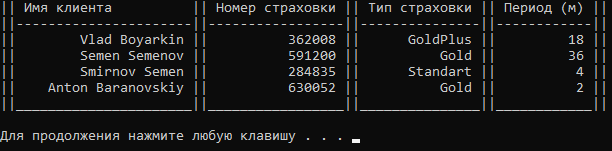
Здесь необходимо последовательно через «Enter» ввести: номер страховки клиента, оплаченный этим клиентом период, сегодняшнюю дату.



e) Вывести список клиентов

При нажатии «e» выводится список клиентов в таблице.

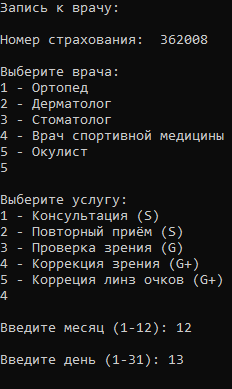
Здесь можно увидеть соответствующие им имена, их номера страхования, типы страховки, период страхования.



a) Запись клиента к врачу

При нажатии «a» выводится форма записи клиента к врачу.

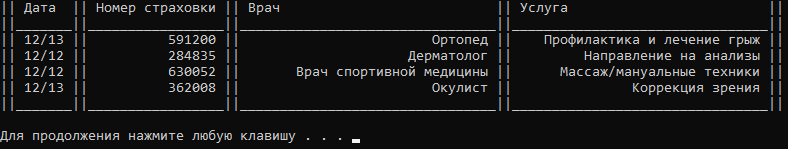
Здесь необходимо указать номер страхования клиента, выбрать из списка врача, к которому записывается клиент, выбрать услугу, в соответствии с типом страхования клиента, указать дату приема.



s) Вывести записи к врачам

При нажатии «s» выводится журнал записи к врачам в виде таблицы.

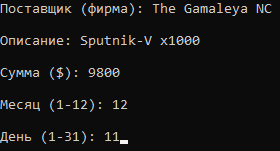
Здесь можно увидеть дату приема, номер страхования записанного клиента, выбранных врача и услугу.



d) Ввести расходы

При нажатии «d» выводится форма ввода расходов.

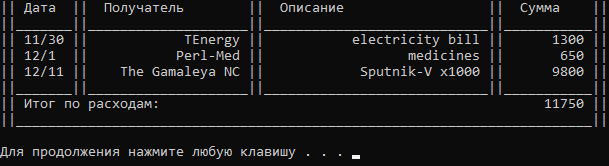
Здесь необходимо последовательно через «Enter» ввести: фирму, предоставляющую услуги или поставляющую товары, описание расхода, сумму, дату.



z) Вывести расходы

При нажатии «z» выводится список расходов в таблице.

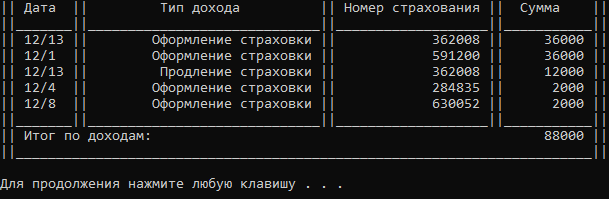
Здесь можно увидеть дату сделки, получателя средств, описание расхода клиники, сумму, а также сумму всех расходов.



x) Вывести доходы

При нажатии «x» выводится список доходов в таблице.

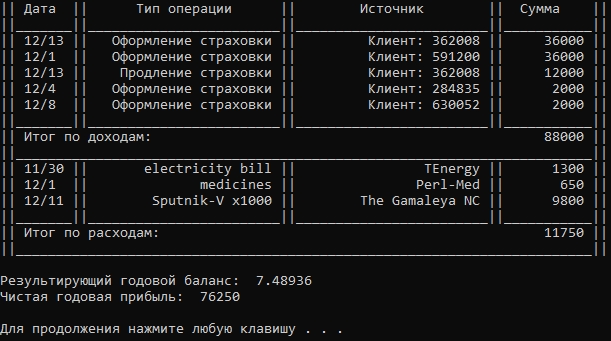
Здесь можно увидеть дату получения средств, тип дохода, номер страхования плательщика, сумму, а также сумму всех доходов.



c) Вывести годовой отчет

При нажатии «с» выводится список доходов и расходов в одной таблице.

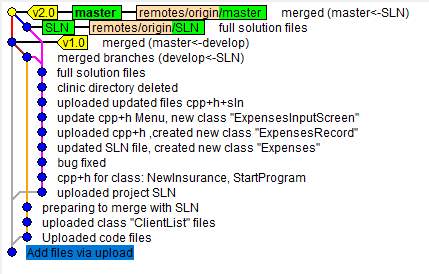
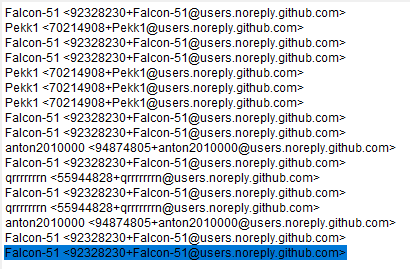
Здесь можно увидеть дату операции, ее тип, источник, суммы всех доходов и расходов, результирующий годовой баланс и чистую прибыль.



3 История проекта на GitHub

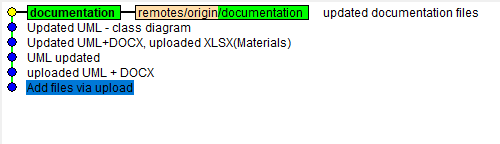
Ветка master

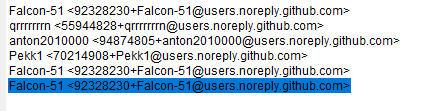
<https://github.com/Falcon-51/Project_Clinic>



* Add files via upload - загружен файл .gitignore (Семёнов)
* Uploaded code files - Добавил главные файлы, а также h и cpp файл для класса Client (Семёнов)
* uploaded class "ClientList" files - cpp+h: Добавил класс ClientList (Барановский)
* preparing to merge with SLN - влил ветку SLN в develop;обновил файлы;решил конфликты слияния(Смирнов)
* uploaded project SLN - залита сборка(Семёнов)
* cpp+h for class: NewInsurance(ClientInputScreen), StartProgram(Menu) - Рабочая версия программы, исправлены баги, написана стартовая страница с меню. Функционал: добавление клиентов в таблицу и вывод таблицы на экран. (Смирнов)
* bug fixed - Исправил баг в начальном меню. Отредактировал вывод клиентов. (Семёнов)
* updated SLN file, created new class "Expenses" - Добавил новый класс "Expenses"(Барановский)
* uploaded cpp+h ,created new class "ExpensesRecord" - Добавил новый класс "ExpensesRecord".(Семёнов)
* update cpp+h Menu, new class "ExpensesInputScreen" - Добавил строки в cpp+h Menu. Создал класс "ExpensesInputScreen"(Семёнов)
* uploaded updated files cpp+h+sln - Переработка меню. Фикс багов. Сделал таблицы вывода (Бояркин)
* clinic directory deleted – Удалил лишнюю директорию на ветке (Бояркин)
* full solution files – Загрузил файлы проекта (Бояркин)
* merged branches (develop<-SLN) - Объединил ветки develop и SLN (Семёнов)
* uploaded cpp+h files "SeeDoctor" - Реализовал запись к врачу и вывод списка записи. Отредактировал класс Menu.(Семёнов)
* merged (master<-develop) - Объединил ветки master и develop. Стабильная версия (Семёнов)
* upload h+cpp IncomeRecord, ProlongInsScreen, AnnualReportScreen - Привел код к единому стилю перед релизом. Косметические изменения в программе. Исправлены баги, потоки ввод очищаются. Добавил: продление страховки, вывод доходов, годовой отчет, функция ввода с проверкой по диапазону getaNumb. Доходы теперь формируются автоматически при оформлении и продлении страховки. Проверка на тип страховки при записи на ВИП-услуги (Бояркин)
* update seeDoctor files – отредактировал файлы seeDoctor (Бояркин)
* full solution files – загрузил все файлы проекта (Бояркин)
* merged (master<-SLN) - final version files (Семёнов)

Ветка documentation





* Add files via upload – загрузил файл gitignore (Семёнов)
* uploaded UML + DOCX - UML: Сделаны диаграмма вариантов использования и 3 диаграммы действий. DOCX: Шаблон отчета (Семёнов)
* UML updated - Изменена диаграмма вариантов использования и диаграмма активности ExtendInsurance (Бояркин)
* Updated UML+DOCX, uploaded XLSX(Materials) - Добавил файл Excel - данные с которыми работал заказчик. Дополнил пункты в отчёте. Отредактировал диаграмму вариантов использования и описания к прецедентам. Добавил диаграмму последовательности. (Барановский)
* Updated UML - class diagram - Добавил 2 диаграммы последовательности - income и expenses. Добавил диаграмму классов (Client, ClientList) (Смирнов)
* updated documentation files - Добавил диаграмму последовательности. Сделал некоторые пункты(отмечены в содержании) (Семёнов)
* updated documentation files – загрузил финальные версии файлов (Семёнов)

Заключение

В ходе выполнения командного задания вся работа велась в VC 2019 на ЯП С++. Для удобной работы всех членов команды использовалась СКВ GitHub. На момент завершения работы все прецеденты были реализованы. Программа не имеет сбоев и зависаний. В разработке программы использовался принцип раздельной компиляции и очистка памяти. Программа не имеет неиспользуемых переменных, а её алгоритмы не избыточны ненужными циклами, массивами и т.д. В отчёте представлены диаграммы вариантов использования, действий, классов и диаграммы последовательностей. По окончанию работы можно сделать вывод, что преследуемая цель достигнута, несмотря на трудности. Наиболее обширными были вылеты программы из-за переполнения памяти и обработка исключительных ситуаций.