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

Пекарня

Командный проект по дисциплине «Технология программирования»

Авторы работы:

студенты группы 21312

Запорожец Артём

Шаркевич Арсений

Рокотянский Александр

Смирнов Алексей

Научный руководитель:

канд. физ.-мат. наук, доцент КИИСиФЭ А. В. Бульба

Петрозаводск 2023

**Цель работы**

В рамках командного задания, разработать программное обеспечение для обслуживания пекарни.

Для достижения данной цели, мы применяем язык моделирования UML, благодаря чему о в нашем отчете представлены важные диаграммы вариантов использования, деятельности, классов и последовательностей. При этом, мы также регулярно фиксируем все промежуточные результаты работы, используя систему контроля версий Git. Для удобства совместной работы, мы создали открытый репозиторий на платформе GitHub, где всегда можно найти актуальную версию нашего проекта. Ниже представлена последняя версия данного репозитория.

**В процессе работы были обозначены следующие правила:**

1. **По составлению названий:**
   1. Говорящие имена классов, функций, переменных.
   2. Имена классов должны быть существительными, соответствующими понятиям предметной области.
   3. Имена классов должны начинаться с заглавной буквы.
   4. Имена атрибутов и операций должны начинаться с маленькой буквы.
   5. Составные имена должны быть сплошными, без подчёркиваний, каждое отдельное слово должно начинаться с заглавной буквы.
2. **По разработке приложения:**
   1. Программа не должна вызывать системных сбоев или зависать.
   2. Программа должна освобождать всю выделенную память.
   3. Должны отсутствовать конструкции, приводящие к неопределённому поведению.
   4. Переменные должны быть локальны. Не должно быть неиспользуемых переменных.
   5. Алгоритмы не должны быть избыточны.
   6. Классы необходимо реализовывать в отдельных файлах.
   7. Необходимо отделять интерфейс от реализации.
   8. Необходимо использовать STL-контейнеры и итераторы.
   9. Избегать, по возможности, слишком длинных функций и файлов с кодом.
3. **По контролю версий:**
4. Не изменять коммиты, которые попали в публичный репозиторий.
5. Ветвь «master» должна содержать стабильный код. Программа должна собираться и запускаться в любой момент истории ветви.
6. Ветвь «master» должна содержать тэг, в котором указывается функционал, реализованный в данной версии.
7. В репозиторий не отправляются файлы, создаваемые при компиляции и файлы пользовательских настроек IDE. Задействуется файл «.gitignore».
8. При каждом начале работы используется git pull во избежание конфликтов.

**Программная реализация**

В качестве среды разработки использовалась программа “Qt Creator” на Qt 5.4.2 (MSVC 2013, 32 бита).

Программа написана на языке программирования высокого уровня C++.

**Заголовочные файлы и их краткое описание:**

**"libs.h"**

В этом файле подключаются все используемые библиотеки (iostream, vector, list, string, numeric). Впоследствии он подключается к заголовочным файлам базовых классов (Expense.h, Produce.h, Profit.h)

**"AnnualReport.h"**

Хранит интерфейсную часть класса AnnualReport (Экран вывода годового отчёта). Объявлены указатели на записи о доходах и расходах. Хранит в себе объявление метода display() для отображения годового отчета.

**"Expense.h"**

Хранит интерфейсную часть класса Expense (Расход). Используется для ввода и получения информации о затратах (категория; название; дата; сумма расхода). Объявлены методы GetAmount() (используется для получения суммы затрат), GetCategory() (используется для получения категории), GetAmount() (используется для получения информации о сумме расходов), GetDate() (используется для получения даты расхода)

**"ExpenseInputScreen.h"**

Хранит интерфейсную часть класса ExpenseInputScreen (Экран ввода расходов). Хранит в себе объявление указателя типа ExpenseList на записи о расходах. Объявлен метод SetExpense() для добавления данных о расходе.

**"ExpenseList.h"**

Хранит интерфейсную часть класса ExpenseList (Таблица расходов). Хранит контейнер и итератор класса Expense. Объявлены методы InsertExpense(Expense\*) (необходим для добавления расхода), DisplaySummExpenses() (необходим для вывода всех расходов для годового отчета), DisplayExp() (вывести строку расходов), геттеры: GetCategoryExpense(string) (получить категорию расходов), string GetDateExpense(string) (получить дату расхода), float GetAmountExpense(string) (получить сумму расхода)

**"Profit.h"**

Хранит интерфейсную часть класса Profit (доходы). Используется для ввода и получения информации о доходах (название; количество; сумма; дата). Объявлены методы GetName() (нужен для возврата названия дохода), GetQuantity() (нужен для возврата количества), GetGlobalcost() (возврат суммы), GetDate() (возврат даты)

**"ProfitInputScreen.h"**

Хранит интерфейсную часть класса ProfitInputScreen (Экран ввода доходов). Хранит в себе объявление указателя типа ProfitList на записи о доходах. Объявлен метод SetProfit() для добавления данных о доходе.

**"ProfitList.h"**

Хранит интерфейсную часть класса ProfitList (Таблица доходов). Хранит контейнер и итератор класса Profit. Вектор отвечает за указатели на класс доходов. Объявлены методы insertProfit(Profit\*) (необходим для добавления дохода в вектор), DisplayProf() (необходим для вывода списка доходов) и DisplaySummProfits () (используется при составлении годового отчета), геттеры: GetQuantityProfit(string) (возвращает количество доходов), GetGlobalcostProfit(string) (возвращает сумму всех доходов), GetDateProfit(string) (возвращает дату расходов).

**"Produce.h"**

Хранит интерфейсную часть класса Produce (Продукция). Используется для ввода и получения информации о продукции (название; id; цена). Объявлены методы GetName() (нужен для возврата суммы дохода), GetId() (нужен для возврата id), GetCost() (возврат цены продукции).

**"ProduceInputScreen.h"**

Хранит интерфейсную часть класса ProduceInputScreen (Экран ввода/удаления в прайс-лист). Объявлены указатель на ProduceList и метод setProduce () (добавить данные о продукции).

**"ProduceList.h"**

Хранит интерфейсную часть класса ProduceList (Список продукции). Объявлен список и итератор на класс Produce. Объявлены методы InsertProduce(Produce\*) (добавить продукцию в список), RemoveProduce(Produce\*) (удалить продукцию в списке), DisplayProd() (вывод списка продукции) и геттеры: GetIdProduce(string) (возвращает id продукции) и GetCostProduce(string) (возвращает стоимость продукции).

**"UserInterface.h"**

Хранит интерфейсную часть класса UserInterface (Экран пользовательского интерфейса). Здесь подключаются все заголовочные файлы программы. Объявлены указатели классов ProduceList, ProduceInputScreen, ProfitList, ProfitInputScreen, ExpenseList, ExpenseInputScreen, AnnualReport. Объявлен метод interact() (для взаимодействия пользователя с программой).

## **Файлы реализации и их краткое описание:**

**"AnnualReport.cpp"**

Файл реализации класса AnnualReport. В нем реализован метод AnnualReport::display(), в котором выводится информация о доходах, расходах и разницы между ними.

**"Expense.cpp"**

Файл реализации класса Expense. Реализован методы Expense::GetCategory() в котором происходит получение категории, Expense::GetName() в котором происходит получение названия расхода, Expense::GetDate() в котором происходит получение информации о дате расхода и Expense::GetAmount() в котором происходит получение информации о сумме расходов.

**"ExpenseInputScreen.cpp"**

Файл реализации класса ExpenseInputScreen. Реализован метод ExpenseInputScreen::SetExpense (), в котором мы можем заполнить поля объекта.

**"ExpenseList.cpp"**

Файл реализации класса ExpenseList. Реализован деструктор для удаления указателей (а соответственно и самого элемента списка) из контейнера. Реализованы методы ExpenseList::InsertExpense(Expense\*) в котором происходит вставка нового элемента в контейнер, ExpenseList::DisplayExp() в котором выводятся все расходы по порядку, проходя итератором по списку и ExpenseList::DisplaySummExpenses() в котором выводятся сумма всех расходы.

**"Profit.cpp"**

Файл реализации класса Income. Реализованы методы Profit::GetName() в котором происходит получение названия дохода, Profit::GetQuantity() в котором происходит получение информации о количестве доходов, Profit::GetGlobalcost() в котором происходит получение информации о сумме доходов и Profit::GetDate() в котором происходит получение информации о дате доходов.

**"ProfitInputScreen.cpp"**

Файл реализации класса ProfitInputScreen. Реализован метод ProfitInputScreen::setProfit (), в котором мы можем заполнить поля объекта.

**"ProfitList.cpp"**

Файл реализации класса ProfitList. Реализован метод ProfitList:: insertProfit в котором происходит вставка нового элемента в контейнер. Реализованы методы ProfitList::DisplaySummProfits (), используемый при составлении годового отчета, ProfitList::DisplayProf(), для вывода строки дохода и геттеры: GetQuantityProfit(string), GetGlobalcostProfit(string), GetDateProfit(string).

**"main.cpp"**

Основной файл программы, где создан объект класса UserInterface.

**"Produce.cpp"**

Файл реализации класса Produce. Реализованы методы Produce::GetName() для возврата названия продукции, Produce::GetId() для возврата id продукции, Produce::GetCost() для возврата цены продукции.

**"ProduceInputScreen.cpp"**

Файл реализации класса ProduceInputScreen. Реализован метод ProduceInputScreen::setProduce(), в котором мы можем заполнить поля объекта. После того, как мы заполнили все пункты, создается конкретный объект с помощью конструктора c параметром ProduceInputScreen(ProduceList\* ptrPL) и в него передается указатель на список продукции.

**"ProduceList.cpp"**

Файл реализации класса ProduceList. Реализован деструктор для удаления указателей (а соответственно и самого элемента списка) из контейнера. Реализованы методы ProduceList::InsertProduce(Produce\*) в котором происходит вставка новой продукции в контейнер, ProduceList::RemoveProduce(Produce\*) в котором происходит удалении занесенной продукции из контейнера, ProduceList::DisplayProd() в котором выводятся все товары по порядку, проходя итератором по списку продукции и геттеры: GetIdProduce(string) (ID-код продукции), GetCostProduce(string) (цена товара).

**"UserInterface.cpp"**

# Файл реализации класса UserInterface. Реализован метод UserInterFace::interact для взаимодействия пользователя с интерфейсом.

# **Пошаговое описание процесса разработки**

## **Предметная область**

К нам обратился владелец одной небольшой пекарни. Заказчик хочет, чтобы мы написали программу для автоматизации процессов работы пекарни. В распоряжении заведения есть отдел, занимающийся приготовлением изделий и точка сбыта. Пекарня, соответственно, занимается приготовлением хлебобулочных изделий по личным рецептам и их продажей. Требуется реализовать регистрацию и отслеживание доходов и расходов, ведения списка продукции (прайс-листа).

## **Рукописные формы**

В данный момент владелец заведения записывает всю информацию в следующих таблицах:

**1. Прайс-лист с продукцией пекарни** - содержит информацию о наименовании и цене продукции.

|  |  |  |
| --- | --- | --- |
| Идентификатор продукта | Наименование | Цена оптовая  (руб.) |
| 1 | Хлеб пшеничный | 17 |
| 2 | Хлеб ячменный | 26 |
| 3 | Хлеб «Дарницкий» | 37 |
| 4 | Хлеб «Бородинский» | 44 |
| 5 | Хлеб «Рижский» | 55 |

**2.Расходы**

Таблица расходов содержит такие столбцы: название расхода, сумма, дата и его категория.

|  |  |  |  |
| --- | --- | --- | --- |
| Расходы |  |  |  |
| Название расхода | Сумма расхода(руб.) | Дата расхода | Категория расхода |
| Электроэнергия | 1000 | 10.12.2023 | ЖКХ |
| Тесто | 5000 | 10.12.2023 | Сырьё |
| Затраты на воду | 250 | 10.12.2023 | ЖКХ |
| Затраты на воду | 500 | 10.12.2023 | Сырьё |

**Доход**

Здесь хранятся записи о платежах клиентов за оказанные услуги. При каждой продаже товара записывается заплаченная сумма в соответствующую ячейку таблицы

|  |  |  |  |
| --- | --- | --- | --- |
| Доход |  |  |  |
| Название дохода | Количество проданных товаров(шт.) | Полная стоимость покупки (руб.) | Дата дохода |
| Хлеб пшеничный | 1000 | 17000 | 10.12.2023 |

В годовом отчете используется информация, как из таблицы доходов, так и из таблицы расходов для подсчета сумм, пришедших за год от клиентов и заплаченных в процессе ведения бизнеса. Суммируются все доходы и все расходы, из доходов вычитаются расходы, в результате чего получается значение чистой годовой прибыли.

|  |  |
| --- | --- |
| Расходы | (руб.) |
| ЖКХ | 1250 |
| Сырьё | 5500 |
| … | .. |
| Доходы |  |
| Хлеб пшеничный | 17000 |
| … | … |
| Итого |  |
| Общий расход | 2250 |
| Общий доход | 17000 |
| Прибыль | 14750 |

## **Действующие субъекты (актеры)**

С программой будут взаимодействовать работники пекарни. Они будут вводить информацию в таблицы и запрашивать данные из них. Больше никто не будет взаимодействовать с программой и программа не будет взаимодействовать с другими программами.

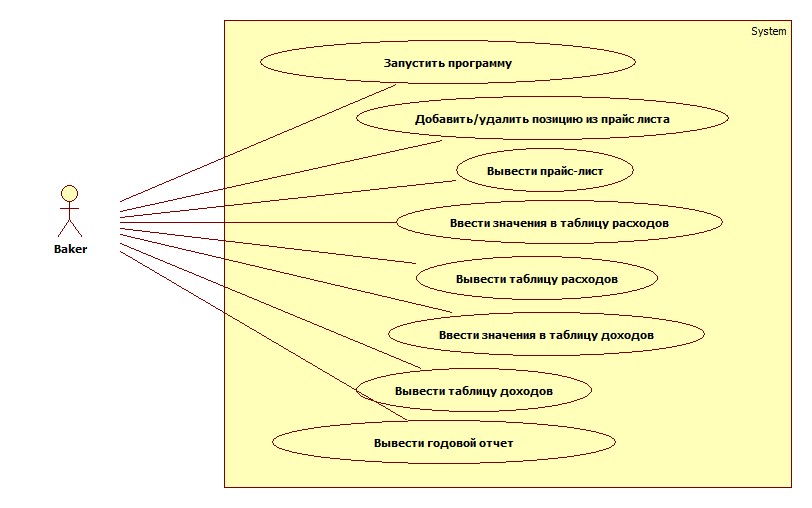
## 

## **Варианты использования**

Работнику пекарни потребуется выполнять следующие действия:

* начать работу с программой;
* добавить новую продукцию в список продукции;
* ввести новый доход в таблицу доходов от продажи;
* ввести новый расход в таблицу расходов;
* вывести список продукции;
* вывести таблицу доходов;
* вывести таблицу расходов;
* вывести годовой отчет.

**Диаграмма вариантов использования:**



## **Описание вариантов использования**

“Начать программу”:

* Когда запускается программа, на экран должно выводиться меню, из которого пользователь может выбрать нужное действие.

“Добавить/удалить продукцию из прайс листа”:

* Выводится сообщение с веткой выбора: добавить продукцию в прайс-лист или удалить. После этого программа просит ввести наименование товара, и если его ещё нет в таблице - позиция добавляется, если она есть - удаляется.

“Ввести значения в таблицу расходов”:

* На экран выводится сообщение с предложением выбора категории расходов. После этого открывается окно ввода, куда требуется ввести конкретный пункт расходов, месяц оплаты и потраченную сумму. Программа просматривает нужную таблицу и по выбранной категории и пункту расходов находит соответствующую запись в таблице. Если такой пункт уже существует, то запись вводится в существующую строку таблицы, в обратном случае - создаётся новая строка и в неё вводится запись.

“Вывести таблицу расходов”:

* Программа выводит на экран таблицу расходов, каждая строка списка состоит из четырёх полей: категории, пункт расходов, месяц и суммы расходов.

“Ввести значения в таблицу доходов;”

* На экран выводится окно ввода, куда требуется ввести наименование продукции,количество проданной продукции, месяц и потраченную сумму.

“Вывести таблицу доходов”

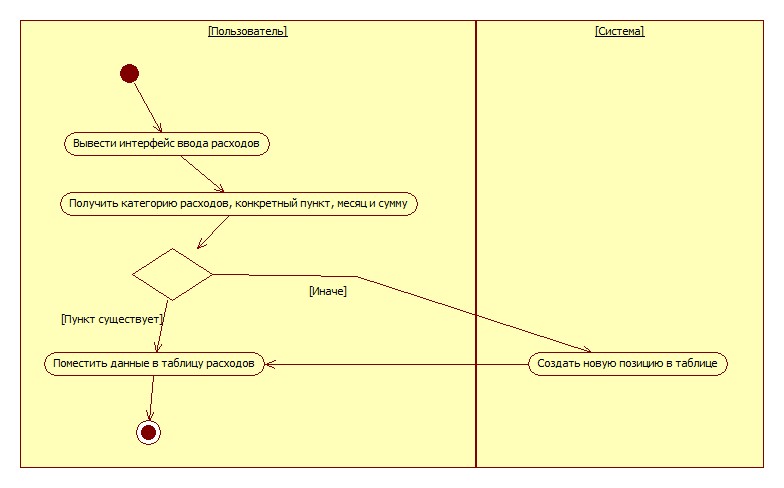
* Программа выводит на экран таблицу расходов, каждая строка списка состоит из четырёх полей: продукции, количества, даты и суммы доходов.

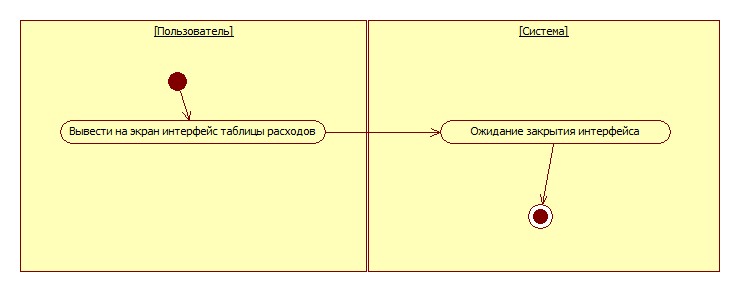
“Вывести годовой отчет”

* Программа выводит годовой отчет, состоящий из:   
  суммарного дохода за прошедший год;   
  списка всех расходов с указанием бюджетной категории;   
  общей суммы расходов;   
  результирующего годового баланса (доходы/убытки).

**Диаграммы действий:**

1.Ввести значения в таблицу расходов

****2.Вывести таблицу расходов



## **Список существительных**

1. Экран пользовательского интерфейса
2. Ингредиент
3. Продукция
4. Пекарь
5. Покупатель
6. Расход
7. Доход
8. Отчет
9. Поставка
10. Баланс
11. Месяц
12. Дата
13. Экран ввода доходов
14. Экран ввода расходов
15. Таблица расходов
16. Таблица доходов
17. Таблица продукции
18. Категория расходов
19. Цена продукции
20. Строка в таблице расходов
21. Строка в таблице доходов
22. Экран ввода/удаления в прайс-лист
23. Список продукции (прайс-лист)

**Список классов:**

1. Экран пользовательского интерфейса (UserInterface)
2. Продукция (Produce)
3. Список продукции (ProduceList)
4. Экран ввода/удаления в прайс-лист (ProduceInputScreen)
5. Экран ввода расходов (ExpenseInputScreen)
6. Таблица расходов (ExpenseList)
7. Расход (Expense)
8. Экран ввода доходов (ProfitInputScreen)
9. Таблица доходов (ProfitList)
10. Поле таблицы доходов (Profit)
11. Экран вывода годового отчёта (AnnualReport)

**Список атрибутов:**

Экран пользовательского интерфейса:

* Элементы управления (кнопки, поля ввода и т.д.)
* События пользовательского взаимодействия (нажатия кнопок, ввод данных и т.д.)

Продукция:

* Название продукта
* Идентификатор продукта
* Цена продукта

Экран ввода/удаления в прайс-лист:

* Поля для ввода новой продукции (название, цена, вес и т.д.)
* Список текущих продуктов с возможностью удаления

Список продукции (прайс-лист):

* Список всех продуктов в наличии (экземпляров класса продукция)
* Информация о каждом продукте (название, цена, вес и т.д.)

Экран ввода расходов:

* Поля для ввода расходов (название, сумма, дата и т.д.)

Таблица расходов:

* Список всех расходов
* Информация о каждом расходе (название, сумма, дата и т.д.)

Расход:

* Название расхода
* Сумма расхода
* Дата расхода
* Категория расхода (сырье, электроэнергия и т.д.)

Экран ввода доходов:

* Поля для ввода доходов (название, сумма, дата и т.д.)

Таблица доходов:

* Список всех доходов
* Информация о каждом доходе (название, сумма, дата и т.д.)

Доход: //наши доходы это только продажа выпечки

* Название дохода (какой товар продали)
* Количество проданных товаров
* Полная стоимость покупки
* Дата дохода

Годовой отчет:

* Сводная информация по году (общий доход, общие расходы, прибыль и т.д.)
* Анализ прибыльности продукции и эффективности бизнеса

**Сообщения, которыми будут обмениваться классы:**

Начать программу:

* Сообщение: start\_program
* Роль: Запуск программы, инициализация интерфейса и предоставление пользователю меню выбора действий.

Добавить/удалить продукцию из прайс-листа:

* Сообщения:
  + add\_product\_to\_price\_list(product\_info)
  + remove\_product\_from\_price\_list(product\_name)
* Роль: Добавление новой продукции в прайс-лист или удаление существующей.

Ввести значения в таблицу расходов:

* Сообщение: add\_expense(category, expense\_item, month, amount)
* Роль: Добавление информации о расходах в соответствующую таблицу.

Вывести таблицу расходов:

* Сообщение: display\_expense\_table
* Роль: Запрос на отображение таблицы расходов.

Ввести значения в таблицу доходов:

* Сообщение: add\_income(product\_name, quantity, month, amount)
* Роль: Добавление информации о доходах в соответствующую таблицу.

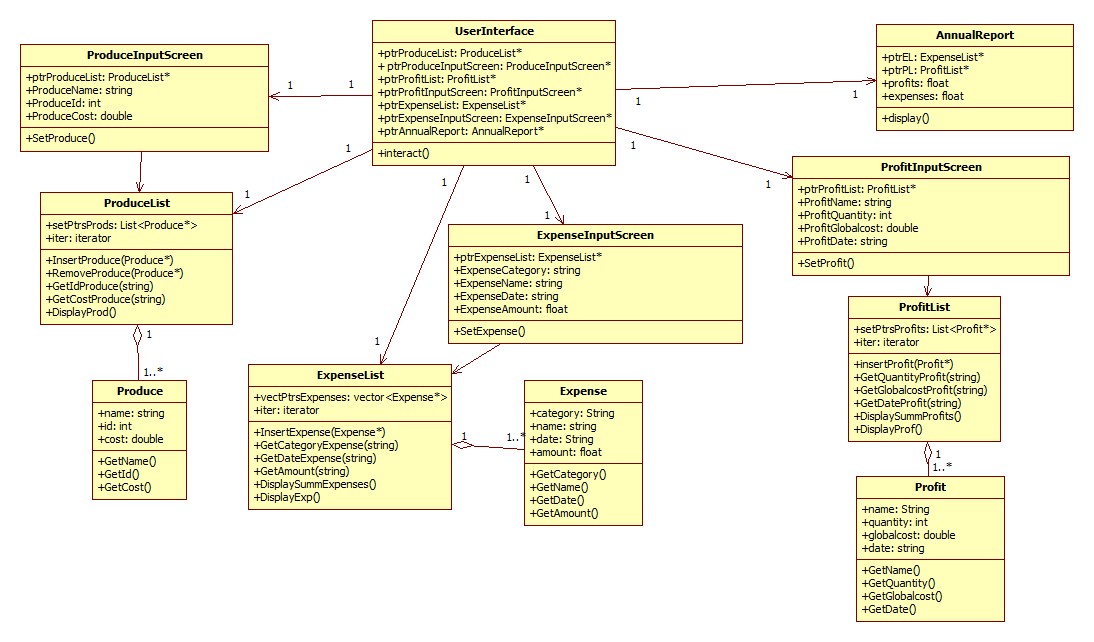
Вывести таблицу доходов:

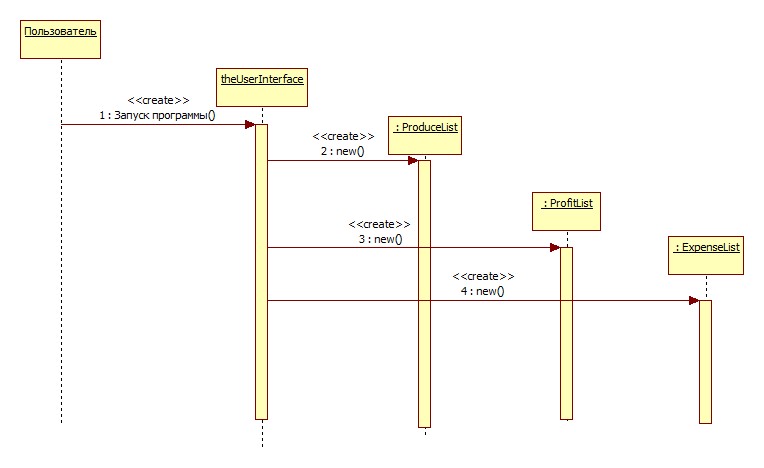
* Сообщение: display\_income\_table
* Роль: Запрос на отображение таблицы доходов.

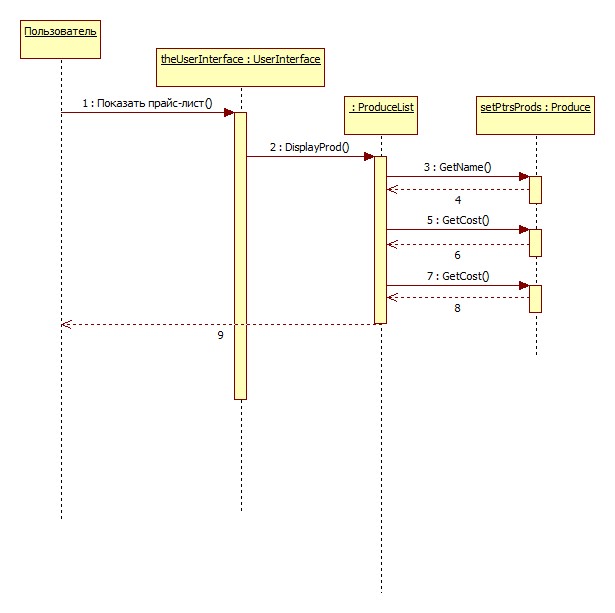
Вывести годовой отчет:

* Сообщение: display\_annual\_report
* Роль: Запрос на отображение годового отчета.

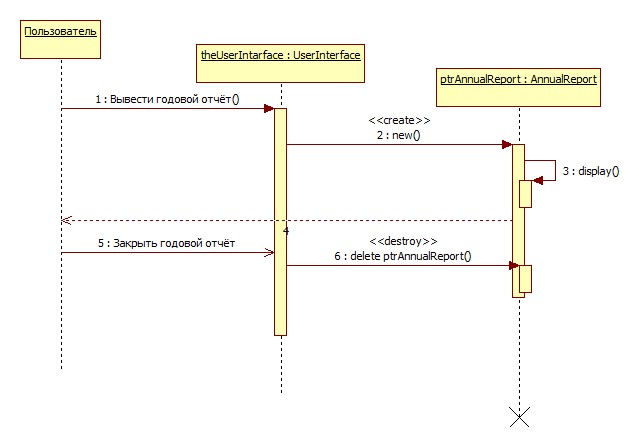
**Диаграмма классов:**

**Диаграммы последовательностей**

1. запустить программу
2. вывести прайс-лист на экран

****

1. вывести годовой отчёт

****

**Заголовочные файлы**

Полный код заголовочных файлов представлен в Приложении 1.

**Файлы реализации**

Полный код файлов реализации представлен в Приложении 2.

**Руководство пользователя**

**ГЛАВНОЕ МЕНЮ:**

При запуске программы пользователь видит **ГЛАВНОЕ МЕНЮ**, экран которого говорит ему о возможных способах взаимодействия:

**To enter data press 'e', - “ВВОД ДАННЫХ”**

**to display the report 'd', -”ВЫВОД ОТЧЕТА”**

**to exit 'q': -”ВЫХОД ИЗ ПРОГРАММЫ”**

**ВВОД ДАННЫХ:**

При вводе пользователем 'e' в главном меню у него появляется поле для ввода новых данных:

**press 'p' to add products, - “ДОБАВИТЬ ТОВАР В ПРАЙС-ЛИСТ”**

**press 'i' to record income, - “ДОБАВИТЬ ЗАПИСЬ О ДОХОДЕ”**

**press 'e' to record expenses: -”ДОБАВИТЬ ЗАПИСЬ О РАСХОДЕ”**

**ДОБАВИТЬ ТОВАР В ПРАЙС-ЛИСТ:**

При вводе пользователем 'p' у него появляется поле для ввода новых товаров в прайс-лист, в котором ему предлагают сначала ввести название товара, затем id, затем цену. При последовательном вводе всех данных экран возвращается к **ГЛАВНОМУ МЕНЮ**.

**ДОБАВИТЬ ЗАПИСЬ О ДОХОДЕ:**

При вводе пользователем 'i' у него появляется поле для ввода новых доходов, в котором ему предлагают сначала ввести название дохода (название проданного товара), затем количество проданных товаров с таким названием, затем их цену и затем дату продажи. При последовательном вводе всех данных экран возвращается к **ГЛАВНОМУ МЕНЮ**.

**ДОБАВИТЬ ЗАПИСЬ О РАСХОДЕ:**

При вводе пользователем 'e' у него появляется поле для ввода новых расходов, в котором ему предлагают сначала ввести категорию расхода, затем название, затем дату, затем сумму потраченных средств. При последовательном вводе всех данных экран возвращается к **ГЛАВНОМУ МЕНЮ**.

**ВЫВОД ОТЧЕТА:**

При вводе пользователем 'd' в **ГЛАВНОМ МЕНЮ**, программа предлагает ему выбрать, что выводить на экран:

**Press 'l' to display the list of products, -”ВЫВОД ПРАЙС-ЛИСТА”**

**press 'i' to display income, -”ВЫВОД ВСЕХ ЗАПИСЕЙ О ДОХОДАХ”**

**press 'e' to display expenses, -”ВЫВОД ВСЕХ ЗАПИСЕЙ О РАСХОДАХ”**

**press 'a' to display the annual report: -”ВЫВОД ГОДОВОГО ОТЧЕТА”**

**ВЫВОД ПРАЙС-ЛИСТА:**

При вводе пользователем 'l' на экран в виде таблицы со столбцами ID, Name и Cost выводятся построчно все товары в прайс-листе.

Если он пуст, то пользователь видит надпись \*\*\*There's nothing here\*\*\*. После вывода данных пользователь возвращается к экрану главного меню.

**ВЫВОД ВСЕХ ЗАПИСЕЙ О ДОХОДАХ:**

При вводе пользователем 'i' на экран в виде таблицы со столбцами Name, Quantity , Globalcost, Date выводятся построчно все записи о доходах.

Если он пуст, то пользователь видит надпись \*\*\*There's nothing here\*\*\*. После вывода данных пользователь возвращается к экрану главного меню.

**ВЫВОД ВСЕХ ЗАПИСЕЙ О РАСХОДАХ**

При вводе пользователем 'e' на экран в виде таблицы со столбцами Category, Name, Date, Amount выводятся построчно все записи о расходах.

Если он пуст, то пользователь видит надпись **\*\*\*There's nothing here\*\*\***. После вывода данных пользователь возвращается к экрану главного меню.

**ВЫВОД ГОДОВОГО ОТЧЕТА**

При вводе пользователем 'a' на экран выводится в виде строк сумма всех продаж, расходов и чистая прибыль (разница между доходами и расходами):

**Annual report**

**----------------------------**

**Incomes**

**Bakery sells: 3434**

**Expenses**

**Total expense: 543**

**Profit: 2891**

После этого выводится экран **главного меню**.

**ВЫХОД ИЗ ПРОГРАММЫ:**

При вводе пользователем 'q' в главном меню работа программы прекращается.

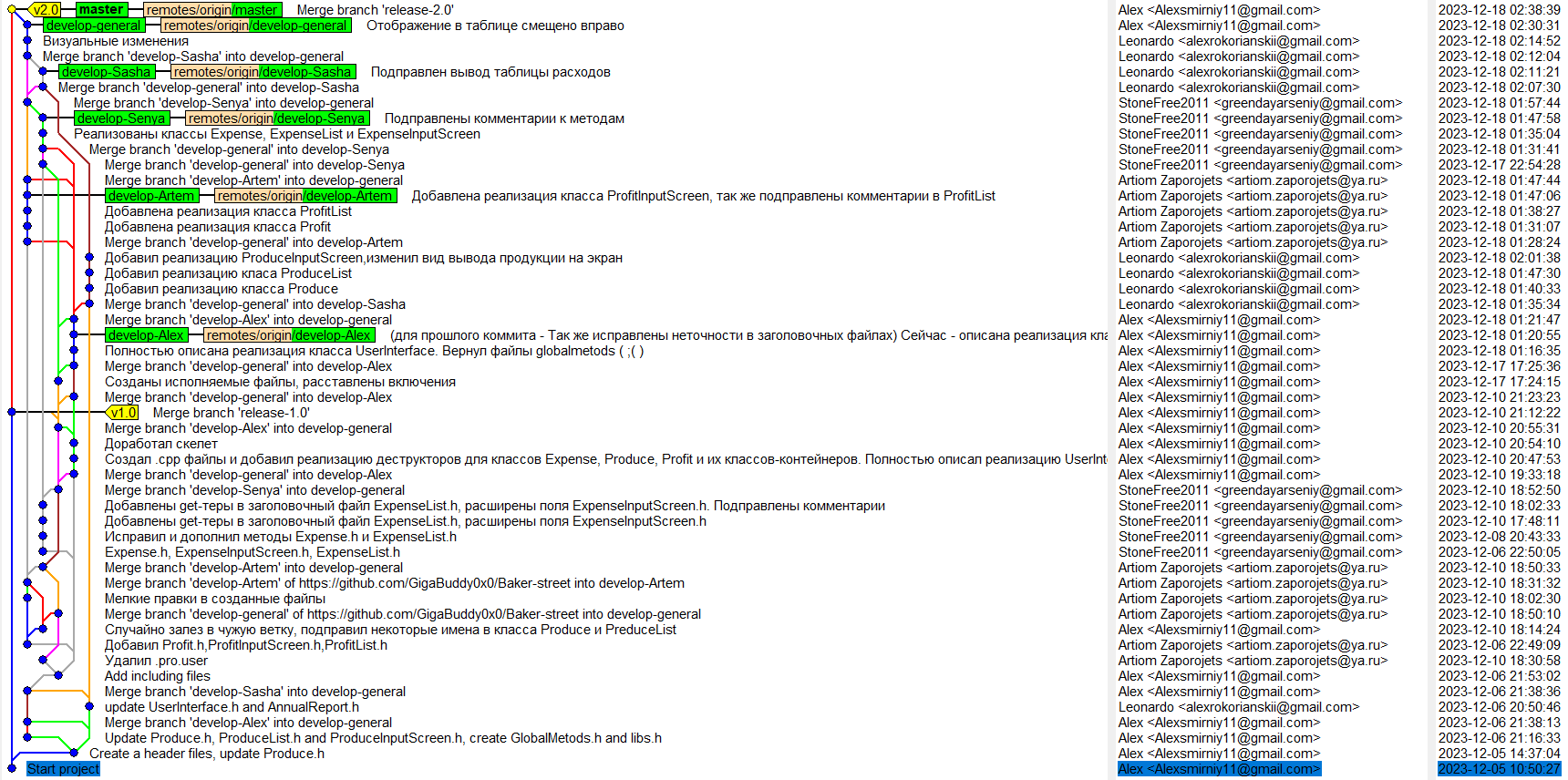
**История проекта на GitHub**

Для контроля версий проекта был создан публичный репозиторий Baker-street[[1]](#footnote-1). Разработка осуществляется с использованием двух ветвей – master и develop. На первой помещаются стабильные версии с тэгом, в котором указываются реализованные функции. Во второй могут находиться менее стабильные, но рабочие версии. От ветви develop создаются ветви функциональностей и ветви релизов. Ветви функциональностей используются для создания и отладки новых функций. Последний коммит ветви должен быть достаточно стабильным, чтобы ветвь можно было слить с develop. Ветви релизов используются для подготовки к выпуску новых версий. На ней можно добавлять минорные исправления перед слиянием с master.

В разработке участвуют все члены команды. Для удобства автор коммита будет подписан инициалами. Сокращения представлены ниже:

* Смирнов Алексей Германович – САГ
* Запорожец Артём Алексеевич – ЗАА
* Шаркевич Арсений – ШАИ
* Рокотянский Александр Леонидович – РАЛ

История (от создания проекта до его финального коммита):



Разберем коммиты подробнее:

1. **Start project** (САГ) – создание репозитория
2. **Create a header files, update Produce.h** (САГ) – Созданы заголовочные файлы для всех классов. Создан прототип класса Produce.
3. **Update Produce.h, ProduceList.h and ProduceInputScreen.h, create GlobalMetods.h and libs.h** (САГ) – обновлены файлы Produce.h, ProduceList.h и ProduceInputScreen.h. Созданы заголовочный файл для GlobalMetods, а так же файл libs.h.
4. **Merge branch 'develop-Alex' into develop-general** (САГ) – Залил все наработки с 'develop-Alex' ветки в основную ветку разработки.
5. **Update UserInterface.h and AnnualReport.h** (РАЛ) – Обновлены файлы UserInterface.h и AnnualReport.h
6. **Merge branch 'develop-Sasha' into develop-general** (САГ) – Залил все наработки с 'develop-Sasha' ветки в основную ветку разработки.
7. **Add including files** (САГ) – Добавлен файл, в котором подключены все библиотеки. Этот файл для упрощения подключается во всех остальных классах.
8. **Удалил .pro.user** (ЗАА) – Удалил Baker-street.pro.user, тк вызвал конфликт.
9. **Добавил Profit.h, ProfitInputScreen.h, ProfitList.h** (ЗАА) – В заголовочные файлы были добавлены прототипы класса Profit, ProfitInputScreen и ProfitList.
10. **Случайно залез в чужую ветку, подправил некоторые имена в класса**

**Produce и PreduceList** (САГ) – Ошибочно изменён код на чужой ветке разработки. Были внесены правки в методы классов Produce и PreduceList

1. **Merge branch 'develop-general' ofhttps://github.com/GigaBuddy0x0/Baker-street into develop-general** (ЗАА) – добавил данные с удалённого репозитория в свой локальный
2. **Мелкие правки в созданные файлы** (ЗАА) –Проведены мелкие изменения в уже созданных файлах
3. **Merge branch 'develop-Artem' of https://github.com/GigaBuddy0x0/Baker-street into develop-Artem** (ЗАА) –Залил данные с ветки 'develop-Artem' из удалённого репозитория в ветку 'develop-Artem' своего локального
4. **Merge branch 'develop-Artem' into develop-general** (ЗАА) – Залил все наработки с ветки 'develop-Artem'в основную ветку разработки.
5. **Expense.h, ExpenseInputScreen.h, ExpenseList.h** (ШАИ) – В заголовочные файлы были добавлены прототипы класса Expense, ExpenseInputScreen и ExpenseList.
6. **Исправил и дополнил методы Expense.h и** **ExpenseList.h** (ШАИ) – Были внесены изменения в прототипы классов Expense и ExpenseList.
7. **Добавлены get-теры в заголовочный файл ExpenseList.h, расширены поля ExpenseInputScreen.h** (ШАИ) – Были добавлены методы Get в прототип класса Expense, а так же добавлены новые поля в прототип метода ExpenseList.
8. **Добавлены get-теры в заголовочный файл ExpenseList.h, расширены поля ExpenseInputScreen.h. Подправлены комментарии** (ШАИ) – Предыдущий коммит, только в этом были изменены комментарии к строкам кода.
9. **Merge branch 'develop-Senya' into develop-general** (ШАИ) – Залил все наработки со своей ветки в основную ветку разработки.
10. **Merge branch 'develop-general' into develop-Alex** (САГ) – Актуальный прототип программы был залит на личною ветку Алексея Германовича.
11. **Создал .cpp файлы и добавил реализацию деструкторов для классов Expense, Produce, Profit и их классов-контейнеров. Полностью описал реализацию UserInterface и создал работающий скелет будущей программы** (САГ) – Были добавлены исполняемые файлы для всех существующих классов, реализованы деструкторы для них. Создан рабочий прототип программы.
12. **Доработан скелет** (САГ) – Внесены правки в существующий прототип программы.
13. **Merge branch 'develop-Alex' into develop-general** (САГ) – Залил все наработки с ветки 'develop-Alex' в основную ветку разработки.
14. **Merge branch 'release-1.0'** (САГ) – Первый релиз программы.
15. **Merge branch 'develop-general' into develop-Alex** (САГ) – Залил все наработки с 'develop-general' ветки в свою ветку разработки.
16. **Созданы исполняемые файлы, расставлены включения** (САГ) – Были созданы исполняемые файлы, так же в них были расставлены включения заголовочных файлов.
17. **Merge branch 'develop-general' into develop-Alex** (САГ) – Залил все наработки с 'develop-general' ветки в свою ветку разработки.
18. **Полностью описана реализация класса UserInterface. Вернул файлы globalmetods ( ;( )** (САГ) – Была описана реализация класса UserInterface. Обратно добавлен файл globalmetods.
19. **(для прошлого коммита - Так же исправлены неточности в заголовочных файлах) Сейчас - описана реализация класса AnnualReport.** (САГ) – Были исправлены неточности в заголовочных файлах. Так же была написана реализация класса AnnualReport.
20. **Merge branch 'develop-Alex' into develop-general** (САГ) – Залил все наработки с 'develop-Alex' ветки в основную ветку разработки.
21. **Merge branch 'develop-general' into develop-Sasha** (РАЛ) – Залил все наработки с 'develop-general' ветки в свою ветку разработки.
22. **Добавил реализацию класса Produce** (РАЛ) – Была написана реализация класса Produce
23. **Добавил реализацию класа ProduceList** (РАЛ) – Была написана реализация класса ProduceList
24. **Добавил реализацию ProduceInputScreen,изменил вид вывода продукции на экран** (РАЛ) – Была написана реализация класса ProduceInputScreen, так же внесены изменения в вывод продукции на экран.
25. **Merge branch 'develop-general' into develop-Artem** (ЗАА) – Залил все наработки с 'develop-general' ветки в свою ветку разработки.
26. **Добавлена реализация класса Profit** (ЗАА) – Была написана реализация класса Profit
27. **Добавлена реализация класса ProfitList** (ЗАА) – Была написана реализация класса ProfitList
28. **Добавлена реализация класса ProfitInputScreen, так же подправлены комментарии в ProfitList** (ЗАА) – Была написана реализация класса ProfitInputScreen, так же исправил комментарии в классе ProfitList
29. **Merge branch 'develop-Artem' into develop-general** (ЗАА) – Залил все наработки с ветки 'develop-Artem'в основную ветку разработки.
30. **Merge branch 'develop-general' into develop-Senya** (ШАИ) – Залил все наработки с 'develop-general' ветки в свою ветку разработки.
31. **Merge branch 'develop-general' into develop-Senya** (ШАИ) – Залил все наработки с 'develop-general' ветки в свою ветку разработки.
32. **Реализованы классы Expense, ExpenseList и ExpenseInputScreen** (ШАИ) – Была написана реализация классов Expense, ExpenseList и ExpenseInputScreen.
33. **Подправлены комментарии к методам** (ШАИ) – Были исправлены неточности в комментариях к методам.
34. **Merge branch 'develop-Senya' into develop-general** (ШАИ) – Залил все наработки с ветки 'develop-Senya'в основную ветку разработки.
35. **Merge branch 'develop-general' into develop-Sasha** (РАЛ) – Залил все наработки с 'develop-general' ветки в свою ветку разработки.
36. **Подправлен вывод таблицы расходов** (РАЛ) – Исправлен вывод данных в таблице расходов.
37. **Merge branch 'develop-Sasha' into develop-general** (РАЛ) – Залил все наработки с ветки 'develop-Sasha'в основную ветку разработки.
38. **Визуальные изменения** (РАЛ) – Были внесены небольшие визуальные изменения.
39. **Отображение в таблице смещено вправо** (САГ) – Отображение информации в таблицах было смещено вправо.
40. **Merge branch 'release-2.0'** (САГ) – Второй релиз программы.

**Заключение**

В процессе работы над проектом командой разработчиков было создано консольное приложение с использованием популярной среды разработки qt-creator и высокоуровневого языка программирования C++. Все изменения в коде контролировались с помощью системы контроля версий Git, что позволяло отслеживать и фиксировать важные моменты в истории разработки. В ходе этого процесса могли возникать сбои, но такие проблемные версии не были включены в основную историю Git. Вместо этого они локально исправлялись и только после этого коммитились.

На данный момент при тестировании программы не было обнаружено ошибок или сбоев в ее работе. Принципы раздельной компиляции были аккуратно применены, разделяя классы на заголовочные файлы, содержащие интерфейсные части, и файлы реализации, содержащие тела классов. Нами также был реализован механизм очистки памяти, а именно явно определены

деструкторы в определенных классах для освобождения списков, созданных в процессе работы программы.

Весь код был проведен через проверку неиспользуемых переменных и было убеждено в их отсутствии. Мы также подтвердили, что все алгоритмы используются без избыточности - все циклы, массивы и другие конструкции, применяемые в программе, являются необходимыми для её функционирования.

В качестве дополнительных материалов, мы прикрепили диаграмму вариантов использования, диаграмму классов и диаграммы последовательностей для наглядного представления структуры и работы программы. В итоге проделанной работы была успешно достигнута поставленная цель.

Примечание 1.

Ссылка на репозиторий проекта:  
<https://github.com/GigaBuddy0x0/Baker-street>

# **Приложение 1**

1. AnnualReport.h

#ifndef ANNUALREPORT

#define ANNUALREPORT

//////////////////класс AnnualReport///////////////////////

//Класс годового отчета

#include "ProfitList.h"

#include "ExpenseList.h"

class AnnualReport

{

private:

ProfitList\* ptrPL; // записи доходов

ExpenseList\* ptrEL; // записи расходов

float expenses, profits; // суммы доходов и расходов

public:

AnnualReport(ProfitList\* pPL, ExpenseList\* pEL): ptrPL(pPL), ptrEL(pEL)

{ }

void display(); // отображение годового отчета

};

#endif // ANNUALREPORT

1. Expense.h

#ifndef EXPENSE

#define EXPENSE

#include "libs.h"

//Класс затрат

class Expense

{

private:

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

string name; //название расхода

string date; // дата уплаты расходов

float amount; // сколько платим

public:

Expense(string c, string n, string d, float a):

category(c), name(n), date(d), amount(a)

{ }

~Expense();

//геттеры:

string GetCategory();

string GetName();

string GetDate();

float GetAmount();

};

#endif // EXPENSE

1. ExpenseInputScreen.h

#ifndef EXPENSEINPUTSCREEN

#define EXPENSEINPUTSCREEN

#include "ExpenseList.h"

//Экран для ввода расходов

class ExpenseInputScreen

{

private:

ExpenseList\* ptrExpenseList; // запись о расходах

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

string ExpenseName; //название расхода

string ExpenseDate; // дата уплаты расходов

float ExpenseAmount; // сколько платим

public:

ExpenseInputScreen(ExpenseList\* ptrExp): ptrExpenseList(ptrExp)

{ }

void SetExpense(); //внести расход

};

#endif // EXPENSEINPUTSCREEN

1. ExpenseList

#ifndef EXPENSELIST

#define EXPENSELIST

#include "Expense.h"

//Класс записей о расходах

class ExpenseList

{

private:

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

vector<Expense\*>::iterator iter;

public:

~ExpenseList();

void InsertExpense(Expense\*); //внести расход

float DisplaySummExpenses(); //вывести сумму всех расходов

void DisplayExp(); //вывести строку расходов

//геттеры:

//string GetCategoryExpense(string);

//string GetDateExpense(string);

//float GetAmountExpense(string);

};

#endif // EXPENSELIST

1. GlobalMetods.h

#ifndef GLOBALMETODS

#define GLOBALMETODS

#include "libs.h"

void getaLine(string& inStr); // получение строки текста

char getaChar(); // получение символа

#endif // GLOBALMETODS

1. libs.h

#ifndef LIBS

#define LIBS

#include <iostream>

#include <vector>

#include <list>

#include <string>

#include <numeric>

#include <iomanip>

using namespace std;

#include "globalmetods.h"

#endif // LIBS

1. Produce.h

#ifndef PRODUCE

#define PRODUCE

#include "libs.h"

class Produce

{

private:

string name;

int id;

double cost;

public:

Produce(string, int, double);

~Produce();

string GetName();

int GetId();

double GetCost();

};

#endif // PRODUCE

1. ProduceInputScreen.h

#ifndef PRODUCEINPUTSCREEN

#define PRODUCEINPUTSCREEN

#include "ProduceList.h"

class ProduceInputScreen

{

private:

ProduceList\* ptrProduceList;

string ProduceName;

int ProduceId;

double ProduceCost;

public:

ProduceInputScreen(ProduceList\* ptrPL) : ptrProduceList(ptrPL)

{ }

void setProduce(); // добавить данные о продукции

};

#endif // PRODUCEINPUTSCREEN

1. ProduceList.h

#ifndef PRODUCELIST

#define PRODUCELIST

#include "Produce.h"

class ProduceList {

private:

// установить указатели на продукцию

list <Produce\*> setPtrsProds; // указатели на класс продукция

list <Produce\*>::iterator iter; //итератор

public:

~ProduceList(); // деструктор (удаление продукции из списка)

void InsertProduce(Produce\*); // добавить продукцию в список

void RemoveProduce(Produce\*); // добавить продукцию в список

double GetCostProduce(string); // возвращают id и стоимость записи по имени позиции

void DisplayProd(); // вывод списка продукции

};

#endif // PRODUCELIST

1. Profit.h

#ifndef PROFIT

#define PROFIT

#include "libs.h"

class Profit

{

private:

string name;

int quantity;

double globalcost;

string date;

public:

Profit(string, int, double, string);

~Profit();

string GetName();

int GetQuantity();

double GetGlobalcost();

string GetDate();

};

#endif // PROFIT

1. ProfitInputScreen.h

#ifndef PROFITINPUTSCREEN

#define PROFITINPUTSCREEN

#include "ProfitList.h"

class ProfitInputScreen

{

private:

ProfitList\* ptrProfitList;

string ProfitName;

int ProfitQuantity;

double ProfitGlobalCost;

string ProfitDate;

public:

ProfitInputScreen(ProfitList\* ptrPrfL) : ptrProfitList(ptrPrfL)

{

/\* тут пусто \*/

}

void setProfit(); // добавить данные о доходе

};

#endif // PROFITINPUTSCREEN

1. ProfitList.h

#ifndef PROFITLIST

#define PROFITLIST

#include "Profit.h"

class ProfitList {

private:

// установить указатели на доход

list <Profit\*> setPtrsProfits; // указатели на класс дохода

list <Profit\*>::iterator iter; //итератор

public:

~ProfitList(); // деструктор (удаление дохода из списка)

void InsertProfit(Profit\*); // добавить доход в список

//int GetQuantityProfit(string);

//double GetGlobalcostProfit(string);

//string GetDateProfit(string);

float DisplaySummProfits(); // вывод списка доходов

void DisplayProf();

};

#endif // PROFITLIST

1. UserInterface.h

#ifndef USERINTERFACE

#define USERINTERFACE

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

//Главный класс для запуска приложения:

//этот класс определяет взаимодействие юзера с программой.

#include "ProduceInputScreen.h"

#include "ProfitInputScreen.h"

#include "ExpenseInputScreen.h"

#include "AnnualReport.h"

class UserInterface

{

private:

ProduceList\* ptrProduceList;

ProduceInputScreen\* ptrProduceInputScreen;

ProfitList\* ptrProfitList;

ProfitInputScreen\* ptrProfitInputScreen;

ExpenseList\* ptrExpenseList;

ExpenseInputScreen\* ptrExpenseInputScreen;

AnnualReport\* ptrAnnualReport;

public:

UserInterface();

~UserInterface();

void interact();

};

#endif // USERINTERFACE

# **Приложение 2**

Baker-street/AnnualReport.cpp

#include "AnnualReport.h"

void AnnualReport::display()

{

cout << setw(20) << "Annual report" << endl;

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

cout << "Incomes\n" << endl;

cout << "Bakery sells:\t\t";

profits = ptrPL->DisplaySummProfits();

cout << profits << endl << endl;

cout << "Expenses\n\n";

expenses = ptrEL->DisplaySummExpenses();

cout << "Total expense:\t\t";

cout << expenses << endl << endl;

// вычисляем прибыльность

cout << "\nProfit:\t\t\t" << (profits - expenses) << endl;

}

Baker-street/Expense.cpp  
#include "Expense.h"

Expense::~Expense() // деструктор

{ }

string Expense::GetCategory()

{

return category;

}

string Expense::GetName()

{

return name;

}

string Expense::GetDate()

{

return date;

}

float Expense::GetAmount()

{

return amount;

}

Baker-street/ExpenseInputScreen.cpp  
  
#include "ExpenseInputScreen.h"

void ExpenseInputScreen::SetExpense() // добавить данные о расходе

{

cout << "Input the category of expense: " << endl;

getaLine(ExpenseCategory);

cout << "Input the name of expense: " << endl;

getaLine(ExpenseName);

cout << "Input the date of expense: " << endl;

getaLine(ExpenseDate);

cout << "Input the amount of expense: " << endl;

cin >> ExpenseAmount;

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

Expense\* ptrExpense = new Expense (ExpenseCategory, ExpenseName, ExpenseDate, ExpenseAmount); // создать расход

ptrExpenseList->InsertExpense(ptrExpense); // занести в список расходов

}

Baker-street/ExpenseList.cpp  
  
#include "ExpenseList.h"

ExpenseList::~ExpenseList() // деструктор

{ // удалить объекты expense

// удалить указатели на вектор

while (!vectPtrsExpenses.empty())

{

iter = vectPtrsExpenses.begin();

delete \*iter;

vectPtrsExpenses.erase(iter);

}

}

void ExpenseList::InsertExpense(Expense\* ptrExp)

{

vectPtrsExpenses.push\_back(ptrExp);

}

float ExpenseList::DisplaySummExpenses() // сумма всех платежей

{

float summExpenses = 0.0;

iter = vectPtrsExpenses.begin();

while (iter != vectPtrsExpenses.end())

{ // плюсуем суммы всех платежей жильцов за все время

summExpenses += (\*iter)->GetAmount();

iter++;

}

return summExpenses;

}

void ExpenseList::DisplayExp() // отобразить все строки расходов

{

cout << endl;

cout << left << setw(15) << "Category" << setw(15) << "Name"

<< setw(15) << "Date" << setw(15) << "Amount";

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

if (vectPtrsExpenses.empty()) // если список расходов пуст

cout << "\*\*\*There's nothing here\*\*\*\n" << endl; // выводим запись, что он пуст)

else

{

iter = vectPtrsExpenses.begin();

while (iter != vectPtrsExpenses.end()) // выводим все расходы

{

cout << left << setw(15) << (\*iter)->GetCategory()

<< setw(15) << (\*iter)->GetName()

<< setw(15) << (\*iter)->GetDate()

<< setw(15) << (\*iter)->GetAmount() << endl;

\*iter++;

}

}

}

Baker-street/Produce.cpp  
  
#include "Produce.h"

Produce::Produce(string s, int i, double d): name(s), id(i), cost(d)

{ }

Produce::~Produce() // деструктор

{ }

string Produce::GetName()

{

return name;

}

int Produce::GetId()

{

return id;

}

double Produce::GetCost()

{

return cost;

}

Baker-street/ProduceInputScreen.cpp  
  
#include "ProduceInputScreen.h"

void ProduceInputScreen::setProduce() // добавить данные о товаре

{

cout << "Input the name of produce: " << endl;

getaLine(ProduceName);

cout << "Input the id of produce: " << endl;

cin >> ProduceId;

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

cout << "Input the cost of produce: " << endl;

cin >> ProduceCost;

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

Produce\* ptrProduce = new Produce(ProduceName, ProduceId, ProduceCost); // занести товар в таблицу

ptrProduceList->InsertProduce(ptrProduce); // занести в таблицу товаров

}  
  
Baker-street/ProduceList.cpp  
  
#include "ProduceList.h"

ProduceList::~ProduceList() // деструктор

{ // удалить строки с продукцией,

// удалить указатели из списка.

while (!setPtrsProds.empty())

{

iter = setPtrsProds.begin();

delete \*iter;

setPtrsProds.erase(iter);

}

}

void ProduceList::InsertProduce(Produce\* ptrPr) // добавить продукцию в список

{

setPtrsProds.push\_back(ptrPr); // вставка новой продукции в список

}

//void ProduceList::DeletProduce(Produce\* ptrPr) // удалить продукцию из списка

double ProduceList::GetCostProduce(string name) // получить стоимость товара по его имени

{

int cost;

iter = setPtrsProds.begin();

while (iter != setPtrsProds.end())

{

cost = (\*iter)->GetCost();

if (name == ((\*iter)->GetName())) // сравниваем по именам и

{ // если получившаяся пара совпадает - значит,

//мы нашли запись об этом товаре в списке, в этом случае

return cost; // возвращаем его стоимость

}

iter++;

}

return -1; // если нет - возвращаем -1

}

void ProduceList::DisplayProd() // вывод списка товаров

{

cout << endl;

cout << left << setw(15) << "Id" << setw(15) << "Name" << setw(15) << "Cost";

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

if (setPtrsProds.empty()) // если список товаров пуст

cout << "\*\*\*There's nothing here\*\*\*\n" << endl; // выводим запись, что он пуст)

else

{

iter = setPtrsProds.begin();

while (iter != setPtrsProds.end()) // распечатываем все товары

{

cout << left << setw(15) << (\*iter)->GetId()

<< setw(15) << (\*iter)->GetName()

<< setw(15) << (\*iter)->GetCost() << endl;

\*iter++;

}

}

}

Baker-street/Profit.cpp  
  
#include "Profit.h"

Profit::~Profit() // деструктор

{ }

Profit::Profit(string n, int q, double g, string d): name(n), quantity(q), globalcost(g), date(d)

{ }

string Profit::GetName()

{

return name;

}

int Profit::GetQuantity()

{

return quantity;

}

double Profit::GetGlobalcost()

{

return globalcost;

}

string Profit::GetDate()

{

return date;

}

Baker-street/ProfitInputScreen.cpp  
  
#include "ProfitInputScreen.h"

void ProfitInputScreen::setProfit() // добавить данные о доходе

{

cout << "Input the name of profit: " << endl;

getaLine(ProfitName);

cout << "Input the quantity of profit: " << endl;

cin >> ProfitQuantity;

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

cout << "Input the cost of profit: " << endl;

cin >> ProfitGlobalCost;

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

cout << "Input the date of profit: " << endl;

cin >> ProfitDate;

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

Profit\* ptrProfit = new Profit(ProfitName, ProfitQuantity, ProfitGlobalCost, ProfitDate); // создать доход

ptrProfitList->InsertProfit(ptrProfit); // занести в список доходов

}

Baker-street/ProfitList.cpp  
  
#include "ProfitList.h"

ProfitList::~ProfitList() // деструктор

{ // удалить строки с доходами,

// удалить указатели из списка.

while (!setPtrsProfits.empty())

{

iter = setPtrsProfits.begin();

delete \*iter;

setPtrsProfits.erase(iter);

}

}

void ProfitList::InsertProfit(Profit\* ptrProf)

{

setPtrsProfits.push\_back(ptrProf);

}

void ProfitList::DisplayProf() // отобразить все строки с доходами

{

cout << endl;

cout << left << setw(15) << "Name" << setw(15) << "Quantity"

<< setw(15) << "Globalcost" << setw(15) << "Date";

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

if (setPtrsProfits.empty()) // если список доходов пуст

cout << "\*\*\*There's nothing here\*\*\*\n" << endl; // выводим запись, что он пуст)

else

{

iter = setPtrsProfits.begin();

while (iter != setPtrsProfits.end()) // распечатываем все доходы

{

cout << left << setw(15) << (\*iter)->GetName()

<< setw(15) << (\*iter)->GetQuantity()

<< setw(15) << (\*iter)->GetGlobalcost()

<< setw(15) << (\*iter)->GetDate() << endl;

\*iter++;

}

}

}

float ProfitList::DisplaySummProfits() // сумма всех доходов

{

float sumProfits = 0.0;

iter = setPtrsProfits.begin();

while (iter != setPtrsProfits.end())

{ // плюсуем суммы всех доходов за все время

sumProfits += (\*iter)->GetGlobalcost();

iter++;

}

return sumProfits;

}

Baker-street/UserInterface.cpp  
  
#include "UserInterface.h"

UserInterface::UserInterface()

{

ptrProduceList = new ProduceList;

ptrProfitList = new ProfitList;

ptrExpenseList = new ExpenseList;

}

UserInterface::~UserInterface()

{

delete ptrProduceList;

delete ptrProfitList;

delete ptrExpenseList;

}

void UserInterface::interact()

{

char ch;

while (true)

{

cout << "To enter data press 'e', \n"

<< " to display the report 'd', \n"

<< " to exit 'q': \n";

ch = getaChar();

if (ch == 'e') // enter data

{

cout << " press 'p' to add products, \n"

<< " press 'i' to record income, \n"

<< " press 'e' to record expenses: \n";

ch = getaChar();

switch (ch)

{

// input screens only exist during their use

case 'p':

ptrProduceInputScreen = new ProduceInputScreen(ptrProduceList);

ptrProduceInputScreen->setProduce();

delete ptrProduceInputScreen;

break;

case 'i':

ptrProfitInputScreen = new ProfitInputScreen(ptrProfitList);

ptrProfitInputScreen->setProfit();

delete ptrProfitInputScreen;

break;

case 'e':

ptrExpenseInputScreen = new ExpenseInputScreen(ptrExpenseList);

ptrExpenseInputScreen->SetExpense();

delete ptrExpenseInputScreen;

break;

default: cout << "Unknown function\n";

break;

} // end of switch section

} // end of if condition

else if (ch == 'd') // output data

{

cout << "Press 'l' to display the list of products, \n"

<< " press 'i' to display income, \n"

<< " press 'e' to display expenses, \n"

<< " press 'a' to display the annual report: \n";

ch = getaChar();

switch (ch)

{

case 'l': ptrProduceList->DisplayProd();

break;

case 'i': ptrProfitList->DisplayProf();

break;

case 'e': ptrExpenseList->DisplayExp();

break;

case 'a':

ptrAnnualReport = new AnnualReport(ptrProfitList, ptrExpenseList);

ptrAnnualReport->display();

delete ptrAnnualReport;

default: cout << "Unknown display function\n";

break;

} // end switch

} // end elseif

else if (ch == 'q') {

return; // exit

}

else

cout << "Unknown function. Press only 'e', 'd', or 'q'\n";

} // end while

}

1. https://github.com/GigaBuddy0x0/Baker-street.git [↑](#footnote-ref-1)