**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**федеральное государственное бюджетное образовательное учреждение высшего профессионального образования**

**«Казанский национальный исследовательский технический университет**

**им. А.Н. Туполева-КАИ»**

Институт Компьютерных технологий и защиты информации

Кафедра Прикладной математики и информатики имени Ю.В. Кожевникова

Пояснительная записка

к курсовой работе

по дисциплине «Структуры и алгоритмы обработки данных»

на тему «Реализация комбинированных структур данных»

вариант №47.11

**Выполнил**

Студент группы 4210

Нигамадянов Фанис Магефурович

**Проверил**

Доцент кафедры ПМИ

Сотников Сергей Викторович

Казань 2022

**Содержание**

[**Глава 1. Постановка задачи** 3](#_Toc105499253)

[1.1 Цель работы 3](#_Toc105499254)

[1.2 Исходные данные 3](#_Toc105499255)

[1.3 Ожидаемый результат 3](#_Toc105499256)

[1.4 Требования к реализации 4](#_Toc105499257)

[2. ОПИСАНИЕ РАЗРАБОТАННЫХ КЛАССОВ И 5](#_Toc105499258)

[АЛГОРИТМЫ РЕАЛИЗАЦИИ ОСНОВНЫХ ОПЕРАЦИЙ 5](#_Toc105499259)

[2.1 Реализация упорядоченного статического списка 5](#_Toc105499260)

[2.2 Реализация упорядоченного динамического списка 6](#_Toc105499261)

[2.3 Краткое описание разработанных классов 10](#_Toc105499262)

[2.4 Схема структуры проекта (взаимодействия классов) 12](#_Toc105499263)

[3. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 13](#_Toc105499264)

[4. РУКОВОДСТВО ПРОГРАММИСТА 15](#_Toc105499265)

[ВЫВОД 21](#_Toc105499266)

[СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ 22](#_Toc105499267)

[ПРИЛОЖЕНИЕ. ЛИСТИНГИ ВСЕХ ФАЙЛОВ ПРОЕКТА 23](#_Toc105499268)

# **Глава 1. Постановка задачи**

Цель работы: программная реализация комбинированной структуры данных –динамический статический упорядоченный список упорядоченных динамических списков – на основе объектно-ориентированного подхода, повторение и закрепление полученных знаний в ходе изучения курсов «Структуры и алгоритмы обработки данных» и «Объектно-ориентированное программирование», выработка навыков владения языком программирования.

Требования к разрабатываемой программе:

1) Реализация всех необходимых операций (добавление и удаление в основной и присоединённой структурах, поиск в списке).

2) Возможность сохранения всей структуры во внешнем файле (текстовом или XML) с обратной загрузкой.

3) Реализация структуры для хранения и обработки данных конкретной информационной задачи.

4) Именование типов, структур и их полей, классов и их свойств и методов в соответствии с конкретной информационной задачей.

5) Наличие демо-модуля с удобным оконным пользовательским интерфейсом.

Требования к реализации:

1. Реализация комбинированной структуры на основе объектного подхода;

2. Реализация всех необходимых операций (добавление и удаление в основной и присоединенной структурах, поиск в списке);

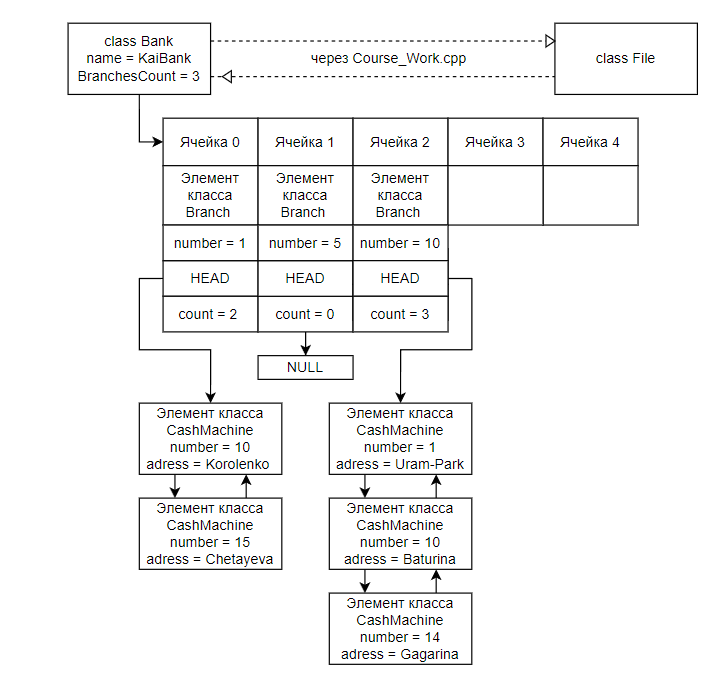
3. Возможность сохранения всей структуры по внешнем файле (текстовом или XML) с обратной загрузкой;

4. Реализация структуры для хранения и обработки данных конкретной информационной задачи;

5. Именование типов, структур и их полей, классов и их свойств, методов в соответствии с конкретной информационной задачей.

# **Глава 2. Теоретическое описание разработанной структуры**

## **2.1. Логическая составляющая структуры**



## **2.2. Набор основных операций**

Все операции комбинированной структуры делятся на 3 типа: операции над главной структурой, операции над подчинённой структурой и обход всей структуры.

Ввиду однотипности главной и подчинённой структур (обе являются списком) набор их операций совпадают:

1. Инициализация

2. Проверка возможности добавления

3. Проверка возможности удаления

4. Обход списка

5. Поиск элемента списка

6. Добавление элемента списка

7. Удаление элемента списка

## **2.3. Алгоритмизация основных операций**

Операции над главным списком:

1. Инициализация

2. Проверка возможности добавления

* Проверка наличия свободного места в главном списке

3. Проверка возможности удаления

* Проверка пустоты главного списка

4. Обход главного списка

* Задание начального значения
* В цикле, пока в списке остались необработанные элементы
* Обработка текущего элемента
* Переход к следующему элементу

5. Поиск подчинённого списка в главном

* Проверка возможности поиска
* Запрос информационной части искомого элемента
* Задание начального значения
* В цикле, пока в списке остались необработанные элементы и нужный элемент не найден
* Если информационная часть текущего элемента удовлетворяет заданному критерию, то определить текущий элемент как искомый (найденный) и выйти из цикла
* Иначе – переход к следующему элементу
* При выходе из цикла сообщить о том, что поиск был неудачен

6. Добавление подчинённого списка

* Проверка возможности добавления
* Определение положения добавляемого элемента (задание последующего и его поиск)
* Выделение памяти
* Запрос информационной части нового элемента
* Определение (инициализация полей) нового элемента
* Восстановление указателей, связанных со структурой

7. Удаление подчинённого списка

* Проверка возможности удаления (не пуст ли список)
* Задание удаляемого элемента (его информационной части)
* Поиск удаляемого элемента
* Обработка удаляемого элемента
* Исключение удаляемого элемента из структуры и восстановление указателей, связанных со структурой
* Освобождение памяти

Операции над подчинённым списком:

Подчинённый список имеет тот же набор операций, что и главный, а также имеет их схожую алгоритмизацию.

Отличие алгоритмизации заключается в том, что в начале каждой операции 1-м шагом осуществляется поиск подчинённого списка в главном, а затем уже все остальные шаги.

Обход всей структуры:

Обход всей структуры включает в себя:

* Обход главного списка
* При обработке элементов главного списка осуществлять обход соответствующих подчинённых списков

## **2.4. Реализация структуры**

**Реализация упорядоченного динамического списка**

Линейный список – это набор связанных однотипных элементов, в котором каждый элемент каким-то образом определяет следующий за ним элемент. В отличие от стека и очереди операции добавления и удаления можно применять к любым элементам этой структуры. Фактически, стек и очередь можно считать некими частными случаями линейного списка.

Стандартный набор операций со списком:

* Добавление нового элемента до или после заданного с проверкой возможности добавления элемента
* Удаление заданного элемента
* Проход по списку от первого элемента к последнему с выполнением заданных действий
* Поиск элемента в списке

Существует два способа реализации статического линейного списка: в первом случае номер элемента структуры соответствует номеру ячейки массива, а во втором случае используются указатели. Т.е. в ячейке массива должна находиться информационная часть элемента и указатель на индекс ячейки, в которой находится следующий элемент структуры. В данной курсовой работе использовался первый способ, поэтому рассмотрим его поближе.

Пусть условно имеем структуру из 3-ех элементов и массив размером 5 ячеек. В первой ячейке будет находится первый элемент структуры, во второй ячейке – второй и так далее. При условии, что элементов у нас меньше, чем количество ячеек массива, у нас всегда в правой части массива будут находится пустые ячейки. Не может быть такого, что в первой ячейке есть элемент, во второй ничего нет, а в третьей элемент присутствует.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ячейка 0 | Ячейка 1 | Ячейка 2 | Ячейка 3 | Ячейка 4 |
| Элемент №1 | Элемент №2 | Элемент №3 |  |  |

Для того, чтобы вставить элемент в ячейку массива с индексом i < N, где N – количество элементов в структуре, нужно все элементы начиная с i-го и до N-го сдвинуть вправо (конечно же, при наличии свободной ячейки в массиве). Копирование происходит с последней ячейки: N вставляется в N+1 ячейку, N-1-ый – в N-ую и так далее. В освободившуюся ячейку i вставляется интересующий нас элемент.

Для того, чтобы удалить элемент i из структуры, нужно сместить все элементы, начиная с i+1, влево. Копирование идет начиная с ячейки i: i-ый есть i+1-ый и так далее.

Поиск элемента происходит элементарным проходом по массиву. Отображение структуры – аналогично.

При сортированной вставке добавляемый элемент последовательно сравнивается с элементами структуры. Когда мы встречаем элемент больший добавляемого (в случае сортировки по возрастанию), то осуществляем вставку перед этим большим элементом.

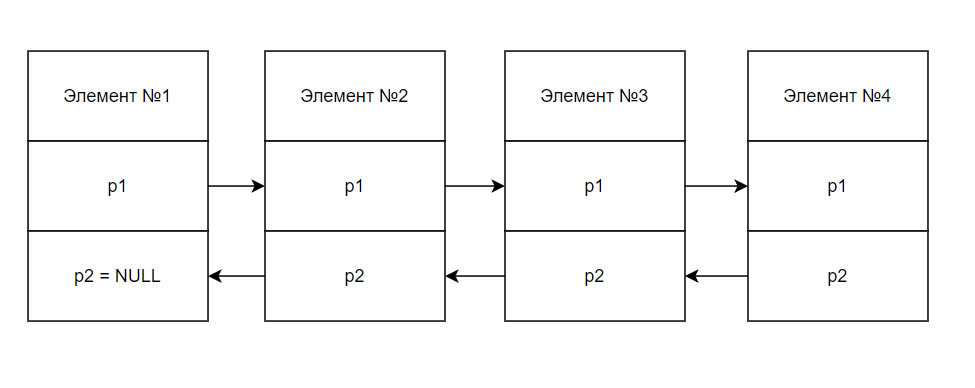
**Реализация упорядоченного динамического списка**

Динамическая реализация линейного списка основана на динамическом выделении и освобождении памяти для элементов списка. Логическая последовательность элементов списка создается ссылочными переменными с адресами последующих элементов.

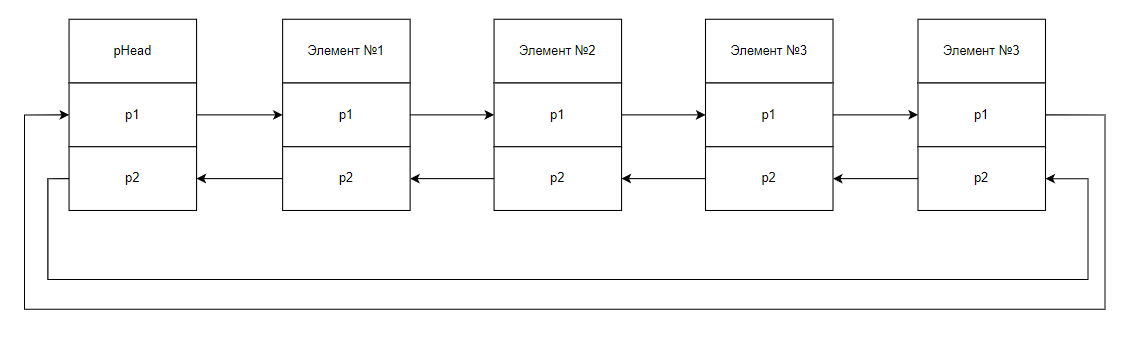
Для удобства реализации, список всегда содержит хотя бы один элемент-заголовок с адресом первого реального элемента списка. Это позволяет унифицировать процедуры добавления и удаления крайних элементов и устранить некоторые проверки. Адрес элемента-заголовка задается переменной-указателем “pHead”. Эта переменная устанавливается при первоначальном создании списка и в дальнейшем не изменяется. Для реализации основных действий используются вспомогательные ссылочные переменные.

Одна из разновидностей линейного списка – упорядоченный список. В нем элементы выстраиваются в соответствии с заданным порядком, например – целые числа по возрастанию, текстовые строки по алфавиту. Для таких списков изменяется процедура добавления – новый элемент должен вставляться в соответствующее место для сохранения порядка элементов. Например, если порядок элементов определяется целыми числами по возрастанию, то при поиске подходящего места надо найти первый элемент, больший заданного и выполнить вставку ПЕРЕД этим элементом.

Структура линейный список имеет более усложненный вариант – двунаправленный линейный список. В нем каждый элемент “знает” обоих своих соседей, как левого, так и правого. Для этого у каждого элемента есть два связующих поля: указатель на элемент слева и указатель на элемент справа.



Поскольку двунаправленные списки являются симметричными, удобно сделать их замкнутыми, кольцевыми: правая ссылка последнего элемента указывает на заголовок, а левая ссылка заголовка – на последний элемент. Адрес заголовка определяется указателем pHead.



В данной курсовой работе фиктивный заголовочный элемент не использовался.

Рассмотрим реализацию основных списковых операций:

**Создание пустого списка:**

* выделяется память под заголовок, адресуемая указателем pHead
* оба ссылочных поля заголовка устанавливаются в адрес самого заголовка: pHead^.left := pHead; pHead^.right := pHead;

**Проход по списку:**

* введение вспомогательной ссылочной переменной pTemp;
* установка pTemp в адрес первого реального элемента   
  (pTemp := pHead^. right);
* организация цикла по условию достижения заголовка;
* в цикле обработка очередного элемента с помощью указателя pTemp и изменения этого указателя на следующий (pTemp := pTemp^.right).

**Проход в обратном направлении:**

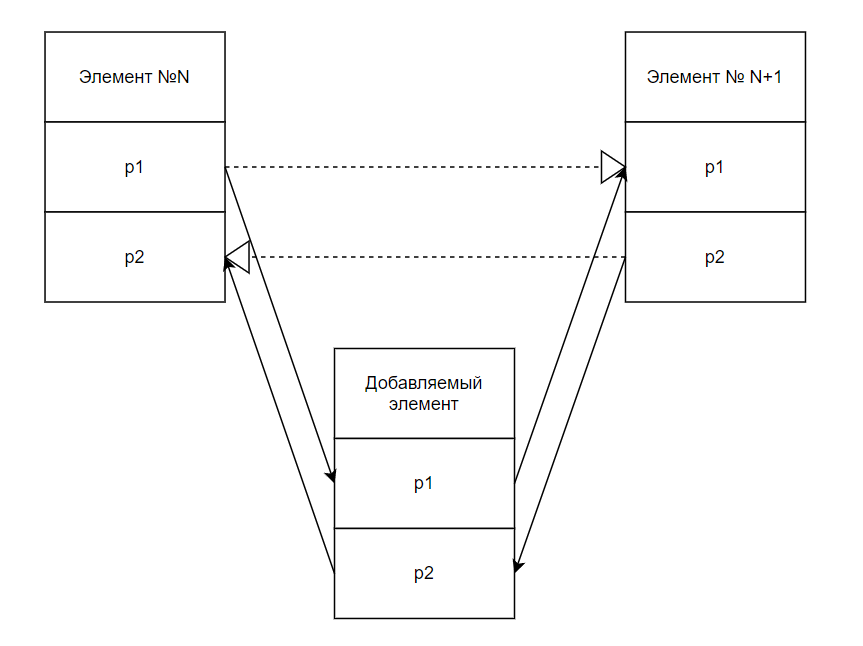
* устанавливаем начальное значение указателя текущего элемента на последний элемент списка: pCurrent := pHead^.left;
* организуем цикл прохода до достижения заголовка **while** pCurrent <> pHead **do** pCurrent := pCurrent^.left;

**Поиск заданного элемента:**

* установка вспомогательного указателя в адрес первого элемента списка
* организация цикла прохода по списку с завершением либо по совпадению информационной части элемента с заданным значением, либо по достижению заголовка списка
* после завершения цикла проверка значения вспомогательного указателя и вывод итога об успешности поиска.

**Добавление элемента**:

Допустим, нам нужно добавить новый элемент после заданного указателем pCurrent. Пусть как обычно новый элемент определяется указателем pTemp. Тогда для вставки его в список надо настроить оба его ссылочных поля, изменить правое ссылочное поле у текущего элемента и левое ссылочное поле у его правого соседа.



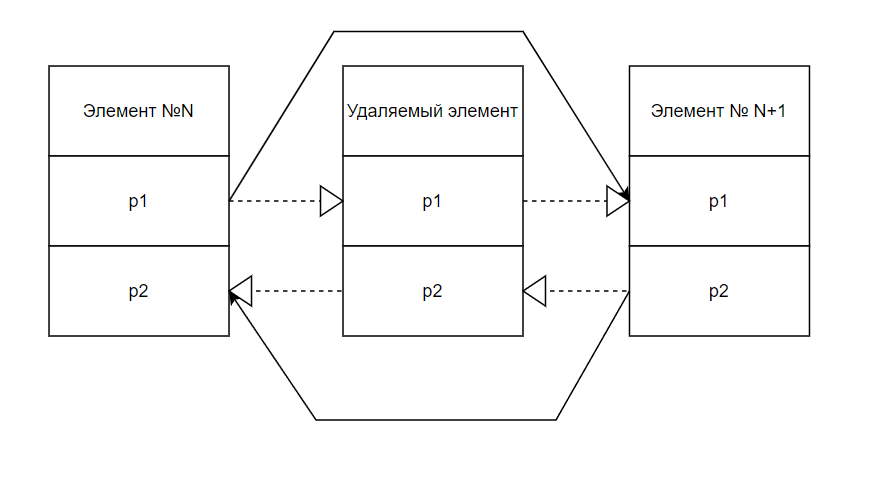
Необходимые присваивания:

pTemp^.right := pCurrent^.right;  
pTemp^.left := pCurrent;  
pCurrent^.right^.left := pTemp;   
pCurrent^.right := pTemp;

Аналогично реализуется **добавление нового элемента перед заданным элементом**, только вместо правого соседа обрабатывается левый сосед текущего элемента.

**Удаление заданного элемента**:

Для этого необходимо измененить ссылочные поля. Надо изменить правый указатель у левого соседа удаляемого элемента и левый указатель у правого соседа.



Если удаляемый элемент адресуется указателем pCurrent, то pCurrent^.left определяет адрес левого соседа, а pCurrent^.right – адрес правого соседа. Тогда необходимые изменения реализуются так:

pCurrent^.left^.right := pCurrent^.right;  
 pCurrent^.right^.left := pCurrent^.left;

# **Глава 3. Описание программного комплекса**

## **3.1. Описание структуры проекта**

Для создания проекта была использована среда разработки Microsoft Visual Studio 2022. Проект имеет следующую иерархию:

Course\_Work

Bank.h

Course\_work.cpp

Course\_work.vcxproj  
 Course\_work.sln  
 ExeptionInput.h  
 FileWorking.h

## 2.3 Краткое описание разработанных классов

При выполнении курсовой работы было создано 4 класса и одна главная подпрограмма с функцией main(). Через эту подпрограмму осуществляется диалог с пользователем. Краткое описание разработанных классов (более подробно см. в пункте «Руководство программиста»):

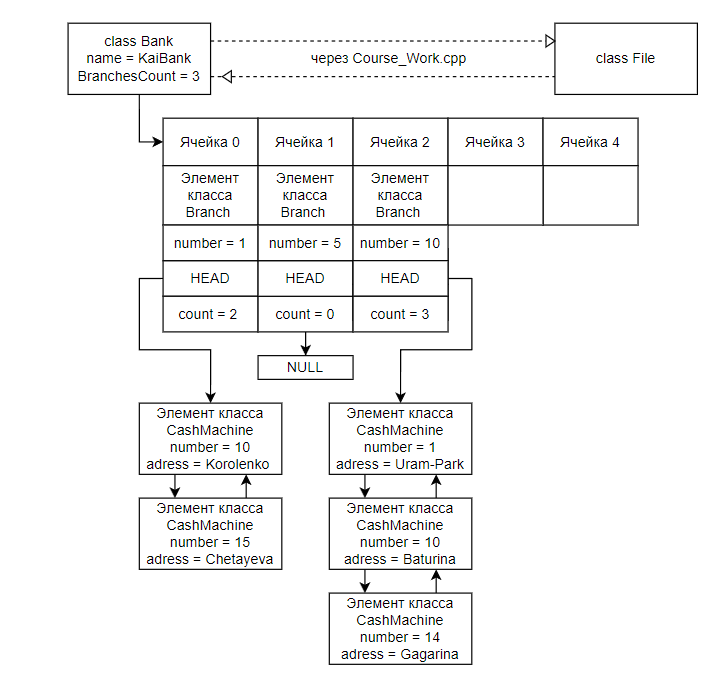
**Класс CashMachine.** Это класс, задающий шаблон для объектов «банкомат», являющихся единицей нашей комбинированной структуры. Для данного класса определены методы для получения доступа и изменения полей экземпляра этого класса (Get/Set методы).

**Класс Branch.** Это класс, задающий шаблон для объектов «филиал», являющихся композицией банкоматов. Содержит в себе двунаправленный упорядоченный динамический линейный список. Содержит в себе методы удаления/добавления экземпляров класса CashMachine в список; поиск элемента в списке и вывод элементов списка на экран; Get/Set методы и некоторые вспомогательные методы.

**Класс Bank.** Это класс, задающий шаблон для объектов «банк», являющихся композицией филиалов. Содержит в себе упорядоченный статический линейный список филиалов. Содержит в себе методы удаления/добавления экземпляров класса Branch в статический список; поиск элемента в списке и вывод элементов статического списка на экран; Get/Set методы и некоторые вспомогательные методы.

**Класс File.** Данный класс предназначен для файлового ввода-вывода. Имеется два метода Input и Output для осуществления ввода и вывода соответственно.

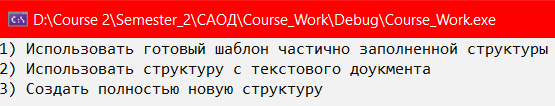
# 2.4 Схема структуры проекта (взаимодействия классов)



# 3. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

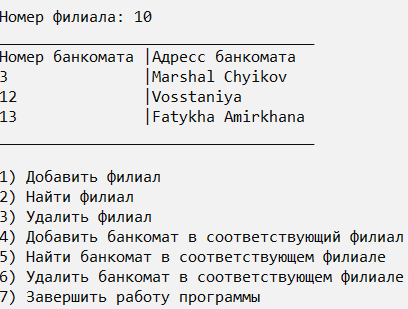
Для начала работы программы пользователь должен запустить файл Course\_work.exe.

После запуска программы перед пользователем предстоит следующий выбор:



Если пользователь введет 1, то в программе будет использоваться уже заполненная некоторыми данными структура. Это бывает удобно, когда нужно быстро проверить работоспособность программы без ручного ввода данных. Если же пользователь введет цифру 2, то в качестве структуры возьмутся данные из файла DocBank.txt. При выборе числа 3 структура будет пустой.

Теперь рассмотрим поближе основной функционал программы:



1. Добавить филиал – добавляет новый пустой филиал банка при условии, что статический линейный список филиалов не заполнен и филиал с таким номером не существует.
2. Найти филиал – осуществляем поиск филиала в массиве
3. Удалить филиал – очищаем соответствующую ячейку массива. Для этого нужно освободить из памяти все элементы этого филиала.
4. Добавить банкомат в соответствующий филиал – пользователь сначала вводит номер филиала, в который необходимо добавить банкомат, затем вводит номер банкомата, а потом его адрес.
5. Найти банкомат в соответствующем филиале - осуществляем поиск банкомата в динамическом двунаправленном списке.
6. Удалить банкомат в соответствующем филиале - пользователь сначала вводит номер филиала, в котором необходимо удалить банкомат, затем вводит номер интересующего банкомата. Ввод номера достаточен, т.к. это исчерпывающая информация банкомата в данном филиале – повторений номеров нет.
7. Завершить работу программы – после выбора пользователем числа 7, ему предлагается сохранить данные в файл out.txt. Если же будет введено что-то отличное от числа 1, то сохранения данных не будет.

# 4. РУКОВОДСТВО ПРОГРАММИСТА

Данный проект был создан для демонстрации работы комбинированной структуры «упорядоченный статический список упорядоченных динамических списков» с использованием объектно-ориентированного программирования (ООП). Для создания программы был выбран язык C++ с поддержкой ООП. В качестве среды разработки было выбрано приложение Microsoft Visual Studio 2019.

Главный структура - упорядоченный статический список. Подструктура – упорядоченные динамические списки.

В ходе разработки было создано 4 класса. Три из них находятся в заголовочном файле Bank.h, а четвертый расположен в другом заголовочном файле под названием File\_working.h. Также был создан отдельный файл с главной подпрограммой, содержащей функцию «int main()». С полными алгоритмами каждого из классов можно ознакомиться в приложении к пояснительной записке. Для взаимодействия с таблицей банка главная программа обращается только к объекту класса Bank, а тот уже взаимодействует со всеми остальными классами.

Описание методов классов:

**Класс CashMachine.** Экземпляры этого класса являются некой единицей нашей структуры. У этого класса есть 4 поля: номер банкомата, адрес банкомата, указатель на следующий за текущим объектом экземпляр этого класса, указатель на предыдущий за текущим объектом экземпляр этого класса.

|  |  |
| --- | --- |
| Метод класса | Описание метода |
| CashMachine(int number, std::string adress) | Конструктор класса |
| void SetAdress(std::string adress) | Метод для изменения адреса банкомата |
| std::string GetAdress() | Метод для получения адреса банкомата |
| Void SetNumber(int number) | Метод для изменения номера банкомата |
| Int GetNumber() | Метод для получения номера банкомата |
| Void SetNext(CashMachine\* next) | Метод для изменения указателя на следующий объект |
| CashMachine\* GetNext() | Метод для возврата указателя на следующий элемент |
| void SetPrevious(CashMachine\* previous) | Метод для изменения указателя на предыдущий элемент |
| CashMachine\* GetPrevious() | Метод для получения указателя предыдущего элемента |
| ~CashMachine() | Деструктор класса CashMachine |

**Класс Branch.** Это класс, задающий шаблон для объектов «филиал», являющихся композицией банкоматов. Содержит в себе двунаправленный упорядоченный динамический линейный список. Содержит в себе методы удаления/добавления экземпляров класса CashMachine в список; поиск элемента в списке и вывод элементов списка на экран и другие вспомогательные методы.

|  |  |
| --- | --- |
| Метод класса | Описание метода |
| Branch(int number) | Конструктор класса |
| void SetNumber(int number) | Метод для изменения номера филиала |
| int GetNumber() | Метод для получения номера филиала |
| void SetHead(CashMachine\* Head) | Метод для изменения первого элемента динамического списка |
| CashMachine\* GetHead() | Метод для получения первого элемента динамического списка |
| bool IsEmpty() | Метод для проверки на пустоту динамического списка |
| void AddSortedCashMachine(int number, std::string adress) | Метод для добавления нового банкомата в динамический список |
| bool FindElement(int element) | Метод для поиска банкомата в динамическом списке |
| bool FindElementAdditionall(int element) | Вспомогательный метод для поиска, который не выводит текст |
| void DeleteElement(int element) | Метод для удаления банкомата |
| void ShowBranch() | Метод для вывода динамического списка |
| ~Branch() | Деструктор класса |

**Класс Bank.** Это класс, задающий шаблон для объектов «банк», являющихся композицией филиалов. Содержит в себе упорядоченный статический линейный список филиалов. Содержит в себе методы удаления/добавления экземпляров класса Branch в статический список; поиск элемента в списке и вывод элементов статического списка на экран и некоторые другие вспомогательные методы.

|  |  |
| --- | --- |
| Метод класса | Описание метода |
| Bank(std::string name) | Конструктор класса с передачей аргумента |
| Bank() | Конструктор по умолчанию для создания частично заполненной структуры |
| void SetName(std::string name) | Метод для изменения имени банка |
| std::string GetName() | Метод для получения имени банка |
| int GetBranchesCount() | Метод для получения количества филиалов |
| void SetBranch(Branch\* Branches[]) | Метод для изменения статического линейного списка филиалов |
| Branch\*\* GetBranches() | Метод для получения статического линейного списка филиалов |
| bool IsFull() | Метод для проверки заполненности статического линейного списка филиалов |
| bool IsEmpty() | Метод для проверки пустоты статического линейного списка филиалов |
| bool SearchElement(int BranchNumber) | Метод для поиска филиала |
| bool SearchElementAdditionall(int BranchNumber) | Вспомогательный метод для поиска филиала без вывода текста в консоль |
| void AddSortedBranch(int BranchNumber) | Метод для сортированного добавления филиала |
| int GetElementIndex(int element) | Метод для получения индекса соответствующего филиала в статическом линейном списке филиалов |
| void DeleteBranch(int BranchNumber) | Метод для удаления филиала |
| void ShowBank() | Метод для вывода содержимого банка |
| ~Bank() | Деструктор класса |

**Класс File.** Данный класс предназначен для файлового ввода-вывода. Имеется два метода Input и Output для осуществления ввода и вывода соответственно.

|  |  |
| --- | --- |
| Метод класса | Описание метода |
| bool Input(Bank\* bank) | Метод для заполнения структуры с помощью информации, заключенной в текстовом файле |
| bool Output(Bank\* bank) | Метод для вывода структуры в текстовый файл |

# ВЫВОД

В результате выполнения данной курсовой работы была создана программа, реализующая структуру данных *упорядоченный статический список упорядоченных динамических списков,*  используя принципы объектно-ориентированного программирования. Были повторены и закреплены знания, полученные в ходе изучения курсов «Структуры и алгоритмы обработки данных» и «Объектно-ориентированное программирование», а также были выработаны новые навыки владения языком программирования С++.

На основе исходных данных выполнены все требования к реализации программы, получен ожидаемый результат.

# СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Козин А. Н. Структуры и алгоритмы обработки данных. Учебно-методическое пособие. Издательство: Татарский институт содействия бизнесу, 2003.

2. Козин А. Н. Объектно-ориентированное программирование. Учебно-методическое пособие. Издательство: Академия Управления «ТИСБИ», 2006.

# ПРИЛОЖЕНИЕ. ЛИСТИНГИ ВСЕХ ФАЙЛОВ ПРОЕКТА

Course\_Work.cpp – главная программа

#include <iostream>

#include <Windows.h>

#include <ctime>

#include "Bank.h"

#include "FileWorking.h"

#include "ExceptionInput.h"

using namespace std;

void Menu()

{

std::cout <<

"1) Добавить филиал\n" <<

"2) Найти филиал\n" <<

"3) Удалить филиал\n" <<

"4) Добавить банкомат в соответствующий филиал\n" <<

"5) Найти банкомат в соответствующем филиале\n" <<

"6) Удалить банкомат в соответствующем филиале\n" <<

"7) Завершить работу программы\n";

}

int main()

{

setlocale(LC\_ALL, "rus");

srand(time(NULL));

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

char WhatToUse;

std::cout <<

"1) Использовать готовый шаблон частично заполненной структуры\n" <<

"2) Использовать структуру с текстового доукмента\n" <<

"3) Создать полностью новую структуру\n";

WhatToUse = charinput();

if (WhatToUse == '1')

{

bool working = true;

char choice;

string str;

int n;

int k;

Bank\* KaiBank = new Bank();

while (working)

{

std::cout << "\_\_\_\_\_\_\_\_\_\_\_" << KaiBank->GetName() << "\_\_\_\_\_\_\_\_\_\_\_\n" << std::endl;

KaiBank->ShowBank();

Menu();

choice = charinput();

switch (choice)

{

case '1':

std::cout << "Введите номер филиала: ";

n = intinput();

KaiBank->AddSortedBranch(n);

break;

case '2':

std::cout << "Введите номер филиала для поиска: ";

n = intinput();

KaiBank->SearchElement(n);

break;

case '3':

std::cout << "Введите номер филиала для удаления: ";

n = intinput();

KaiBank->DeleteBranch(n);

break;

case '4':

std::cout << "Введите номер филиала для добавления банкомата: ";

k = intinput();

if (!KaiBank->SearchElementAdditionall(k))

{

std::cout << "Филиал не найден!" << std::endl;

break;

}

k = KaiBank->GetElementIndex(k);

std::cout << "Введите номер банкомата: ";

n = intinput();

std::cout << "Введите адрес банкомата: ";

std::cin.ignore();

std::getline(std::cin, str);

KaiBank->GetBranches()[k]->AddSortedCashMachine(n, str);

break;

case '5':

std::cout << "Введите номер филиала для поиска банкомата: ";

k = intinput();

if (!KaiBank->SearchElementAdditionall(k))

{

std::cout << "Филиал не найден!" << std::endl;

break;

}

k = KaiBank->GetElementIndex(k);

std::cout << "Введите номер банкомата для поиска: ";

n = intinput();

KaiBank->GetBranches()[k]->FindElement(n);

break;

case '6':

std::cout << "Введите номер филиала, в котором нужно удалить банкомат: ";

k = intinput();

if (!KaiBank->SearchElementAdditionall(k))

{

std::cout << "Филиал не найден!" << std::endl;

break;

}

k = KaiBank->GetElementIndex(k);

std::cout << "Введите номер банкомата для удаления: ";

n = intinput();

KaiBank->GetBranches()[k]->DeleteElement(n);

break;

case '7':

std::cout << "Желаете ли сохранить данные в файл?(1): ";

choice = charinput();

if (choice == '1')

{

File file("DocBank.txt", "out.txt");

file.Output(KaiBank);

}

delete KaiBank;

return 0;

default:

break;

}

}

}

else if (WhatToUse == '2')

{

string name;

Bank\* DocBank = new Bank("");

bool working = true;

char choice;

string str;

int n;

int k;

File file("DocBank.txt", "out.txt");

if (!file.Input(DocBank))

{

cout << "Не удалось взять структуру из файла!\n";

cout << "Введите название вашего банка: ";

cin >> name;

DocBank->SetName(name);

}

while (working)

{

std::cout << "\_\_\_\_\_\_\_\_\_\_\_" << DocBank->GetName() << "\_\_\_\_\_\_\_\_\_\_\_\n" << std::endl;

DocBank->ShowBank();

Menu();

choice = charinput();

switch (choice)

{

case '1':

std::cout << "Введите номер филиала: ";

n = intinput();

DocBank->AddSortedBranch(n);

break;

case '2':

std::cout << "Введите номер филиала для поиска: ";

n = intinput();

DocBank->SearchElement(n);

break;

case '3':

std::cout << "Введите номер филиала для удаления: ";

n = intinput();

DocBank->DeleteBranch(n);

break;

case '4':

std::cout << "Введите номер филиала для добавления банкомата: ";

k = intinput();

if (!DocBank->SearchElementAdditionall(k))

{

std::cout << "Филиал не найден!" << std::endl;

break;

}

k = DocBank->GetElementIndex(k);

std::cout << "Введите номер банкомата: ";

n = intinput();

std::cout << "Введите адрес банкомата: ";

std::cin.ignore();

std::getline(std::cin, str);

DocBank->GetBranches()[k]->AddSortedCashMachine(n, str);

break;

case '5':

std::cout << "Введите номер филиала для поиска банкомата: ";

k = intinput();

if (!DocBank->SearchElementAdditionall(k))

{

std::cout << "Филиал не найден!" << std::endl;

break;

}

k = DocBank->GetElementIndex(k);

std::cout << "Введите номер банкомата для поиска: ";

n = intinput();

DocBank->GetBranches()[k]->FindElement(n);

break;

case '6':

std::cout << "Введите номер филиала, в котором нужно удалить банкомат: ";

k = intinput();

if (!DocBank->SearchElementAdditionall(k))

{

std::cout << "Филиал не найден!" << std::endl;

break;

}

k = DocBank->GetElementIndex(k);

std::cout << "Введите номер банкомата для удаления: ";

n = intinput();

DocBank->GetBranches()[k]->DeleteElement(n);

break;

case '7':

std::cout << "Желаете ли сохранить данные в файл?(1): ";

choice = charinput();

if (choice == '1')

{

File file("DocBank.txt", "out.txt");

file.Output(DocBank);

}

delete DocBank;

return 0;

break;

default:

break;

}

}

}

else if (WhatToUse == '3')

{

string name;

Bank\* EmptyBank = new Bank("");

bool working = true;

char choice;

string str;

int n;

int k;

std::cout << "Введите название вашего банка: ";

cin >> str;

EmptyBank->SetName(str);

while (working)

{

std::cout << "\_\_\_\_\_\_\_\_\_\_\_" << EmptyBank->GetName() << "\_\_\_\_\_\_\_\_\_\_\_\n" << std::endl;

EmptyBank->ShowBank();

Menu();

choice = charinput();

switch (choice)

{

case '1':

std::cout << "Введите номер филиала: ";

n = intinput();

EmptyBank->AddSortedBranch(n);

break;

case '2':

std::cout << "Введите номер филиала для поиска: ";

n = intinput();

EmptyBank->SearchElement(n);

break;

case '3':

std::cout << "Введите номер филиала для удаления: ";

n = intinput();

EmptyBank->DeleteBranch(n);

break;

case '4':

std::cout << "Введите номер филиала для добавления банкомата: ";

k = intinput();

if (!EmptyBank->SearchElementAdditionall(k))

{

std::cout << "Филиал не найден!" << std::endl;

break;

}

k = EmptyBank->GetElementIndex(k);

std::cout << "Введите номер банкомата: ";

n = intinput();

std::cout << "Введите адрес банкомата: ";

std::cin.ignore();

std::getline(std::cin, str);

EmptyBank->GetBranches()[k]->AddSortedCashMachine(n, str);

break;

case '5':

std::cout << "Введите номер филиала для поиска банкомата: ";

k = intinput();

if (!EmptyBank->SearchElementAdditionall(k))

{

std::cout << "Филиал не найден!" << std::endl;

break;

}

k = EmptyBank->GetElementIndex(k);

std::cout << "Введите номер банкомата для поиска: ";

n = intinput();

EmptyBank->GetBranches()[k]->FindElement(n);

break;

case '6':

std::cout << "Введите номер филиала, в котором нужно удалить банкомат: ";

k = intinput();

if (!EmptyBank->SearchElementAdditionall(k))

{

std::cout << "Филиал не найден!" << std::endl;

break;

}

k = EmptyBank->GetElementIndex(k);

std::cout << "Введите номер банкомата для удаления: ";

n = intinput();

EmptyBank->GetBranches()[k]->DeleteElement(n);

break;

case '7':

std::cout << "Желаете ли сохранить данные в файл?(1): ";

choice = charinput();

if (choice == '1')

{

File file("DocBank.txt", "out.txt");

file.Output(EmptyBank);

}

delete EmptyBank;

return 0;

default:

break;

}

}

}

FileWorking.h - программа для работы с файлами

#pragma warning (disable : 4996)

#include "Bank.h"

#include <fstream>

#include <iostream>

using namespace std;

inline bool isInteger(const std::string& s)

{

if (s.empty() || ((!isdigit(s[0])) && (s[0] != '-') && (s[0] != '+')))

return false;

char\* p;

strtol(s.c\_str(), &p, 10);

return (\*p == 0);

}

class File

{

private:

string input\_path;

string output\_path;

public:

File(string in\_path, string out\_path)

{

input\_path = in\_path;

output\_path = out\_path;

};

bool Input(Bank\* bank)

{

string text, Branch;

int number = -1;

char buffer[100];

char\* f;

ifstream file(input\_path);

if (!file.is\_open())

{

return false;

}

if (!file.eof())

{

file.getline(buffer, 100);

text = buffer;

bank->SetName(text);

}

else

{

cout << "Файл пуст!\n";

return false;

}

while (!file.eof())

{

file.getline(buffer, 100);

f = strtok(buffer, " ");

if (f == NULL)

{

continue;

}

if (f)

{

Branch = f;

}

bank->AddSortedBranch(stoi(Branch));

f = strtok(NULL, " ");

while (f)

{

if (f)

{

text = f;

}

f = strtok(NULL, " ");

if (!isInteger(f))

{

text += " ";

text += f;

f = strtok(NULL, " ");

}

if (f)

{

text += " ";

text += f;

}

f = strtok(NULL, " ");

if (f)

{

number = stoi(f);

}

else

{

return false;

}

bank->GetBranches()[bank->GetElementIndex(stoi(Branch))]->AddSortedCashMachine(number, text);

f = strtok(NULL, " ");

}

}

file.close();

return true;

}

bool Output(Bank\* bank)

{

ofstream file(output\_path);

if (!file.is\_open())

{

return false;

}

file << "\_\_\_\_\_\_\_\_\_\_\_" << bank->GetName() << "\_\_\_\_\_\_\_\_\_\_\_\n" << std::endl;

for (int i = 0; i < bank->GetBranchesCount(); i++)

{

bank->GetBranches()[i]->GetNumber();

CashMachine\* Current = bank->GetBranches()[i]->GetHead();

file << "Номер филиала: " << bank->GetBranches()[i]->GetNumber() << "\n";

file << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

file << "Номер банкомата\t|Адресс банкомата\n";

while (Current != NULL)

{

file << Current->GetNumber() << "\t\t|";

file << Current->GetAdress() << '\n';

Current = Current->GetNext();

}

file << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

file << std::endl;

}

file.close();

return true;

}

};

CashMachine – класс банкоматов

class CashMachine

{

public:

CashMachine(int number, std::string adress)

{

this->number = number;

this->adress = adress;

}

void SetAdress(std::string adress)

{

this->adress = adress;

}

std::string GetAdress()

{

return adress;

}

void SetNumber(int number)

{

this->number = number;

}

int GetNumber()

{

return number;

}

void SetNext(CashMachine\* next)

{

this->next = next;

}

CashMachine\* GetNext()

{

return next;

}

void SetPrevious(CashMachine\* previous)

{

this->previous = previous;

}

CashMachine\* GetPrevious()

{

return previous;

}

~CashMachine()

{

number = 0;

adress = "";

next = NULL;

previous = NULL;

}

private:

int number;

std::string adress;

CashMachine\* next;

CashMachine\* previous;

};

Branch – класс филиалов

class Branch

{

public:

Branch(int number)

{

this->number = number;

this->Head = NULL;

countOfElements = 0;

}

void SetNumber(int number)

{

this->number = number;

}

int GetNumber()

{

return this->number;

}

void SetHead(CashMachine\* Head)

{

this->Head = Head;

}

CashMachine\* GetHead()

{

return this->Head;

}

bool IsEmpty()

{

if (Head == NULL)

{

return true;

}

else

{

return false;

}

}

void AddSortedCashMachine(int number, std::string adress)

{

if (IsEmpty())

{

Head = new CashMachine(number, adress);

Head->SetNext(NULL);

countOfElements++;

}

else if (FindElementAdditionall(number))

{

std::cout << "Банкомат с таким номером уже существует!" << std::endl;

return;

}

else

{

CashMachine\* AddElem = new CashMachine(number, adress);

CashMachine\* Current = Head;

while (number > Current->GetNumber())

{

if (Current->GetNext() == NULL)

{

AddElem->SetPrevious(Current);

AddElem->SetNext(Current->GetNext());

Current->SetNext(AddElem);

countOfElements++;

return;

}

Current = Current->GetNext();

}

if (Current == Head)

{

AddElem->SetPrevious(NULL);

AddElem->SetNext(Head);

Head->SetPrevious(AddElem);

Head = AddElem;

countOfElements++;

}

else

{

AddElem->SetPrevious(Current->GetPrevious());

AddElem->SetNext(Current);

(AddElem->GetPrevious())->SetNext(AddElem);

Current->SetPrevious(AddElem);

countOfElements++;

} }

}

bool FindElement(int element)

{

bool WasFound = false;

if (IsEmpty())

{

std::cout << "Филиал пуст. Добавьте банкоматы" << std::endl;

return false;

}

else

{

CashMachine\* Current = Head;

while (Current != NULL)

{

if (Current->GetNumber() == element)

{

std::cout << "Банкомат под номером " << element << " найден!" << std::endl;

return true;

}

Current = Current->GetNext();

}

std::cout << "Банкомат под номером " << element << " не найден!" << std::endl;

return false;

}

}

bool FindElementAdditionall(int element)

{

bool WasFound = false;

CashMachine\* Current = Head;

while (Current != NULL)

{

if (Current->GetNumber() == element)

{

return true;

}

Current = Current->GetNext();

}

return false;

}

void DeleteElement(int element)

{

bool ItsMax = false;

if (IsEmpty())

{

std::cout << "Филиал пуст. Добавьте банкоматы" << std::endl;

return;

}

else if (!FindElementAdditionall(element))

{

std::cout << "Элемент не обнаружен" << std::endl;

return;

}

else

{

CashMachine\* Current = Head;

while (Current->GetNumber() != element)

{

Current = Current->GetNext();

if (Current->GetNext() == NULL)

{

ItsMax = true;

break;

}

}

if (Current == Head)

{

Head = Head->GetNext();

countOfElements--;

delete Current;

}

else

{

if (ItsMax)

{

(Current->GetPrevious())->SetNext(NULL);

countOfElements--;

delete Current;

}

else

{

(Current->GetNext())->SetPrevious(Current->GetPrevious());

(Current->GetPrevious())->SetNext(Current->GetNext());

countOfElements--;

delete Current;

}

}

}

}

void ShowBranch()

{

CashMachine\* Current = Head;

std::cout << "Номер филиала: " << number << "\n";

std::cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

std::cout << "Номер банкомата\t|Адресс банкомата\n";

while (Current != NULL)

{

std::cout << Current->GetNumber() << "\t\t|";

std::cout << Current->GetAdress() << '\n';

Current = Current->GetNext();

}

std::cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

std::cout << std::endl;

}

~Branch()

{

this->number = 0;

CashMachine\* Current = Head;

CashMachine\* temp;

while (Current != NULL)

{

temp = Current;

Current = Current->GetNext();

delete temp;

}

}

private:

int number;

int countOfElements;

CashMachine\* Head;

};

Bank – класс банков

class Bank

{

public:

Bank(std::string name)

{

this->name = name;

BranchesCount = 0;

}

Bank()

{

this->name = "KaiBank";

BranchesCount = 0;

AddSortedBranch(1);

AddSortedBranch(4);

AddSortedBranch(10);

Branches[GetElementIndex(1)]->AddSortedCashMachine(3, "Короленко 12");

Branches[GetElementIndex(1)]->AddSortedCashMachine(4, "Батурина 33");

Branches[GetElementIndex(1)]->AddSortedCashMachine(10, "Проспект Победы 31");

Branches[GetElementIndex(4)]->AddSortedCashMachine(45, "Четаева 18");

Branches[GetElementIndex(4)]->AddSortedCashMachine(10, "Гоголя 32");

Branches[GetElementIndex(4)]->AddSortedCashMachine(1, "Гагарина 1");

Branches[GetElementIndex(10)]->AddSortedCashMachine(13, "Фатыха Амирхана 88");

Branches[GetElementIndex(10)]->AddSortedCashMachine(12, "Восстания 3");

Branches[GetElementIndex(10)]->AddSortedCashMachine(3, "Маршала Чуйкова 32"); }

void SetName(std::string name)

{

this->name = name;

}

std::string GetName()

{

return this->name;

}

int GetBranchesCount()

{

return BranchesCount;

}

void SetBranch(Branch\* Branches[])

{

for(int i = 0; i < MAX\_SIZE; i++)

{

this->Branches[i] = Branches[i];

}

}

Branch\*\* GetBranches()

{

return Branches;

}

bool IsFull()

{

if (BranchesCount == MAX\_SIZE)

{

std::cout << "Статический список филиалов заполнен!" << std::endl;

return 1;

}

else

{

return 0;

}

}

bool IsEmpty()

{

if (BranchesCount == 0)

{

return 1;

}

else

{

return 0;

}

}

bool SearchElement(int BranchNumber)

{

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

{

if (Branches[i]->GetNumber() == BranchNumber)

{

std::cout << "Филиал с таким номером найден под индексом " << i << std::endl;

return 1;

}

}

std::cout << "Филиал не найден!" << std::endl;

return 0;

}

bool SearchElementAdditionall(int BranchNumber)

{

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

{

if (Branches[i]->GetNumber() == BranchNumber)

{

return 1;

}

}

return 0;

}

void AddSortedBranch(int BranchNumber)

{

if (IsEmpty())

{

Branches[0] = new Branch(BranchNumber);

BranchesCount++;

return;

}

else if (IsFull())

{

return;

}

else if (SearchElementAdditionall(BranchNumber))

{

std::cout << "Филиал с таким номером уже существует!" << std::endl;

return;

}

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

{

if (i == BranchesCount)

{

Branches[BranchesCount] = new Branch(BranchNumber);

BranchesCount++;

break;

}

if (Branches[i]->GetNumber() > BranchNumber)

{

for (int j = BranchesCount; j > i; j--)

{

Branches[j] = Branches[j - 1];

}

Branches[i] = new Branch(BranchNumber);

BranchesCount++;

break;

}

}

}

int GetElementIndex(int element)

{

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

{

if (Branches[i]->GetNumber() == element)

{

return i;

}

}

return -1;

}

void DeleteBranch(int BranchNumber)

{

if (IsEmpty())

{

std::cout << "Статический список филиалов пуст. Удалять нечего!" << std::endl;

return;

}

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

{

if (Branches[i]->GetNumber() == BranchNumber)

{

delete Branches[i];

for (int j = i; j < BranchesCount-1; j++)

{

Branches[j] = Branches[j + 1];

}

BranchesCount--;

return;

}

}

std::cout << "Элемент не обнаружен в списке филиалов!\n";

}

void ShowBank()

{

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

{

Branches[i]->ShowBranch();

}

}

~Bank()

{

this->name = "";

BranchesCount = 0;

}

private:

std::string name;

int BranchesCount;

Branch\* Branches[MAX\_SIZE];

};