## Министерство науки и высшего образования Российской Федерации

## федеральное государственное бюджетное образовательное учреждение высшего образования «Казанский национальный исследовательский технический университет им. А.Н.Туполева-КАИ»

«Сomputer-центр»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

по дисциплине

«Структуры и алгоритмы обработки данных»

Выполнил:

студент гр. №4280

Салимов И.Г.

Проверил(а): Шакирзянов Р.М.

Казань 2021

Содержание:

[Постановка задачи 3](#_bookmark0)

[Теоретическое описание используемых структур данных с](#_bookmark1) [алгоритмами реализации основных операций 5](#_bookmark1)

[Описание основных понятий и механизмов ООП 11](#_bookmark2)

[Описание всех разработанных классов (объектная модель) 13](#_bookmark3)

[Описание демонстрационного модуля с характеристикой](#_bookmark4) [использованных стандартных компонентов 16](#_bookmark4)

[Описание структуры проекта 20](#_bookmark5)

[Листинги всех подпрограмм с краткими комментариями 21](#_bookmark6)

[Список использованной литературы 33](#_bookmark7)

# Постановка задачи

Цель работы: получение навыков разработки объектных программ, включая создание набора собственных взаимосвязанных классов для объектной реализации специализированного контейнера. Контейнер предназначен для хранения и обработки данных некоторой информационной задачи. Контейнер представляет собой двухуровневую структуру данных, в которой уровни реализуются разными способами – один статически на базе массива (непрерывная реализация), другой – динамически с использованием адресных связей (связная реализация).

Исходные данные:

Объектная реализация контейнера на основе комбинированной структуры «Двунаправленный список массивов-стеков».

Задача «Производственное предприятие»

* Информационные объекты: оборудование цеха предприятия (свойства – НаименованиеОборудования, ГодВыпуска)
* Оборудование объединяется в рамках объекта Цех (свойство – НомерЦеха)
* Цеха объединяются в рамках объекта-контейнера Предприятие (свойство – Название)

Требования

1. Полная объектная реализация с определением классов для всех элементов реализуемой структуры: информационные объекты (обязательно!), объекты-элементы списка (динамическая реализация), объекты-списки, объект-контейнер
2. Имена классов, свойств и методов должны носить содержательный смысл и соответствовать информационной задаче
3. Соблюдение принципа инкапсуляции – использование в классах только закрытых свойств и реализация необходимого набора методов доступа
4. Реализация в классах всех необходимых методов: конструкторы, методы доступа к свойствам, методы добавления и удаления на каждом из двух уровней, метод поиска (при необходимости)
5. Возможность сохранения всей структуры во внешнем файле с обратной загрузкой
6. Наличие модуля, демонстрирующего все возможности созданной библиотеки классов и обладающего удобным оконным пользовательским интерфейсом
7. Язык и среда разработки – по выбору: Delphi, Java, C++, С#

# Теоретическое описание используемых структур данных с алгоритмами реализации основных операций

Двунаправленный список. Здесь ссылки в каждом узле указывают на предыдущий и на последующий узел в списке. Как и односвязный список, двусвязный допускает только последовательный доступ к элементам, но при этом дает возможность перемещения в обе стороны. В этом списке проще производить удаление и перестановку элементов, так как легко доступны адреса тех элементов списка, указатели которых направлены на изменяемый элемент. Стандартный набор операций со списком включает:

* добавление нового элемента после заданного или перед заданным элементом с проверкой возможности добавления элемента
* удаление заданного элемента
* проход по списку от первого элемента к последнему и наоборот с выполнением заданных действий
* поиск в списке заданного элемента

Сам двунаправленный список состоит из хранимый сущности – в моем случае это Company под название Data, ссылки на первый элемент – Head, последнего элемента – Tail и количества элементов в самом списке – Count.

**Создание пустого списка** включает:

* выделение памяти под сам объект;
* ссылки остаются пустыми, т.е. **null**

**Поиск заданного элемента** включает:

* установку вспомогательного указателя на «голову» списка
* проход по всей коллекции до указанного, переходя на ссылку последующего элемента
* при нахождении соответствия немедленный выход и возвращение ссылки найденного элемента

**Обратный поиск элемента** включает:

* установку вспомогательного указателя на «хвост» списка
* проход по всей коллекции до заданного элемента, переходя на предыдущий элемент
* при нахождении заданного элемента немедленный выход и возвращение ссылки найденного элемента

**Удаление заданного элемента** включает:

* поиск удаляемого элемента по заданным параметрам
* переопределение ссылок на удаляемый элемент у предыдущего и последующего элемента на друг друга, для удаления ссылок на удаляемый элемент и добавления ссылок для исключения разрыва между элементами
* уменьшение переменной, хранящей число количества элементов в списке

**Добавление** нового элемента после заданного включает:

* поиск элемента по заданным параметрам
* переопределение ссылок у найденного и последующего (если он есть) элементов на следующий и предыдущий соответственно на новый элемент
* увеличение переменной, хранящей число количества элементов в списке

**Добавление** нового элемента перед заданным включает:

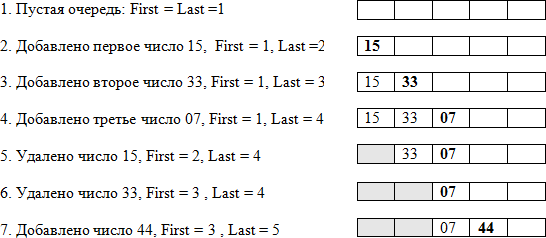
* обратный поиск заданного элемента
* переопределение ссылок у найденного и предыдущего (если он есть) элементов на предыдущий и следующий соответственно на новый элемент
* увеличение переменной, хранящей число количества элементов в списке

Если при использовании списка часто приходится добавлять или удалять элементы в конце списка, то для уменьшения расходов на просмотр всего списка можно ввести второй основной указатель - на последний элемент списка. Это потребует изменения процедур удаления и добавления для отслеживания этого указателя.

Довольно часто используется упорядоченная разновидность линейного списка, в котором 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 перер этим элементом.

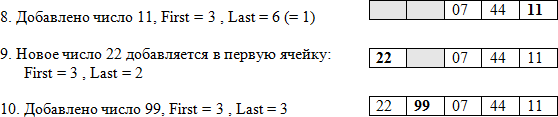
**Очередь** – это линейная структура данных, в которую элементы добавляются с одного конца (конец очереди), а удаляются - с другого (начало очереди). Очередь работает по принципу “элемент, помещенный в очередь первым, извлечен будет тоже первым”. Иногда этот принцип обозначается сокращением **FIFO** (First In – First Out, т.е. первым зашел – первым вышел). Элементами стеков и очередей могут быть любые **однотипные** данные. В простейшем случае – целые числа, чаще всего – записи заранее определенной структуры.

Для реализации статической очереди надо объявить массив и две переменные – указатель начала очереди First и указатель конца очереди Last. Будем считать, что очередь-массив заполняется (растет) от первых элементов массива к последним. Тогда указатель First будет определять первую **занятую** ячейку массива, а указатель Last - первую **свободную** ячейку. Тогда пустую очередь определим как First = Last = 1 (если индексация элементов массива начинается с 1), и при каждом добавлении нового элемента переменная Last увеличивается на 1, а при удалении на 1 увеличивается указатель First. Последовательность операций для массива из пяти элементов показана на следующей схеме:



Рассмотренная выше простейшая реализация очереди-массива имеет один существенный недостаток: освобождающиеся при удалении ячейки в начале массива не используются при последующих добавлениях, и поэтому при интенсивном использовании очереди быстро может возникнуть ситуация, когда указатель Last выходит за пределы массива, тогда как в начале массива есть свободные ячейки.

Для устранения этого недостатка можно использовать два подхода:

* при очередном удалении элемента из начала очереди сдвигать все элементы влево на одну ячейку, что при большом числе элементов в очереди может привести к большим вычислительным затратам
* более эффективно использовать так называемую **кольцевую** очередь, в которой при достижении указателем Last конца массива добавление производится в начало массива:

В этом случае добавление становится невозможным только если в массиве нет ни одной свободной ячейки, как в рассмотренном примере.

Для программной реализации удобно ввести переменную-счетчик числа элементов в очереди, с помощью которой легко отслеживаются состояния пустой и заполненной очереди.

Само **добавление элемента** в очередь выполняется следующим образом:

* проверить возможность добавления (в массиве есть свободные ячейки?)
* добавить элемент в массив по индексу Last
* изменить указатель Last на 1
* если Last выходит за пределы массива, то установить Last в 1
* увеличить счетчик числа элементов в очереди

**Удаление элемента** из очереди:

* проверить возможность удаления (в очереди есть элементы?)
* извлечь элемент из массива по индексу First и выполнить с ним необходимые действия
* увеличить указатель First на 1
* если First выходит за пределы массива, то установить First в 1
* уменьшить счетчик числа элементов в очереди

# Описание основных понятий и механизмов ООП

**Объект** – это структурированная переменная, содержащая всю информацию о некотором физическом предмете или реализуемом в программе понятии.

**Класс** – это описание множества объектов программирования (объектов) и выполняемых над ними действий.

**Метод** – это функция, получающая в качестве обязательного параметра указатель на объект и выполняющие определенные действия с данными объекта программирования.

**Инкапсуляция** – это механизм, который объединяет данные и код, манипулирующий с этими данными, а также защищает и то, и другое от внешнего вмешательства или неправильного использования. В ООП код и данные могут быть объединены вместе (в так называемый «черный ящик») при создании объекта.

Внутри объекта коды и данные могут быть закрытыми или открытыми.

Закрытые коды или данные доступны только для других частей того же самого объекта и, соответственно, недоступны для тех частей программы, которые существуют вне объекта.

Открытые коды и данные, напротив, доступны для всех частей программы, в том числе и для других частей того же самого объекта.

**Наследование**. Новый, или производный класс может быть определен на основе уже имеющегося, или базового класса.

При этом новый класс сохраняет все свойства старого: данные объекта базового класса включаются в данные объекта производного, а методы базового класса могут быть вызваны для объекта производного класса,

причем они будут выполняться над данными включенного в него объекта базового класса

Иначе говоря, новый класс наследует как данные старого класса, так и методы их обработки.

Если объект наследует свои свойства от одного родителя, то говорят об одиночном наследовании. Если объект наследует данные и методы от нескольких базовых классов, то говорят о множественном наследовании.

**Полиморфизм** – это свойство, которое позволяет один и тот же идентификатор (одно и то же имя) использовать для решения двух и более схожих, но технически разных задач.

Целью полиморфизма, применительно к ООП, является использование одного имени для задания действий, общих для ряда классов объектов. Такой полиморфизм основывается на возможности включения в данные объекта также и информации о методах их обработки (в виде указателей на функции).

Будучи доступным в некоторой точке программы, объект , даже при отсутствии полной информации о его типе, всегда может корректно вызвать свойственные ему методы.

# Описание всех разработанных классов (объектная модель)

Классы для информационных объектов:

Bank – Класс банка. Приватные свойства:

1. string name – название банка задается при инициализации.
2. BankDepartment\* pHead – указатель на заголовочный элемент списка отделов банка.

Методы:

1. Bank(string \_name) – конструктор класса.
2. ~Bank() – деструктор класса. При удалении банка все его отделы удаляются из памяти.
3. string getName() – возвращает название банка.
4. void addDepartment() – запрашивает данные у пользователя и добавляет новый отдел в банк.
5. void delDepartment() – запрашивает данные у пользователя и удаляет указанный отдел из банка.
6. BankDepartment\* findDepartment(string \_name) – возвращает указатель на найденный в списке отдел в банке.
7. void addEmployeeIntoDep() – запрашивает данные у пользователя и вызывает метод добавления сотрудника в соответствующий отдел.
8. void delEmployeeFromDep()– запрашивает данные у пользователя и вызывает метод удаления сотрудника из соответствующего отдела.
9. void showAllStruct() – выводит всю информацию о банке в консоль, т.е. всю структуру.
10. void saveToFile() – сохраняет всю структуру в файл.
11. void loadFromFile() – загружает сохраненную структуру из файла(если есть).

BankDepartment – Класс отдела банка. Приватные свойства:

1. string name –название отдела банка. Задается при инициализации.
2. int size – размер массива-очереди.
3. int count – количество добавленных элементов в массиве-очереди.
4. int first – индекс первой заполненной ячейки массива-очереди.
5. int last – индекс первой свободной ячейки массив-очереди.
6. BankEmployee\* queue – указатель на массив-очередь. Размер задается пользователем при добавлении отдела в банк.
7. BankDepartment\* next – указатель на следующий отдел банка в списке. Методы:
8. BankDepartment(string \_name, int \_size, BankEmployee\* \_queue) – конструктор класса.
9. ~BankDepartment() – деструктор класса. При удалении отдела банка, удаляются все его сотрудники из памяти.
10. string getName() – возвращает название отдела.
11. bool isFull() – возвращает истинное значение, если очередь полная.
12. void addEmployee(string \_lastName, string \_post) – добавлят сотрудника в отдел банка.
13. void delEmployee() – удаляет первого сотрудника из отдела банака.
14. void showEmployee() – выводит очередь сотрудников в консоль.
15. int findEmployee(string \_lastName) – возвращает индекс указанного сотрудника в массиве-очереди.
16. void setNext(BankDepartment\* \_next) – устанавливает указатель на следующий элемент в списке отделов банка.
17. BankDepartment\* getNext() – возвращает указатель на следующий элемент в списке отделов банка.
18. void saveToFile(ofstream\* out) – записывает в файл сотрудников текущего отдела.

BankEmployee – Класс сотрудника банка. Приватные свойства:

1. string lastName – фамилия сотрудника.
2. string post – должность сотрудника. Методы:
3. BankEmployee() – конструктор класса.
4. string getLastName() – возвращает фамилию сотрудника.
5. void setLastName(string \_lastName) – устанавливает фамилию сотрудника.
6. string getPost() – возвращает должность сотрудника.
7. void setPost(string \_post) – устанавливает должность сотрудника. Курсовик 10.10 – модуль для взаимодействия пользователя со структурой.

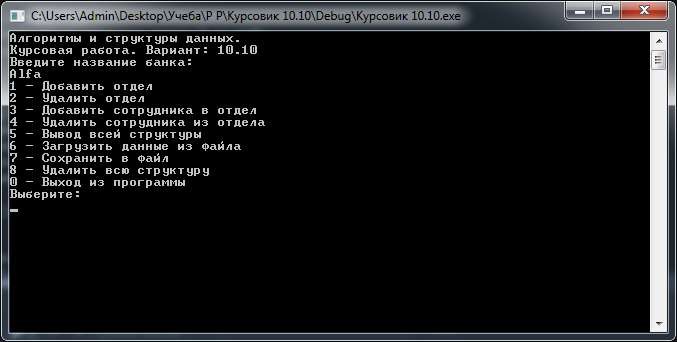
# Описание демонстрационного модуля с характеристикой использованных стандартных компонентов

Демонстрационный модуль реализован в виде консольной программы, которая позволяет пользователю взаимодействовать со структурой.

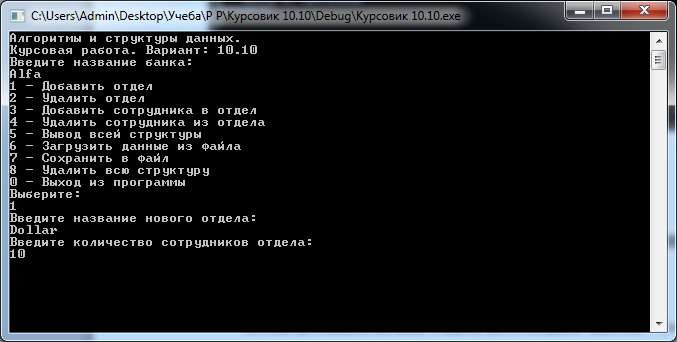
После запуска программы, необходимо ввести название банка:



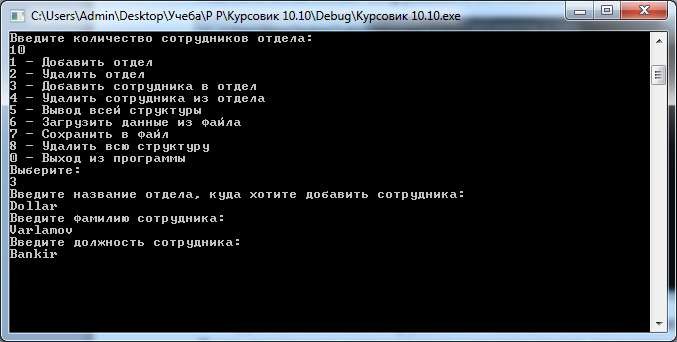
Далее Вы увидите меню, для взаимодействия:



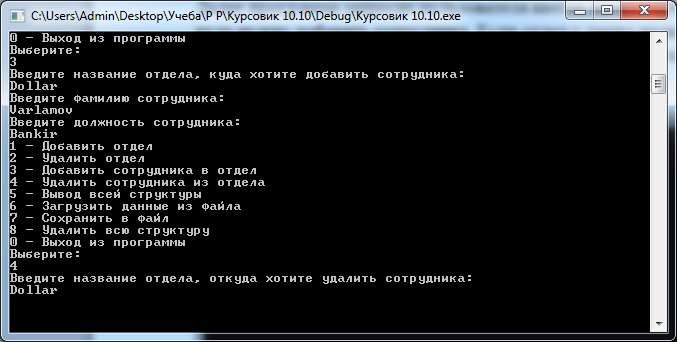
Чтобы добавить новый отдел, необходимо ввести «1». После этого программа запросит пользователя вести название нового отдела и количество сотрудников последовательно:



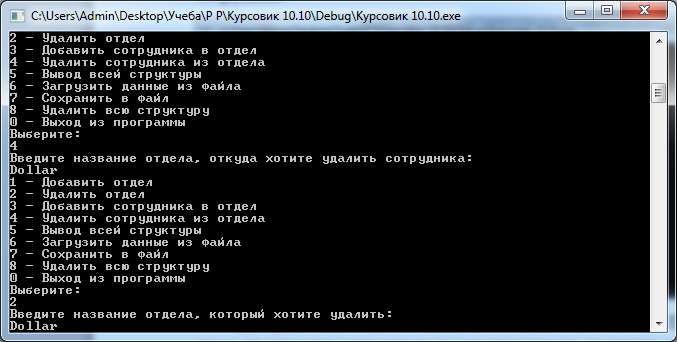
Для добавления сотрудника в отдел, необходимо ввести «3». Далее программа запросит пользователя ввести название отдела, куда нужно добавить сотрудника. Если отдел с таким названием существует, то программа запросит у пользователя фамилию и должность сотрудника последовательно:



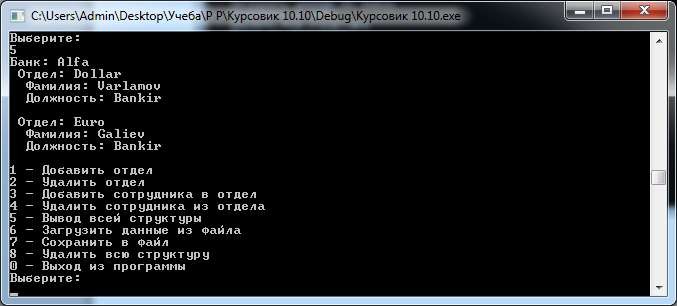
Для удаления сотрудника из отдела, необходимо ввести «4». Далее программа запросит пользователя ввести название отдела, откуда нужно удалить сотрудника. Если отдел с таким названием существует, то из этого отдела будет удален первый в очереди сотрудник:



Для удаления отдела банка, необходимо ввести «2». После этого программа запросит пользователя ввести название отдела, который нужно удалить из банка. Если отдел с таким названием существует, он будет удален:

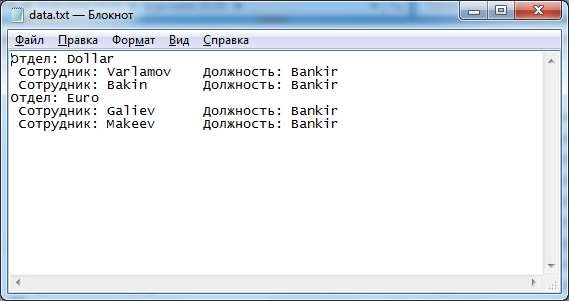


Чтобы вывести всю информацию о банке, необходимо ввести «5»:



Для загрузки данных из текстового файла, необходимо ввести «6».

Данные будут успешно загружены, если структура файла не разрушена:



Для сохранения данных, необходимо ввести «7». Данные будут сохранены в структуре, которую можно увидеть на рисунке выше.

Для удаления всей структуры, необходимо ввести «8». После этого программа вновь запросит ввести название банка. Банк в свою очередь будет пустой.

Чтобы выйти из программы, необходимо ввести «9». Программа удалит всю структуру из памяти без сохранения в файл и завершит работу программы.

# Описание структуры проекта

Структура проекта:

1. Исходные файлы:
   1. Bank.cpp – реализация класса Bank
   2. BankDepartment.cpp – реализация класса BankDepartment
   3. BankEmployee.cpp – реализация класса BankEmployee
   4. Курсовик 10.10.cpp – модуль взаимодействия с пользователем
2. Файлы заголовков:
   1. Bank.h – заголовочный файла класса Bank
   2. BankDepartment.h – заголовочный файла класса BankDepartment
   3. BankEmployee.h– заголовочный файла класса BankEmployee

# Листинги всех подпрограмм с краткими комментариями

Bank.h – заголовочный файла класса Bank:

#pragma once

#include "BankDepartment.h"

class Bank

{

private:

string name; BankDepartment\* pHead;

public:

};

Bank(string \_name);

~Bank();

string getName();

void addDepartment(); void delDepartment();

BankDepartment\* findDepartment(string \_name); void addEmployeeIntoDep();

void delEmployeeFromDep(); void showAllStruct();

void saveToFile(); void loadFromFile();

BankDepartment.h – заголовочный файла класса BankDepartment:

#pragma once

#include "BankEmployee.h" #include <iostream>

using namespace std;

class BankDepartment

{

private:

string name; int size;

int count; int first, last;

BankEmployee\* queue; BankDepartment\* next;

public:

BankDepartment(string \_name, int

void showEmployees();

int findEmployee(string \_lastName);

\_size, BankEmployee\* \_queue);

~BankDepartment(); string getName(); bool isFull();

void addEmployee(string \_lastName, string \_post);

void delEmployee();

\_next);

};

void setNext(BankDepartment\*

BankDepartment\* getNext(); void saveToFile(ofstream\* out);

BankEmployee.h – заголовочный файл класса BankEmployee:

#pragma once #include <iostream> using namespace std;

class BankEmployee

{

private:

string lastName;

public:

};

string post;

BankEmployee(); string getLastName();

void setLastName(string \_lastName); string getPost();

void setPost(string \_post);

Bank.cpp – реализация класса Bank:

#include "Bank.h" #include <string> #include <fstream>

Bank::Bank(string \_name) { name = \_name;

pHead = new BankDepartment("", 0, nullptr);

pHead->setNext(nullptr);

}

}

BankDepartment\* Bank::findDepartment(string \_name) {

BankDepartment\* pCurrent = pHead->getNext();

while (pCurrent != nullptr) {

if (pCurrent->getName() ==

\_name)

return pCurrent; pCurrent = pCurrent-

string Bank::getName() { return name;

>getNext();

}

return nullptr;

}

cout << "Некорректный ввод.

Попробуйте снова: ";

void Bank::addDepartment() { BankDepartment\* temp; BankDepartment\* pCurrent; BankDepartment\* prev; BankEmployee\* queue;

continue;

}

} while

string var; string name; string size; int size\_;

cout << "Введите название нового отдела: " << endl;

cin >> name;

if (findDepartment(name) != nullptr)

{

cout << "Отдел с таким названием уже существует!" << endl;

return;

}

else {

cout << "Введите количество сотрудников отдела: " <<

(!(size.find\_first\_not\_of("0123456789", 0)

== string::npos));

try {

size\_ =

stoi(size);

break;

}

catch

(out\_of\_range) {

cout << "Слишком большое число! Попробуйте снова..." << endl;

continue;

}

}

queue = new

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| endl; |  |  |  | (std::nothrow) BankEmployee[size\_]; |
|  | while (true) { |  |  | if (queue == NULL) { |
|  | while (true) { |  |  | cout << "Не |
|  | do |  |  | удалось выделить память... Попробуйте |
|  | { |  |  | снова..." << endl; |
|  |  | cin | >> | continue; |
| size; |  |  |  | } |
|  |  | if |  | else { |

(!(size.find\_first\_not\_of("0123456789", 0)

== string::npos)) {

temp = new BankDepartment(name, size\_, queue);

break;

}

}

findDepartment(name); nullptr)

pCurrent = if (pCurrent ==

cout <<

nullptr) {

if (pHead->getNext() ==

pHead-

"Отдела с таким названием нет. Попробуйте снова: " << endl;

} while (pCurrent ==

>setNext(temp);

>setNext(nullptr);

}

temp-

nullptr);

if (var == "2") {

temp-

else {

cout << "Выберите

>setNext(pCurrent->getNext());

pCurrent-

вариант добавления нового элемента:" << endl;

cout << "1 - до заданного элемента;" << endl;

cout << "2 - после заданного элемента;" << endl;

do

>setNext(temp);

pHead->getNext();

}

else if (var == "1") {

prev = pHead; pCurrent =

while

var != "2")

{

cin >> var;

if (var != "1" &

cout <<

(pCurrent != nullptr) {

if

(pCurrent->getName() == name)

break;

"Некорректный ввод! Попробуйте снова..." << endl;

} while (var != "1" &

pCurrent;

prev =

var != "2");

cout << "Введите

pCurrent = pCurrent->getNext();

}

temp-

название отдела:" << endl;

do {

cin >> name;

>setNext(prev->getNext());

>setNext(temp);

prev-

}

} endl;

}

}

}

else

cout << "Отдел удален..." <<

cout << "Отдел с таким

void Bank::delDepartment() { BankDepartment\* prev; BankDepartment\* pCurrent; string name;

if (pHead->getNext() == nullptr) { cout << "Нет отделов" <<

endl;

return;

}

названием не найден..." << endl;

}

void Bank::addEmployeeIntoDep() { BankDepartment\* pCurrent; string name;

string lastName; string post;

if (pHead->getNext() == nullptr) { cout << "Нет отделов" <<

cout << "Введите название отдела, который хотите удалить: " << endl;

cin >> name;

endl;

}

return;

name)

prev = pHead;

pCurrent = pHead->getNext(); while (pCurrent != nullptr) {

if (pCurrent->getName() ==

break; prev = pCurrent;

pCurrent = pCurrent-

cout << "Введите название отдела, куда хотите добавить сотрудника: " << endl;

cin >> name;

pCurrent = findDepartment(name);

if (pCurrent != nullptr && pCurrent-

>isFull())

>getNext();

}

if (pCurrent != nullptr) {

prev->setNext(pCurrent-

>getNext());

delete pCurrent;

{

endl;

}

cout << "Отдел полон" << return;

if (pCurrent != nullptr) {

cout << "Введите фамилию сотрудника:" << endl;

cin >> lastName;

if (pCurrent-

>findEmployee(lastName) == -1) {

cout << "Введите должность сотрудника:" << endl;

cin >> post; pCurrent-

>addEmployee(lastName, post);

}

else

cout << "Сотрудник с такой фамилией уже существует" << endl;

}

else

cout << "Отдел с таким названием не найден" << endl;

}

void Bank::delEmployeeFromDep() { BankDepartment\* pCurrent; string name;

if (pHead->getNext() == nullptr) { cout << "Нет отделов" <<

endl;

return;

}

pCurrent = findDepartment(name); if (pCurrent != nullptr) {

pCurrent->delEmployee();

}

else {

cout << "Отдел с таким названием не найден" << endl;

}

}

void Bank::showAllStruct() { BankDepartment\* pCurrent;

if (pHead->getNext() == nullptr) { cout << "Нет отделов" <<

endl;

return;

}

pCurrent = pHead->getNext();

cout << "Банк: " + this->getName()

<< endl;

while (pCurrent != nullptr) {

cout << " Отдел: " + pCurrent->getName() << endl;

pCurrent->showEmployees(); pCurrent = pCurrent-

>getNext();

}

cout << "Введите название отдела, откуда хотите удалить сотрудника: " << endl;

cin >> name;

}

Bank::~Bank() {

BankDepartment\* pCurrent;

while (pHead->getNext() != nullptr)

{

pCurrent = pHead-

}

void Bank::loadFromFile() {

>getNext();

>getNext());

}

pHead->setNext(pCurrent- delete pCurrent;

ifstream in("data.txt"); string line;

string dep = "Отдел:";

string employee = "Сотрудник:"; string post = "Должность:";

delete pHead;

}

string depName;

string employeeLastName; string employeePost;

void Bank::saveToFile() {

ofstream\* out = new ofstream(); BankDepartment\* pCurrent; out->open("data.txt");

nullptr;

BankDepartment\* tempDep =

|  |  |  |  |
| --- | --- | --- | --- |
| if (out->is\_open()) |  |  | if (in.is\_open()) |
| { |  |  | { |
| pCurrent | = | pHead- | while (getline(in, line)) |
| >getNext(); |  |  | { |

while (pCurrent != nullptr)

{

\*out << "Отдел: " << pCurrent->getName() << endl;

!= string::npos)

if (line.find(employee)

{

if

>saveToFile(out);

>getNext();

pCurrent-

pCurrent = pCurrent-

(line.find(post) != string::npos) {

if

(tempDep != nullptr) {

}

out->close();

employeeLastName = line;

}

else

cout << "Не удалось

employeePost = line;

открыть файл..." << endl; try {

employeeLastName.erase(0, employee.size() + 1);

employeeLastName.erase(employee LastName.find("\t"), employeeLastName.size());

in.close();

}

}

else {

}

cout <<

employeeLastName.erase(0, 1);

employeePost.erase(0, employeePost.find(post) + post.size() + 1);

if (tempDep-

>findEmployee(employeeLastName) == -1)

{

"Структура файла разрушена!" << endl;

cout << "Возможно, частично данные были загружены." << endl;

in.close();

}

}

tempDep-

>addEmployee(employeeLastName, employeePost);

!= string::npos)

else if ((line.find(dep))

{

}

}

catch (exception ex) {

dep.size() + 1); line;

findDepartment(depName);

== nullptr) {

line.erase(0, depName =

tempDep = if (tempDep

cout << "Структура файла разрушена!" << endl;

cout << "Возможно, частично данные были загружены." << endl;

tempDep = new BankDepartment(depName, 10, new BankEmployee[10]);

tempDep->setNext(pHead-

>getNext());

>setNext(tempDep);

}

}

else {

pHead-

cout <<

"Возможно, частично данные были загружены." << endl;

in.close();

}

cout <<

"Структура файла разрушена!" << endl;

}

in.close();

}

}

BankDepartment.cpp – реализация класса BankDepartment:

#include "BankDepartment.h" #include <fstream>

BankDepartment::BankDepartment(string

\_name, int \_size, BankEmployee\* \_queue) { name = \_name;

size = \_size; queue = \_queue; first = 0;

last = 0;

count = 0; next = nullptr;

}

string BankDepartment::getName() { return name;

}

bool BankDepartment::isFull() { if (count < size)

return false;

else

return true;

}

void BankDepartment::addEmployee(string

\_lastName, string \_post) { if (!isFull()) {

queue[last].setLastName(\_lastName)

;

queue[last].setPost(\_post); last++;

count++;

if (last >= size)

last = 0;

}

}

void BankDepartment::delEmployee() { if (count > 0) {

queue[first].setLastName(""); queue[first].setPost(""); first++;

count--;

if (first >= size)

first = 0;

}

}

}

void BankDepartment::showEmployees() { }

int index = first; int c = 0;

while (c < count) {

cout << " Фамилия: " + queue[index].getLastName() << endl;

cout << " Должность: " + queue[index].getPost() << endl;

cout << endl;

int BankDepartment::findEmployee(string

\_lastName) {

int index = first; int c = 0;

while (c < count) {

if (queue[index].getLastName() ==

\_lastName)

return index;

|  |  |  |
| --- | --- | --- |
| index++; |  | index++; |
| c++; |  | c++; |
| if (index >= size) |  | if (index >= size) |
| index = 0; |  | index = 0; |
| } | } |  |

}

void BankDepartment::saveToFile(ofstream\*

}

void

return -1;

out) {

int index = first; int c = 0;

while (c < count) {

\*out << " Сотрудник: " +

BankDepartment::setNext(BankDepartment

\* \_next) {

next = \_next;

}

queue[index].getLastName() << "\t" << "Должность: " + queue[index].getPost() << endl;

index++; c++;

if (index >= size)

index = 0;

BankDepartment\* BankDepartment::getNext() {

return next;

}

BankDepartment::~BankDepartment() { delete[] queue;

}

BankEmployee.cpp – реализация класса BankEmployee:

#include "BankEmployee.h"

BankEmployee::BankEmployee() { lastName = "";

post = "";

}

string BankEmployee::getLastName() { return lastName;

}

string BankEmployee::getPost() {

return post;

}

void BankEmployee::setLastName(string

\_lastName) {

lastName = \_lastName;

}

void BankEmployee::setPost(string \_post){ post = \_post;

}

Курсовик 10.10.cpp – модуль взаимодействия с пользователем:

#include "Bank.h" #include <iostream> #include <string> using namespace std;

bool isInteger(string& s); void print();

Bank\* bank;

int main()

{

setlocale(LC\_ALL, "");

cout << "Алгоритмы и структуры данных." << endl;

cout << "Курсовая работа.

Вариант: 10.10 " << endl; for (;;)

{

string name; string var;

if (bank == nullptr) {

cout << "Введите название банка: " << endl;

cin >> name;

bank = new

Bank(name);

}

print(); do

{

cin >> var;

if (!isInteger(var))

{

cout <<

"Некорректный ввод! Попробуйте снова..." << endl;

continue;

}

} while (!isInteger(var)); switch (stoi(var))

{

case 1:

>addDepartment();

case 2:

>delDepartment();

case 3:

bank- break; bank- break;

bank-

cout << "Неверный ввод. Повторите еще раз..." << endl;

break;

}

}

}

void print()

{

>addEmployeeIntoDep();

break;

case 4:

bank-

>delEmployeeFromDep();

endl; endl;

cout << "1 - Добавить отдел" <<

cout << "2 - Удалить отдел" << cout << "3 - Добавить сотрудника в

case 5:

>showAllStruct();

case 6:

>loadFromFile();

case 7:

case 8:

case 0:

break; bank- break; bank- break;

bank->saveToFile(); break;

delete bank; bank = nullptr; break;

delete bank; return 0;

отдел" << endl;

cout << "4 - Удалить сотрудника из отдела" << endl;

cout << "5 - Вывод всей структуры" << endl;

cout << "6 - Загрузить данные из файла" << endl;

cout << "7 - Сохранить в файл" <<

endl;

cout << "8 - Удалить всю структуру" << endl;

cout << "0 - Выход из программы"

<< endl;

cout << "Выберите: " << endl;

}

bool isInteger(string& s)

{

return s.find\_first\_not\_of("0123456789", 0) ==

default: string::npos;

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.