Министерство образования РФ

Пермский государственный технический университет

Кафедра ИТАС

ЛАБОРАТОРНАЯ РАБОТА № 10

ПО ИНФОРМАТИКЕ ЗА I СЕМЕСТР

Вариант 9

Выполнил студент:

Главатских Максим Николаевич

Группа РИС-20-1бз

Шифр 20-ЭТФ-659

Кафедра ИТАС:

Полякова Ольга Андреевна

ПЕРМЬ, 2020

# Лабораторная работа №10 "Хранение данных на внешних носителях”

**Вариант №9**

**Задача:**

1. Написать приложение для работы с простой базой данных, хранящей информацию об объекте на внешнем носителе. Приложение должно выполнять следующие функции:
2. Создание базы данных, содержащей записи указанного формата.
3. Просмотр базы данных.
4. Удаление элементов из базы данных (по ключу/ по номеру).
5. Корректировка элементов в базе данных (по ключу / по номеру).
6. Добавление элементов в базу данных (в начало / в конец/ с заданным номером).
7. Выполнение задания, указанного в варианте.

**Вариант задания:**

9. Типизированный файл, имеет следующую структуру: Автор, Название, Год издания, Издательство. Программа должна

* 1. Добавлять, удалять, корректировать, позволять просматривать записи файла.
  2. Выдавать по запросу пользователя:
     1. список литературы, указанного пользователем автора;
     2. список литературы, изданной в указанный пользователем период.

**Текст программы**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <iomanip>

#include <cstring>

#include <cstdlib>

#include <string>

#include <stdio.h>

#include <vector>

#include <fstream>

using namespace std;

struct Info

{

Info(string ath, string bk\_nm, int pb\_yr, string pb\_nm) : author(ath), book\_name(bk\_nm),

public\_year(pb\_yr), publication\_name(pb\_nm)

{}

// Copy constructor

Info(Info const& info) : author(info.author), book\_name(info.book\_name), public\_year(info.public\_year),

publication\_name(info.publication\_name)

{

cout << "Copy constructor" << endl;

}

// Assgn operator

Info& operator= (const Info& info)

{

if (&info != this) {

// Выполняем копирование значений

this->author = info.author;

this->book\_name = info.book\_name;

this->public\_year = info.public\_year;

this->publication\_name = info.publication\_name;

}

// Возвращаем текущий объект, чтобы иметь возможность связать в цепочку выполнение нескольких операций присваивания

return \*this;

}

operator string() {

return author + string(" ") + book\_name + string(" ") + to\_string(public\_year) + string(" ") + publication\_name;

}

string author;

string book\_name;

int public\_year;

string publication\_name;

};

// Out operator - for using cout << <Info obj>;

std::ostream& operator<< (std::ostream& out, const Info& info)

{

// Поскольку operator<< является другом класса Point, то мы имеем прямой доступ к членам Point

out << info.author << " " << info.book\_name << " " << info.public\_year << " " << info.publication\_name;

return out;

}

struct Item

{

Item(int k, Info const& info\_, Item\* n = NULL)

:key(k), info(info\_), next(n), length(sizeof(Info) + 1)

{}

int key;//ключ

Info info;//информация

Item\* next;//следующий элемент списка

int length;//длина информации

int shift;//смещение в файле (байт)

};

//проверяет принадлежит ли ключ списку

bool Belong(Item\*& list, int key)

{

Item\* temp(list);

while (temp && temp->key != key)//пока есть элементы в списке и ключ текущего элемнта не равен ключу что мы ищем

temp = temp->next;

return (temp ? true : false);//возвращаем true если temp!=NULL, те если ключ найден в списке

}

//добавляет в список элемент

void Add(Item \* &list, int key, Info & val)

{

if (Belong(list, key))//проверяем нет ли у нас элемента уже с таким ключом

{

cerr << "Element with key = " << key << " already exist\n";

return;

}

if (!list)//если список пуст то добавим в голову списка элемент

list = new Item(key, val, NULL);

else//иначе найдём хвост и добавим после него элемент

{

Item\* temp(list);

while (temp->next)

temp = temp->next;

temp->next = new Item(key, val, NULL);

}

}

void Insert(Item \* &list, int pos, int key, Info & val) {

Item\* cur = list;

int i = 0;

Item\* new\_item = new Item(key, val, NULL);

if (pos == 0) {

new\_item->next = list;

list = new\_item;

return;

}

while (cur) {

if (i == (pos - 1)) {

Item\* time\_next = cur->next;

cur->next = new\_item;

if (time\_next != NULL)

new\_item->next = time\_next->next;

}

i++;

cur = cur->next;

}

}

void Update(Item \* &list, int key, Info & val) {

Item\* cur(list);

while (cur) {

if (cur->key == key) {

cur->info = val;

break;

}

cur = cur->next;

}

}

//удаление элемента с заданным ключом

bool Delete(Item \* &list, int key)

{

Item\* temp(list);

while (temp && temp->key != key)//ищем элемент с таким ключом как нас просят

temp = temp->next;

if (temp)//если елемент с таким ключом существует

{

//если елемент что нужно удалять это голова списка

if (temp == list)

{

Item\* toDel(list);

list = list->next;

delete toDel;

return true;

}

//иначе элемент где-то в середине или в конце

else

{

Item\* toDel(temp);

Item\* newElem(list);

//найдём элемент перед тем что собрались удалять

while (newElem->next != temp)

newElem = newElem->next;

//уберём удаляемый элемент из списка

newElem->next = temp->next;

delete toDel;//удалим элемент

return true;

}

}

else//иначе элемента нет. Уведомляем пользователя об ошибке

{

cerr << "Can't delete element with key = " << key << endl;

return false;

}

}

//поиск элемента по ключу

Item\* Find(Item \* &list, int key)

{

Item\* result;//результат. Этот указатель мы и вернём пользователю

Item\* temp(list);

while (temp && temp->key != key)//цикл происходит пока не будет найдено совпадение ключа (или пока не кончится список)

temp = temp->next;

if (temp)//если список не кончился - мы нашли ключ, скопируем этот элемент и вернём его как результа

return (result = new Item(temp->key, temp->info, NULL));

else//иначе список закончился быстрее чем мы нашли нужный элемент. Уведомим пользователя об ошибке и вернём указатель на NULL

cerr << "Can't find element with key = " << key << endl;

return (result = NULL);

}

//Удаление всего списка, необходимо чтобы можно было освободить память

void DeleteList(Item \* &list)

{

Item\* toDel(list);

while (list)//пока есть что удалять - удаляем элементы

{

list = list->next;

delete toDel;

toDel = list;

}

}

//вывод в поток списка

ostream& operator<<(ostream & os, const Item \* print)

{

const Item\* temp(print);

if (!temp)//если элементов в списке нет (list = NULL) то не будем ничего пытаться печатать

os << "List is empty.";

else//иначе распечатаем построчно ключи и информационную часть элементов

while (temp)

os << temp->key << " -> " << temp->info << '\n', temp = temp->next;

return (os << endl);//вернём адрес потока чтобы пользователь мог туда ещё что-то засунуть (позволяет делать так cout<<list1<<list2<<list3)

}

// Работа с файлами

/\*

1. Создание базы данных, содержащей записи указанного формата.

2. Просмотр базы данных.

3. Удаление элементов из базы данных (по ключу/ по номеру).

4. Корректировка элементов в базе данных (по ключу / по номеру).

5. Добавление элементов в базу данных (в начало / в конец/ с заданным номером).

6. Выполнение задания, указанного в варианте.

1. Типизированный файл, имеет следующую структуру: Автор, Название, Год издания, Издательство. Программа должна

a. Добавлять, удалять, корректировать, позволять просматривать записи файла.

b. Выдавать по запросу пользователя:

i. список литературы, указанного пользователем автора;

ii. список литературы, изданной в указанный пользователем период.

\*/

struct Database {

Database(string fileName) : fileName(fileName), manager(NULL) {}

// Применяем перезапись файла согласно менджеру объектов

void save() {

Item\* cur = manager;

ofstream out;

out.open(fileName);

while (cur != NULL) {

out << string(cur->info) << endl;

cur = cur->next;

}

out.close();

}

void printDatabse() {

cout << "Databse:" << endl << manager << endl;

}

void add(int key, Info& info) {

Add(manager, key, info);

}

void del(int key) {

Delete(manager, key);

}

void update(int key, Info& info) {

Update(manager, key, info);

}

void insert(int pos, int key, Info& info) {

Insert(manager, pos, key, info);

}

vector<string> get\_literature\_author(string author) {

vector<string> literature;

Item\* cur = manager;

while (cur != NULL) {

if (cur->info.author == author)

literature.push\_back(cur->info.book\_name);

cur = cur->next;

}

return literature;

}

vector<string> get\_literature\_year(int year) {

vector<string> literature;

Item\* cur = manager;

while (cur != NULL) {

if (cur->info.public\_year == year)

literature.push\_back(cur->info.book\_name);

cur = cur->next;

}

return literature;

}

string fileName;

Item\* manager;

};

int main()

{

Database db("db.txt");

Info a = Info("Толстой", "Война и мир", 1843, "Хзнает");

Info b = Info("Gogol", "Shinel", 1823, "Хзнает");

db.add(1, a);

db.add(2, b);

db.printDatabse();

db.del(2);

db.printDatabse();

Info c = Info("Pushking", "Ne znayu", 1863, "Хзнает");

db.update(1, c);

db.printDatabse();

db.save();

Info d = Info("Nikto", "Nichego", 1893, "Хзнает");

db.insert(0, 3, d);

db.insert(2, 4, d);

db.printDatabse();

db.save();

// Водим первые элементы списков литературы -- ПРОСТО ПРИМЕР РАБОТЫ ФУНКЦИОНАЛА

cout << db.get\_literature\_author("Nikto")[0] << endl;

cout << db.get\_literature\_year(1863)[0] << endl;

}

