Пермский Национальный Исследовательский  
Политехнический Университет

**Лабораторная работа № 1**

Информатика

за 2 семестр

Вариант № 1

Выполнил:

Студент группы РИС 20-1-бз

КургановН.В.

20-ЭТФ-631

Проверила:

Доцент кафедры ИТАС

Полякова О.А.

2021

**Лабораторная работа № 11**

# Поиск данных с помощью хэш-таблиц

### Цель работы:

1. Изучить построение функции хеширования и алгоритмов хеширования данных
2. Научиться разрабатывать алгоритмы открытого и закрытого хеширования при решении задач на языке C++.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Вариант** | **Данные** | **Ключ (string)** | **Хэш-функция** | **Метод рехеширования** |
| 1 | ФИО, группа, рейтинг | ФИО | H(k)=k mod M | Метод цепочек |

#include <iostream>

#include <conio.h>

#include <fstream>

using namespace std;

struct Student //структура ученика

{

string FIO;//ФИО

int group;//группа

int rate;//место в рейтинге

};

struct LinkedHashEntry

{

int key;

int value;

LinkedHashEntry\* next;

};

//функция генерации строки, на входе получаем длину строки, на выходе получаем строку

string RandomStr(int n) {

char\* arr\_ru = new char[26]{ 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z' };

char\* arr\_RU = new char[26]{ 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z' };

string str = "";

str += arr\_RU[rand() % 25];

for (int i = 1; i < n; i++) {

str += arr\_ru[rand() % 25];

}

return str;

}

//функция хеширования на вход подаётся 2 числа, получаем модуль 1-ого от 2-ого

int func(int x, int n) {

return x % n;

}

//функция перевода строки в число, подаётся строка выводится число

int strToint(string g) {

int res = 0;

const char\* cstr = g.c\_str();//преобразуем в char\*

for (int i = 0; i < strlen(cstr); i++)

res += abs(cstr[i]);

return res;

}

const char\* cureFileName;//файл

const char\* realFileName = "dataBase.bd";//ссылка на основной файл с данными

int cureMassLen = 0;//текущая длина массива

//подсчитать коллизию по списку

void countColision(LinkedHashEntry\* mas)

{

int\* numbers = new int[cureMassLen];

//преобразуем список в массив ключей

for (int i = 0; i < cureMassLen; i++) {

numbers[i] = mas->key;

mas = mas->next;

}

//сортируем массив ключей

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

{

for (int j = i + 1; j < cureMassLen; j++)

{

if (numbers[i] > numbers[j])

{

int tmp = numbers[i];

numbers[i] = numbers[j];

numbers[j] = tmp;

}

}

}

int n = 0;

//считаем коллизию (количество повторяющихся ключей)

for (int i = 0; i < cureMassLen; )

{

int j = 0;

//считаем количество повторений

while (++i < cureMassLen && numbers[i] == numbers[i - 1]) ++j;

if (j != 0) n += 1;

}

cout << "Кол-во коллизий равно " << n << "\n";

}

//создаём хештаблицу по массиву и размеру массива

LinkedHashEntry\* CreateLinkedHashEntry(Student\* mas, int n)

{

LinkedHashEntry\* beg;//указатель на первый элемент

LinkedHashEntry\* p, \* r;//вспомогательные указатели

beg = new(LinkedHashEntry);//выделяем память под первый элемент

beg->key = func(strToint(mas[0].FIO),n);//считаем хеш функцию для строки

beg->value = 0;//вводим значение информационного поля

beg->next = 0;//обнуляем адресное поле

//ставим на этот элемент указатель p (последний элемент)

p = beg;

for (int i = 1; i < n; i++)

{

r = new(LinkedHashEntry);//создаем новый элемент

r->key = func(strToint(mas[i].FIO),n);

r->value = i;

r->next = 0;

p->next = r;//связываем p и r

//ставим на r указатель p (последний элемент)

p = r;

}

countColision(beg);

return beg;//возвращаем beg как результат функции

}

//Функция сохранения данных в файл на вход подаётся путь к файлу, массив структуры, длина массива,

//на выходе создаётся файл с сохранёнными данными массива

void save(const char\* filename, struct Student\* mas, int n)

{

fstream f(filename, ios::out);//двунаправленный файловый поток

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

{

//добавляем данные в файл

f << mas[i].FIO; f << "\n";

f << mas[i].group; f << "\n";

f << mas[i].rate; f << "\n";

}

f.close();//закрытие потока

}

//Функция изменения данных по ключу вход подаётся массив структуры, данные на изменение

//на выходи выдаём массив структуры

Student\* change\_elem(Student\* b, int day, int avgT)

{

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

{

if (b[i].group == day)//если i запись дня равна искомому то меняем

{

b[i].rate = avgT;

}

}

save(cureFileName, b, cureMassLen);//сохраняем массив в текущий файл

return b;

}

//Функция добавления данных по индексу вход подаётся массив структуры, добавляемые данные и индекс

//на выходи выдаём массив структуры

Student\* add\_elem(const char\* filename, Student\* b, string FIO ,int day, int avgT, int n)

{

int i = 0;

Student\* a = new Student[cureMassLen + 1];

for (; i < cureMassLen + 1; i++)

{

if (i == n)//если номер равен искомому то заносим новые значения в массив a

{

a[i].FIO= FIO;

a[i].group = day;

a[i].rate = avgT;

}//если текущий индекс меньше искомого переписываем массив b в a

else if (i < n) {

a[i].group = b[i].group;

a[i].rate = b[i].group;

}

else //если текущий индекс больше искомого переписываем массив b в a

{

a[i].group = b[i - 1].group;

a[i].rate = b[i - 1].group;

}

}

cureMassLen++;//увеличиваем текущий размер на 1

save(cureFileName, a, cureMassLen);

return a;

}

//Функция изменения данных по ключу вход подаётся массив структуры, индекс удаляемых данных и изначальный размер массива

//на выходи выдаём массив структуры

Student\* del\_num(const char\* filename, Student\* b, int k, int n)

{

fstream f(filename, ios::out);//вспомогательный файл

//считываем данные из исходного файла в буфер

int i = 0;

if (cureMassLen > 0) {

Student\* a = new Student[cureMassLen - 1];//создаём новый массив в котором на 1 меньше элементов

for (int j = 0; i < cureMassLen-1; i++)

{

if (i != k)//если номер записи не равен k

{

f << b[j].FIO; f << "\n";

f << b[j].group; f << "\n";

f << b[j].rate; f << "\n";

a[i].FIO = b[j].FIO;

a[i].group = b[j].group;

a[i].rate = b[j].rate;

}

j++;

}

f.close();//закрываем исходный файл

cureMassLen--;

if (n >= cureMassLen \* 2)

{

remove(realFileName);//удаляем исходный файл

rename(filename, realFileName);

}//переименовываем временный файл

return a;

}

return new Student[0];

}

//Функция изменения данных по ключу вход подаётся массив структуры, ключ удаляемых данных и изначальный размер массива

//на выходи выдаём массив структуры

Student\* del\_key(const char\* filename, Student\* b, string k, int n)

{

fstream f(filename, ios::out);//вспомогательный файл

//считываем данные из исходного файла в буфер

int i = 0;

Student\* a = new Student[cureMassLen - 1];

for (int j = 0; i < cureMassLen - 1; i++, j++)

{

if (b[i].FIO == k)//если день записи равен k

{

j++; //увеличиваем индекс для основного массива чтобы не переписывать k элемент в новый массив

}

f << b[j].FIO; f << "\n";

f << b[j].group; f << "\n";

f << b[j].rate; f << "\n";

a[i].FIO = b[j].FIO;

a[i].group = b[j].group;

a[i].rate = b[j].rate;

}

f.close();//закрываем исходный файл

cureMassLen--;

if (n >= cureMassLen \* 2) {

remove(realFileName);//удаляем исходный файл

rename(filename, realFileName);

}//переименовываем временный файл

return a;

}

//Функция поиска индексов по ключу, на вход подаётся ключ хеш таблица, таблица данных

//и выдаётся список всех студентов у которых ключ совпадет с введенным

void findIndex(string FIO, LinkedHashEntry\* mas, Student\* mass) {

int key = func(strToint(FIO), cureMassLen);//вычисляем ключ

cout << "\nID\tФИО\tГруппа\tРейтинг\n";

int i = 0;

while(mas) {

if (mas->key == key) {

cout << i + 1 << "\t" << mass[mas->value].FIO << "\t" << mass[mas->value].group << "\t" << mass[mas->value].rate << "\n";

}

i++;

mas = mas->next;

}

}

//Функция загрузки данных, на вход подаётся путь к файлу на выходе получаем массив элементов

Student\* load(const char\* filename)

{

fstream f;

Student a;

Student\* p;

f.open(filename, ios::in);//открываем поток для чтения

if (f) //если файл существует

{

int num = 0;//кол-во элементов

do

{

//достаём данные

f >> a.FIO;

f >> a.group;

f >> a.rate;

if (f.eof())break;

num++;

} while (!f.eof());

f.close();//закрытие потока

f.open(filename, ios::in);

p = new Student[num];

if (num > 0) {

num = 0;

do

{

//достаём данные

f >> a.FIO;

f >> a.group;

f >> a.rate;

if (f.eof())break;

p[num] = a;//записываем в массив

num++;

} while (!f.eof());

}

cureMassLen = num;

f.close();//закрытие потока

return p;

}

return new Student[0];

}

//Функция создания данных, на вход подаётся размер массива на выходе получаем массив элементов

Student\* create(int n = 100) {

Student\* p = new Student[n];

for (int i = 0; i < n; i++) {

p[i].FIO = RandomStr(rand() % 8);

p[i].group = rand() % 11+1;

p[i].rate = rand() % 101 + 1;;

}

return p;

}

//Функция отображения данных, на вход подаётся массив и его размер на выходе получаем список элементов

void show(Student\* dataBase, int n) {

cout << "\nID\tФИО\tГруппа\tРейтинг\n";

for (int i = 0; i < n; i++) {

cout << i+1 << "\t" << dataBase[i].FIO << "\t" << dataBase[i].group << "\t" << dataBase[i].rate << "\n";

}

}

int main()

{

setlocale(LC\_ALL, "Rus");//меняем язык консоли на русский

int n = 0;

Student\* mas = load(realFileName);

LinkedHashEntry\* hashTabel = new LinkedHashEntry;

if (cureMassLen > 0) {

hashTabel = CreateLinkedHashEntry(mas, cureMassLen);

}

while (1) {

cout << "1 - Создать список данных, 2 - Отобразить все данные, 3 - Удалить элемент, 4 - Добавить элемент, 5 - Поиск по ключу, 6 - Отменить, 7 - Сохранить";

cout << "\n";

char c = \_getch();//получаем нажатую кнопку

int reit, group, id,i;

int\* ids;

string fio;

switch (c) {

case '1':

//создаём массив дней

cout << "Введите размер массива: ";

cin >> cureMassLen;

mas = create(cureMassLen);

n = cureMassLen;

cureFileName = realFileName;

hashTabel = CreateLinkedHashEntry(mas, n);

save(cureFileName, mas, n);

break;

case '2':

//выводим массив

if (cureMassLen != 0) {

cout << "\n";

show(mas, cureMassLen);

}

else {

cout << "Нет данных ";

}

break;

case '3':

//удаляем данные

if (cureMassLen != 0) {

cout << "1 - Удалить по ключу, 2 - Удалить по номеру";

cout << "\n";

char c = \_getch();

switch (c)

{

case '1':

//по ключу

cout << "Введите ФИО: ";

cin >> fio;

cureFileName = "temp";

mas = del\_key(cureFileName, mas, fio, n);

cout << "\n";

break;

case '2':

//по номеру

cout << "Введите ID: ";

int num;

cin >> num;

cureFileName = "temp";

mas = del\_num(cureFileName, mas, num, n);

cout << "\n";

break;

default:

break;

}

}

else {

cout << "Нет данных ";

} break;

case '4':

//вставить элемент

cout << "1 - Добавить в начало, 2 - Добавить после номера, 3 - Добавить в конец списка ";

c = \_getch();

switch (c)

{

case '1':

//вставить в начало

cout << "Введите Фамилию Имя Отчество: ";

cin >> fio;

cout << "Группу";

cin >> group;

cout << "Введите температуру: ";

cin >> reit;

mas = add\_elem(cureFileName, mas, fio, group, reit, 0);

hashTabel = CreateLinkedHashEntry(mas, cureMassLen);

break;

case '2':

//вставить после id

cout << "Введите id после которого вставить элемент: ";

cin >> id;

cout << "Введите Фамилию Имя Отчество: ";

cin >> fio;

cout << "Группу";

cin >> group;

cout << "Введите температуру: ";

cin >> reit;

mas = add\_elem(cureFileName, mas, fio, group, reit, id);

hashTabel = CreateLinkedHashEntry(mas, cureMassLen);

break;

case '3':

//вставить в конец

cout << "Введите Фамилию Имя Отчество: ";

cin >> fio;

cout << "Группу";

cin >> group;

cout << "Введите температуру: ";

cin >> reit;

mas = add\_elem(cureFileName, mas, fio, group, reit, cureMassLen);

hashTabel = CreateLinkedHashEntry(mas, cureMassLen);

break;

}

case '5':

cout << "Введите ФИО: ";

cin >> fio;

findIndex(fio, hashTabel,mas);

case '6':

//отменяем изменения загружая основной файл

mas = load(realFileName);

cureFileName = realFileName;

break;

case '7':

//сохраняем изменения и удаляем текущий файл если он существует

save(cureFileName, mas, cureMassLen);

if (cureFileName != realFileName) {

remove(realFileName);

rename(cureFileName, realFileName);

cureFileName = realFileName;

}

break;

case 27:

return 1;

}

}

return 0;

}

# 