

Лабораторная работа №3. Таблицы идентификаторов

Содержание

1 Пример реализации лабораторной работы

2. Задание на лабораторную работу

1. Пример реализации лабораторной работы

Задание

1) Написать программу, которая получает на входе набор идентификаторов, организует таблицу по заданному методу и позволяет осуществить многократный поиск и добавление идентификатора в этой таблице. Список идентификаторов задан в виде текстового файла. Длина идентификаторов ограничена 32 символами.

2) Реализовать две подпрограммы в соответствии с вариантом.

Код программы

```
#include <iostream>
#include <vector>
#include <string>
#include <cstring>
#include <fstream>

using namespace std;

class Map {
public:
    virtual void get(char* id) {}
    virtual void put(char* id, int data) {}
};

class InterpolateMap : public Map {
private:
    struct node {
        char* id;
        int data;
    };
    vector<node> arr;

    int search(char* id) {
        int lo = 0, hi = arr.size() - 1;
        while (lo <= hi && strcmp(id, arr[lo].id) >= 0 && strcmp(id, arr[hi].id) <=
0) {
            if (lo == hi) return lo;
            int pos = lo + (hi - lo) /
                strcmp(arr[hi].id, arr[lo].id) *
```

```

        strcmp(id, arr[lo].id);
        if (pos > arr.size() - 1) pos = arr.size() - 1;
        int cmp = strcmp(arr[pos].id, id);
        if (cmp == 0) return pos;
        if (cmp < 0) lo = pos + 1;
        else hi = pos - 1;
    }
    if (arr.size() && strcmp(arr[arr.size() - 1].id, id) < 0)
        return arr.size();
    return lo;
}

bool exists(char* id, int index) {
    return 0 <= index && index <= (int)(arr.size() - 1) &&
strcmp(arr[index].id, id) == 0;
}

public:
    void get(char* id) {
        int index = search(id);
        if (exists(id, index)) cout << arr[index].data << endl;
        else cout << "undefined" << endl;
    }

    void put(char* id, int data) {
        int index = search(id);
        if (exists(id, index)) {
            arr[index].data = data;
        }
        else {
            node n = { id, data };
            arr.insert(arr.begin() + index, n);
        }
    }
};

```

```

class ChainHashMap : public Map {
private:
    struct node {
        char* id;
        int data;
        int freePtr;
    };

    vector<node> arr;
    int size = 0;
    int freePtr = 0;

    unsigned long hash(char* id) {
        unsigned long hash = 5381;
        while (int c = *id++) hash = ((hash << 5) + hash) + c;
        return hash % arr.size();
    }
};

```

```

    }

    int find(char* id, bool link) {
        int i = 0;
        int index = hash(id);
        while ((long int)(arr[index].id) != 0 && strcmp(arr[index].id, id) != 0 &&
arr[index].freePtr != 0)
            index = arr[index].freePtr;
        if ((long int)(arr[index].id) != 0 && link) {
            arr[index].freePtr = freePtr++;
            index = freePtr - 1;
        }
        return index;
    }

    void resize() {
        vector<node> temp = arr;
        int size = 2 * arr.size() + 1;
        arr.clear();
        for (int i = 0; i < size; i++)
            arr.push_back(node{});
        for (node n : temp)
            if ((long int)(n.id) != 0)
                arr[find(n.id, true)] = node{ n.id, n.data };
    }

public:
    ChainHashMap() {
        arr.push_back(node{});
    }

    void get(char* id) {
        int index = find(id, false);
        if ((long int)(arr[index].id) == 0)
            cout << "undefined" << endl;
        else cout << arr[index].data << endl;
    }

    void put(char* id, int data) {
        if (size + 1 > arr.size() / 2) resize();
        int index = find(id, true);
        if ((long int)(arr[index].id) == 0) {
            arr[index] = node{ id, data };
            size++;
        }
        else arr[index].data = data;
    }
};

vector<string> split(string s) {
    vector<string> res;
    string k;
    for (char t : s) {

```

```

        if (t == ' ' && k.length() > 0) {
            res.push_back(k);
            k = "";
        }
        else if (t != ' ')
            k += t;
    }
    if (k != "") res.push_back(k);
    return res;
}

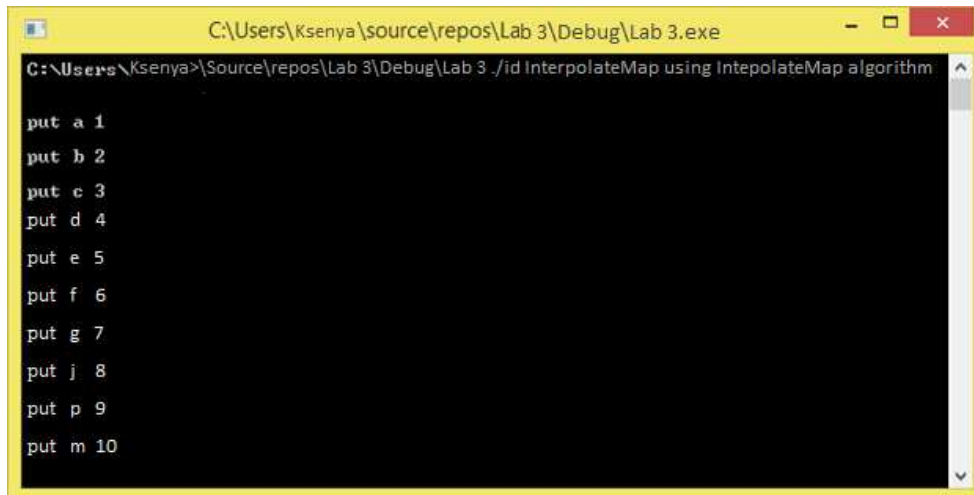
char* get_id(string s)
{
    char* id = (char*)malloc(32);
    memcpy(id, s.c_str(), 32);
    return id;
}

int main(int argc, char* argv[]) {
    string name = argv[1];
    Map* map;

    if (name == "InterpolateMap")
        map = new InterpolateMap();
    else if (name == "ChainHashMap")
        map = new ChainHashMap();
    else {
        cout << "unknown algorithm name" << endl;
        return 0;
    }
    cout << "using " << name << " algorithm" << endl;
    string line;
    while (true) {
        getline(cin, line);
        vector<string> splitted = split(line);
        string cmd = splitted[0];
        if (cmd == "exit") break;
        else if (cmd == "get")
            map->get(get_id(splitted[1]));
        else if (cmd == "put")
            map->put(get_id(splitted[1]), stoi(splitted[2]));
    }
    return 0;
}

```

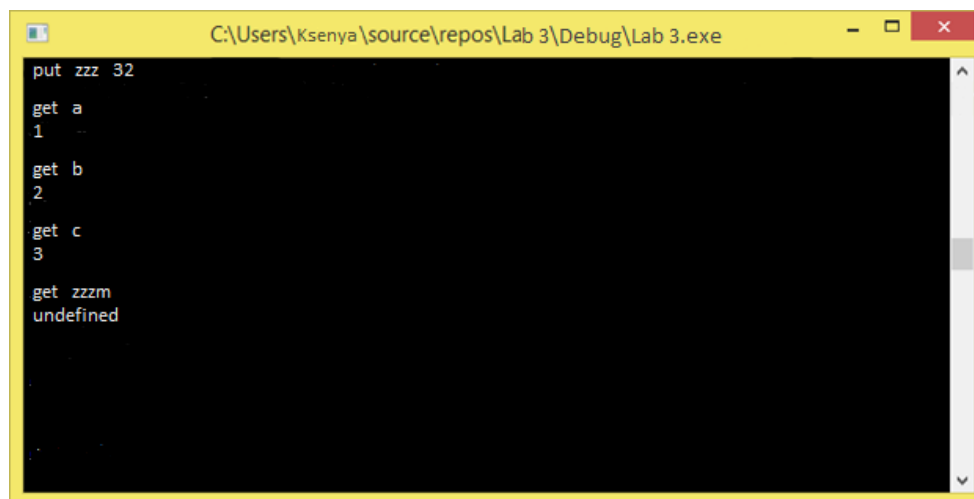
Тестирование программы



```
C:\Users\Ksenya\source\repos\Lab 3\Debug\Lab 3.exe
C:\Users\Ksenya>Source\repos\Lab 3\Debug\Lab 3 ./id InterpolateMap using IntepolateMap algorithm

put a 1
put b 2
put c 3
put d 4
put e 5
put f 6
put g 7
put j 8
put p 9
put m 10
```

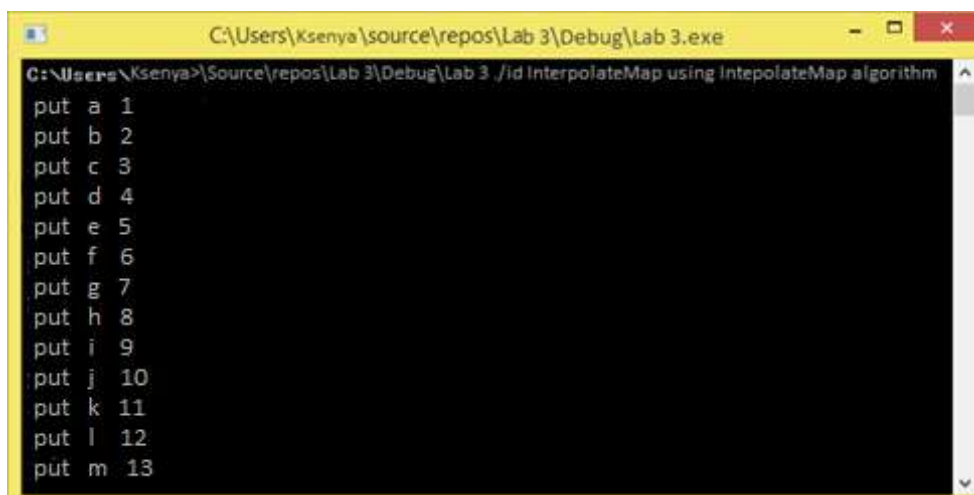
Рисунок 1 – Данные



```
C:\Users\Ksenya\source\repos\Lab 3\Debug\Lab 3.exe

put zzz 32
get a
1
get b
2
get c
3
get zzzm
undefined
```

Рисунок 2 – Данные



```
C:\Users\Ksenya\source\repos\Lab 3\Debug\Lab 3.exe
C:\Users\Ksenya>Source\repos\Lab 3\Debug\Lab 3 ./id InterpolateMap using IntepolateMap algorithm

put a 1
put b 2
put c 3
put d 4
put e 5
put f 6
put g 7
put h 8
put i 9
put j 10
put k 11
put l 12
put m 13
```

Рисунок 3 – Метод цепочек

```

C:\Users\Ksenya\source\repos\Lab 3\Debug\Lab 3.exe
put ff 32
get ff
32
get ee
31
get dd
30
get cc
29
get bb
28
get aa
27
get z

```

Рисунок 4 – Метод цепочек

2. Задание на лабораторную работу

Написать программу, которая получает на входе набор идентификаторов, организует таблицу по заданному методу и позволяет осуществить многократный поиск идентификатора в этой таблице. Список идентификаторов задан в виде текстового файла. Длина идентификаторов ограничена 32 символами.

Варианты заданий

Номер варианта	Первый метод организации таблиц	Первый метод организации таблиц
1	Простое рехэширование	Метод цепочек
2	Простое рехэширование	Простой список
3	Простое рехэширование	Упорядоченный список
4	Простое рехэширование	Бинарное дерево
5	Рехэширование с помощью псевдослучайных чисел	Простое рехэширование
6	Рехэширование с помощью псевдослучайных чисел	Метод цепочек
7	Рехэширование с помощью псевдослучайных чисел	Простой список
8	Рехэширование с помощью псевдослучайных чисел	Упорядоченный список
9	Рехэширование с помощью псевдослучайных чисел	Рехэширование с помощью произведения
10	Рехэширование с помощью псевдослучайных чисел	Бинарное дерево
11	Рехэширование с помощью псевдослучайных чисел	Простое рехэширование

Номер варианта	Первый метод организации таблиц	Первый метод организации таблиц
12	Рехэширование с помощью псевдослучайных чисел	Метод цепочек
13	Рехэширование с помощью псевдослучайных чисел	Простой список
14	Рехэширование с помощью псевдослучайных чисел	Упорядоченный список
15	Рехэширование с помощью псевдослучайных чисел	Бинарное дерево
16	Метод цепочек	Рехэширование с помощью произведения
17	Метод цепочек	Упорядоченный список
18	Метод цепочек	Бинарное дерево
19	Метод цепочек	Простое рехэширование
20	Метод цепочек	Простой список