МИНОБРНАУКИ РОСИИ
Федеральное государственное бюджетное образовательное учреждение высшего образования
«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Отчет по выполнению практического задания №6.1
Тема:
Быстрый доступ к данным с помощью хеш-таблиц
Дисциплина: «Структуры и алгоритмы обработки данных»

Выполнил студент: Васильев Б.А.
Группа: ИКБО-20-23

Москва 2024

СОДЕРЖАНИЕ

ЦЕЛЬ РАБОТЫ 3
ХОД РАБОТЫ 3
Формулировка задачи 3
Описание подхода к решению 4
Коды программы 5
Результаты тестирования 13
ВЫВОД 16
СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ 16

ЦЕЛЬ РАБОТЫ

Освоить приёмы хеширования и эффективного поиска элементов множества.

ХОД РАБОТЫ

Формулировка задачи

Разработайте приложение, которое использует хеш-таблицу (пары «ключ – хеш») для организации прямого доступа к элементам динамического множества полезных данных. Множество реализуйте на массиве, структура элементов (перечень полей) которого приведена в индивидуальном варианте (п.3).

Приложение должно содержать класс с базовыми операциями: вставки, удаления, поиска по ключу, вывода. Включите в класс массив полезных данных и хеш-таблицу. Хеш-функцию подберите самостоятельно, используя правила выбора функции.

Реализуйте расширение размера таблицы и рехеширование, когда это требуется, в соответствии с типом разрешения коллизий.

Предусмотрите автоматическое заполнение таблицы 5-7 записями.

Реализуйте текстовый командный интерфейс пользователя для возможности вызова методов в любой произвольной последовательности, сопроводите вывод достаточными для понимания происходящего сторонним пользователем подсказками.

Проведите полное тестирование программы (все базовые операции, изменение размера и рехеширование), тест-примеры определите самостоятельно. Результаты тестирования включите в отчет по выполненной работе.

Вари-ант	Метод хеширования (тип после-довательностей проб)	Структура элемента множества. Ключи записей под-черкнуты
1	Цепное хеширование	Читательский абонемент: номер читательского - целое пятизначное число, ФИО, адрес

Рисунок 1 – Индивидуальный вариант задачи

Описание подхода к решению

Подход к решению задачи заключается в использовании хеш-таблицы для организации быстрого доступа к элементам данных. Хеш-таблица строится на основе хеш-функции h вида K mod Q, где K — ключ (номер читательского билета), а M — размер хеш-таблицы. Такая функция распределяет элементы по индексам массива, что минимизирует количество коллизий. Коллизии разрешаются с помощью метода цепочек, при котором несколько элементов могут храниться в одном индексе в виде связанных списков.

По мере заполнения таблицы происходит автоматическое рехеширование, когда она становится заполненной на 75% (коэффициент загрузки 0.75). Это означает, что таблица увеличивается в размерах, и все элементы перераспределяются с использованием обновлённой хеш-функции для более равномерного распределения.

Хеш-таблица хранит указатели на записи, каждая из которых представляет собой элемент данных. При возникновении коллизий, несколько записей могут быть размещены в одном индексе в виде связанного списка, где каждая запись содержит ссылку на следующую. Таким образом, в случае совпадения хеш-значений, записи образуют цепочку, что позволяет хранить несколько элементов в одной ячейке таблицы и эффективно разрешать конфликты при вставке и поиске данных.

Коды программы

Реализуем код приложения на языке программирования C++ (рис. 2-9)

```cpp
#ifndef HASH_TABLE_H
#define HASH_TABLE_H
#include <iostream>
#include <vector>
#include <list>
#include <string>
using namespace std;
struct Data_Record
{
    int id;
    string name;
    string address;
    Data_Record* next;

    // Constructor
    Data_Record(int id_value, const string name_value, const string address_value)
                : id(id_value), name(name_value), address(address_value), next(nullptr) {}
    string string_rep();
};
class Hash_Table
{
private:
    Data_Record** table;
    int size;
    int num_of_elements;
    const float load_factor_threshold = 0.75;

    int hash_function(int id) const;
    void rehash();

public:
    Hash_Table(int table_size);
    ~Hash_Table();

    void insert(Data_Record* new_record);
    Data_Record* search(int id);
    int remove(int id);
    void display() const;
};

#endif
```

Рисунок 2 – файл Hash_Table.h

```cpp
#include "Hash_Table.h"

string Data_Record::string_rep()
{
    string output = "";
    output += "[номер читательского билета: " + to_string(id)
        + ", ФИО: " + name
        + ", Адрес: " + address + "]";
    return output;
}

Hash_Table::Hash_Table(int table_size) : size(table_size), num_of_elements(0)
{
    table = new Data_Record*[size];
    for (int i = 0; i < size; i++)
    {
        table[i] = nullptr;
    }
}

Hash_Table::~Hash_Table()
{
    for (int i = 0; i < size; i++)
    {
        Data_Record* current = table[i];
        while (current != nullptr)
        {
            Data_Record* temp = current;
            current = current->next;
            delete temp;
        }

    }
    delete[] table;
}

int Hash_Table::hash_function(int id) const
{
    return id % size;
}
```

Рисунок 3 – Файл Hash_Table.cpp (часть 1)

```cpp
void Hash_Table::rehash()
{
    cout << "Rehashing: increasing table size to " << size * 2 << endl;
    int old_size = size;

    // Creating new empty table of twice the size
    size *= 2;
    Data_Record** new_table = new Data_Record*[size];
    for (int i = 0; i < size; i++)
    {
        new_table[i] = nullptr;
    }

    // Moving data from old table to the new
    for (int i = 0; i < old_size; i++)
    {
        Data_Record* current = table[i];
        while (current != nullptr)
        {
            int new_index = hash_function(current->id);
            Data_Record* next_record = current->next;

            current->next = new_table[new_index];
            new_table[new_index] = current;

            current = next_record;
        }
    }

    delete[] table;
    table = new_table;
}


void Hash_Table::insert(Data_Record* new_record)
{
    int index = hash_function(new_record->id);
    new_record->next = table[index];
    table[index] = new_record;
```

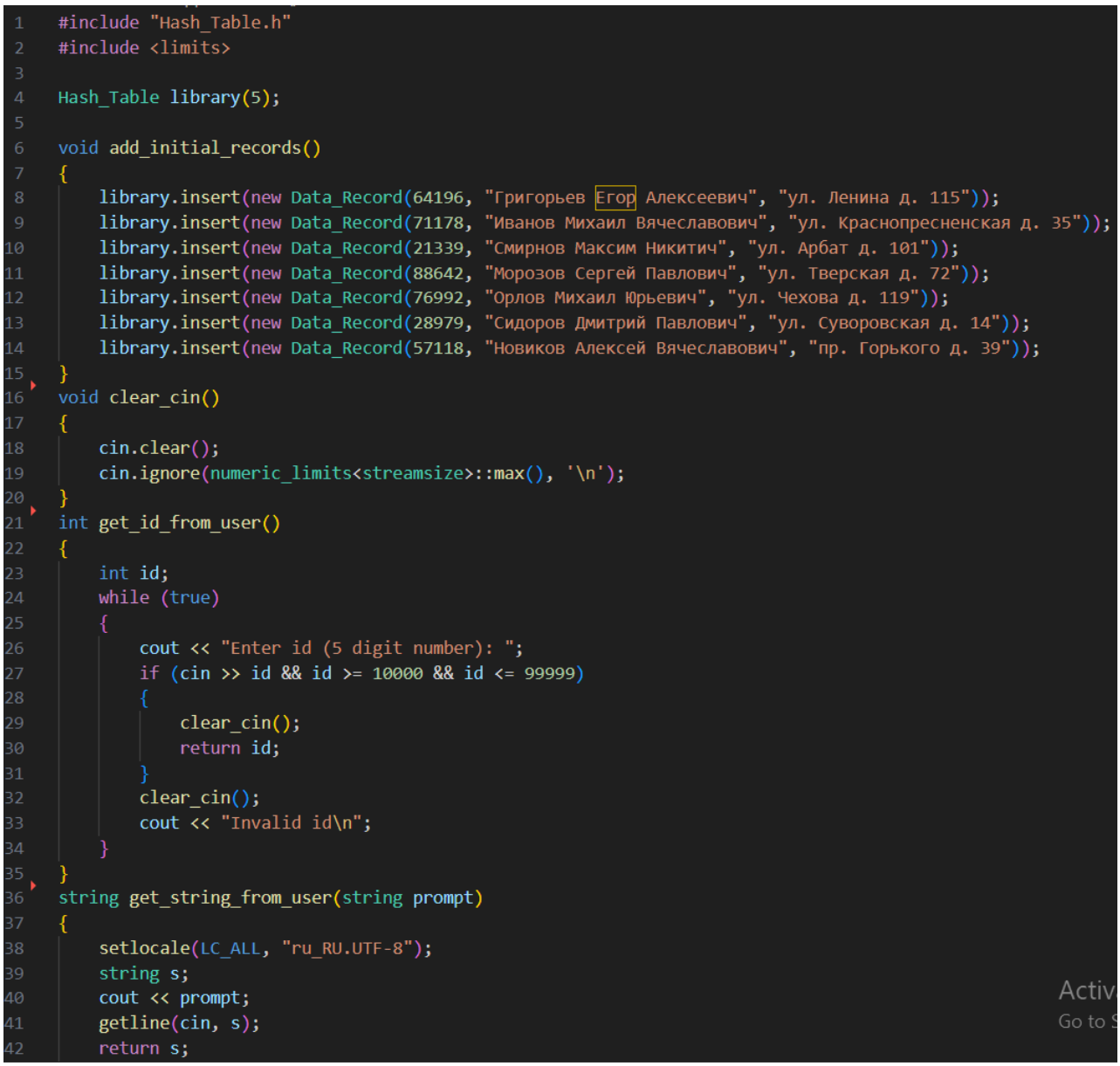
Рисунок 4 – Файл Hash_Table.cpp (часть 2)

```
76  void Hash_Table::insert(Data_Record* new_record)
82      num_of_elements++;
83      if ((static_cast<float>(num_of_elements) / size) > load_factor_threshold)
84      {
85          rehash();
86      }
87
88  }
89
90  Data_Record* Hash_Table::search(int id)
91  {
92      int index = hash_function(id);
93      Data_Record* current = table[index];
94      while (current != nullptr)
95      {
96          if (current->id == id)
97          {
98              return current;
99          }
100         current = current->next;
101     }
102     return nullptr;
103 }
104
105 int Hash_Table::remove(int id)
106 {
107     int index = hash_function(id);
108     Data_Record* current = table[index];
109     Data_Record* prev = nullptr;
110
111     while (current != nullptr)
112     {
113         if (current->id == id)
114         {
115             // Element to delete is first in chain
116             if (prev == nullptr)
117             {
118                 table[index] = current->next;
119             }
```

Рисунок 5 – Файл Hash_Table.cpp (часть 3)

```
105  int Hash_Table::remove(int id)
111      while (current != nullptr)
113          if (current->id == id)
121              else
122              {
123                  prev->next = current->next;
124              }
125              delete current;
126              num_of_elements--;
127              return 0; // Return code: data record successfully deleted
128          }
129          prev = current;
130          current = current->next;
131      }
132      return 1; // Return code: data record not found
133  }
134
135  void Hash_Table::display() const
136  {
137      bool empty_flag = true;
138      for (int i = 0; i < size; i++)
139      {
140          Data_Record* current = table[i];
141          if (current != nullptr)
142          {
143              empty_flag = false;
144              cout << "Index " << i << ": ";
145              while (current != nullptr)
146              {
147                  cout << current->string_rep() << endl;
148                  current = current->next;
149              }
150              cout << endl;
151          }
152      }
153      if (empty_flag)
154      {
155          cout << "No data" << endl;
156      }
157  }
```

Рисунок 6 – Файл Hash_Table.cpp (часть 4)



```cpp
#include "Hash_Table.h"
#include <limits>

Hash_Table library(5);

void add_initial_records()
{
    library.insert(new Data_Record(64196, "Григорьев Егор Алексеевич", "ул. Ленина д. 115"));
    library.insert(new Data_Record(71178, "Иванов Михаил Вячеславович", "ул. Краснопресненская д. 35"));
    library.insert(new Data_Record(21339, "Смирнов Максим Никитич", "ул. Арбат д. 101"));
    library.insert(new Data_Record(88642, "Морозов Сергей Павлович", "ул. Тверская д. 72"));
    library.insert(new Data_Record(76992, "Орлов Михаил Юрьевич", "ул. Чехова д. 119"));
    library.insert(new Data_Record(28979, "Сидоров Дмитрий Павлович", "ул. Суворовская д. 14"));
    library.insert(new Data_Record(57118, "Новиков Алексей Вячеславович", "пр. Горького д. 39"));
}
void clear_cin()
{
    cin.clear();
    cin.ignore(numeric_limits<streamsize>::max(), '\n');
}
int get_id_from_user()
{
    int id;
    while (true)
    {
        cout << "Enter id (5 digit number): ";
        if (cin >> id && id >= 10000 && id <= 99999)
        {
            clear_cin();
            return id;
        }
        clear_cin();
        cout << "Invalid id\n";
    }
}
string get_string_from_user(string prompt)
{
    setlocale(LC_ALL, "ru_RU.UTF-8");
    string s;
    cout << prompt;
    getline(cin, s);
    return s;
```

Рисунок 7 – Файл main.cpp (часть 1)

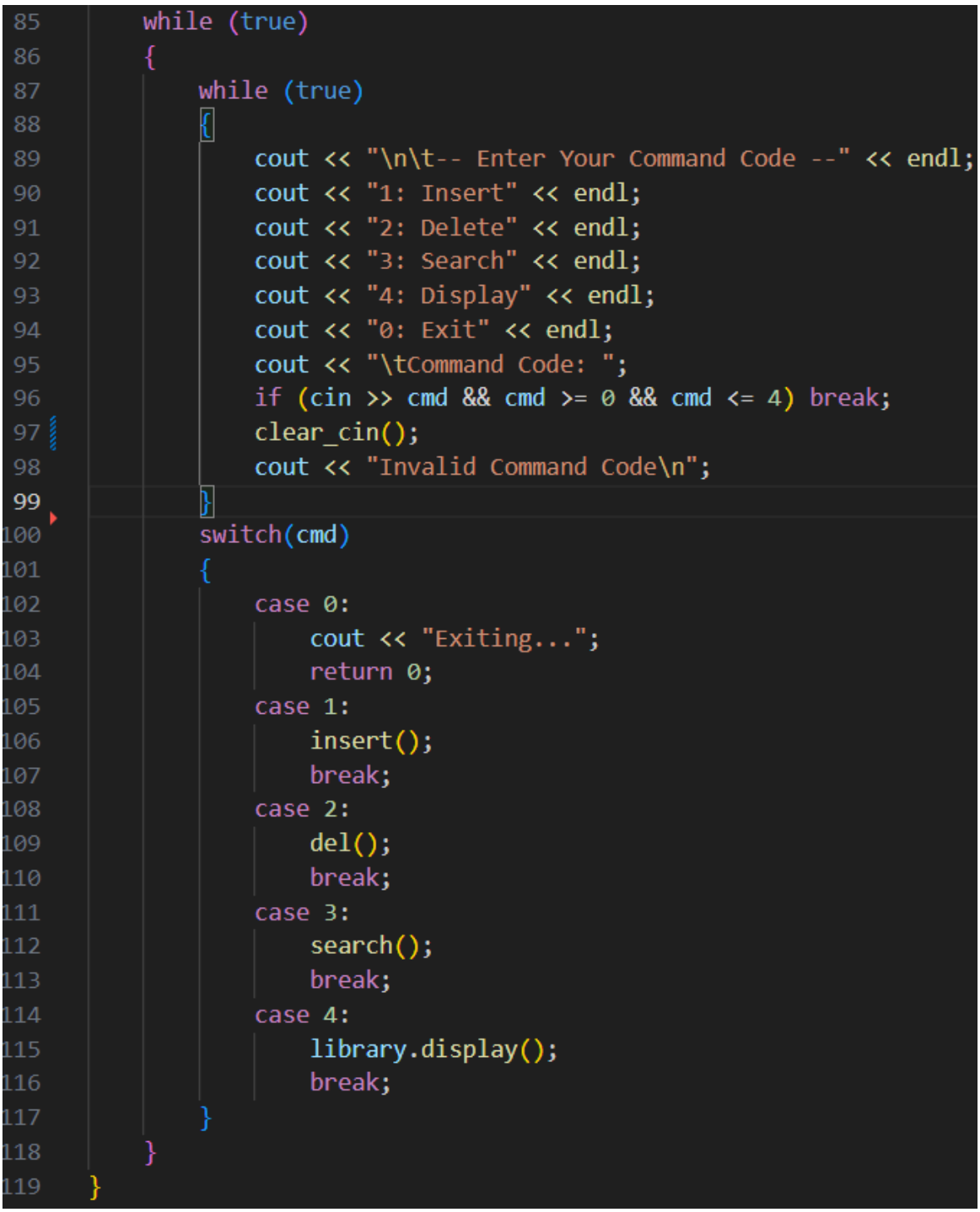
```cpp
void insert()
{
    int id = get_id_from_user();
    string name = get_string_from_user("Enter name: ");
    string address = get_string_from_user("Enter address: ");
    Data_Record* new_record = new Data_Record(id, name, address);
    library.insert(new_record);
    cout << "Successfully inserted record: " << new_record->string_rep() << endl;
};
void del()
{
    int id = get_id_from_user();
    int return_code = library.remove(id);
    if (return_code == 0)
    {
        cout << "Record with id = " << id << " was successfully deleted" << endl;
    }
    else
    {
        cout << "Record with id = " << id << " wasn't found" << endl;
    }
};
void search()
{
    int id = get_id_from_user();
    Data_Record* record = library.search(id);
    if (record == nullptr)
    {
        cout << "Record with id = " << id << " not found" << endl;
        return;
    }
    cout << "Found record: " << record->string_rep() << endl;

};
int main()
{
    setlocale(LC_ALL, "ru_RU.UTF-8");

    add_initial_records();

    int cmd;
    while (true)
```

Рисунок 8 – Файл main.cpp (часть 2)



```
85      while (true)
86      {
87          while (true)
88          {
89              cout << "\n\t-- Enter Your Command Code --" << endl;
90              cout << "1: Insert" << endl;
91              cout << "2: Delete" << endl;
92              cout << "3: Search" << endl;
93              cout << "4: Display" << endl;
94              cout << "0: Exit" << endl;
95              cout << "\tCommand Code: ";
96              if (cin >> cmd && cmd >= 0 && cmd <= 4) break;
97              clear_cin();
98              cout << "Invalid Command Code\n";
99          }
100         switch(cmd)
101         {
102             case 0:
103                 cout << "Exiting...";
104                 return 0;
105             case 1:
106                 insert();
107                 break;
108             case 2:
109                 del();
110                 break;
111             case 3:
112                 search();
113                 break;
114             case 4:
115                 library.display();
116                 break;
117         }
118     }
119 }
```

Рисунок 9 – Файл main.cpp (часть 3)

Результаты тестирования

Выполним тестирование программы (рис. 10-15).

```
D:\Boris\Documents\UNI\SIAOD\6_1>g++ main.cpp Hash_Table.cpp -o main && .\main "d:\Boris\Documents\UNI\SIAOD\6_1\main.cpp"
Rehashing: increasing table size to 10

       -- Enter Your Command Code --
1: Insert
2: Delete
3: Search
4: Display
0: Exit
       Command Code: 4
Displaying all records:
Index 2: [номер читательского билета: 76992, ФИО: Орлов Михаил Юрьевич, Адрес: ул. Чехова д. 119]
[номер читательского билета: 88642, ФИО: Морозов Сергей Павлович, Адрес: ул. Тверская д. 72]

Index 6: [номер читательского билета: 64196, ФИО: Григорьев Егор Алексеевич, Адрес: ул. Ленина д. 115]

Index 8: [номер читательского билета: 57118, ФИО: Новиков Алексей Вячеславович, Адрес: пр. Горького д. 39]
[номер читательского билета: 71178, ФИО: Иванов Михаил Вячеславович, Адрес: ул. Краснопресненская д. 35]

Index 9: [номер читательского билета: 28979, ФИО: Сидоров Дмитрий Павлович, Адрес: ул. Суворовская д. 14]
[номер читательского билета: 21339, ФИО: Смирнов Максим Никитич, Адрес: ул. Арбат д. 101]
```

Рисунок 10 – Результат автоматического заполнения семью записями и тестирование вывода всех записей

```
Index 9: [номер читательского билета: 28979, ФИО: Сидоров Дмитрий Павлович, Адрес: ул. Суворовская д. 14]
[номер читательского билета: 21339, ФИО: Смирнов Максим Никитич, Адрес: ул. Арбат д. 101]


       -- Enter Your Command Code --
1: Insert
2: Delete
3: Search
4: Display
0: Exit
       Command Code: 3
Enter id (5 digit number): 28979
Found record: [номер читательского билета: 28979, ФИО: Сидоров Дмитрий Павлович, Адрес: ул. Суворовская д. 14]
```

Рисунок 11 – Тестирование поиска по ключу (часть 1)

```
       -- Enter Your Command Code --
1: Insert
2: Delete
3: Search
4: Display
0: Exit
       Command Code: 3
Enter id (5 digit number): 99999
Record with id = 99999 not found
```

Рисунок 12 – Тестирование поиска по ключу (часть 2)

```
Index 9: [номер читательского билета: 28979, ФИО: Сидоров Дмитрий Павлович, Адрес: ул. Суворовская д. 14]
[номер читательского билета: 21339, ФИО: Смирнов Максим Никитич, Адрес: ул. Арбат д. 101]


        -- Enter Your Command Code --
1: Insert
2: Delete
3: Search
4: Display
0: Exit
        Command Code: 2
Enter id (5 digit number): 28979
Record with id = 28979 was successfully deleted

        -- Enter Your Command Code --
1: Insert
2: Delete
3: Search
4: Display
0: Exit
        Command Code: 4
Displaying all records:
Index 2: [номер читательского билета: 76992, ФИО: Орлов Михаил Юрьевич, Адрес: ул. Чехова д. 119]
[номер читательского билета: 88642, ФИО: Морозов Сергей Павлович, Адрес: ул. Тверская д. 72]

Index 6: [номер читательского билета: 64196, ФИО: Григорьев Егор Алексеевич, Адрес: ул. Ленина д. 115]

Index 8: [номер читательского билета: 57118, ФИО: Новиков Алексей Вячеславович, Адрес: пр. Горького д. 39]
[номер читательского билета: 71178, ФИО: Иванов Михаил Вячеславович, Адрес: ул. Краснопресненская д. 35]

Index 9: [номер читательского билета: 21339, ФИО: Смирнов Максим Никитич, Адрес: ул. Арбат д. 101]
```

Рисунок 13 – Тестирование удаления

```
        Command Code: 1
Enter id (5 digit number): 10001
Enter name: A
Enter address: B
Successfully inserted record: [номер читательского билета: 10001, ФИО: A, Адрес: B]

        -- Enter Your Command Code --
1: Insert
2: Delete
3: Search
4: Display
0: Exit
        Command Code: 1
Enter id (5 digit number): 20001
Enter name: C
Enter address: D
Rehashing: increasing table size to 20
Successfully inserted record: [номер читательского билета: 20001, ФИО: C, Адрес: D]

        -- Enter Your Command Code --
1: Insert
2: Delete
3: Search
4: Display
0: Exit
        Command Code: 4
Displaying all records:
Index 1: [номер читательского билета: 10001, ФИО: A, Адрес: B]
[номер читательского билета: 20001, ФИО: C, Адрес: D]

Index 2: [номер читательского билета: 88642, ФИО: Морозов Сергей Павлович, Адрес: ул. Тверская д. 72]
```

Рисунок 14 – Тестирование добавления записей с одинаковым значением хэш-функции

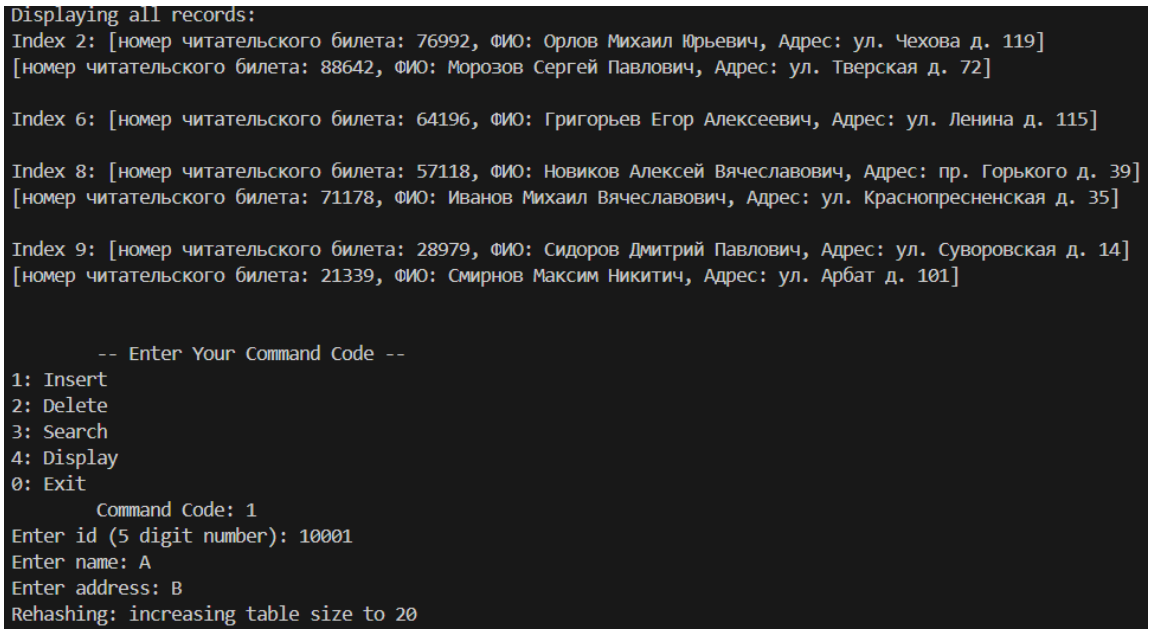


Рисунок 15 – Тестирование рехеширования.

Тестирование показало, что программа работает корректно, в том числе в случае возникновения коллизий.

ВЫВОД

В результате выполнения работы были освоены приёмы хеширования и эффективного поиска элементов множества. Был получен опыт реализации хэш-таблицы с цепным хешированием.

СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ

1. Рысин, М. Л. Введение в структуры и алгоритмы обработки данных : учебное пособие / М. Л. Рысин, М. В. Сартаков, М. Б. Туманова. — Москва : РТУ МИРЭА, 2022 — Часть 2 : Поиск в тексте. Нелинейные структуры данных. Кодирование информации. Алгоритмические стратегии — 2022. — 111 с. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/310826 (дата обращения: 25.09.2024).

2. Документация по языку C++ [Электронный ресурс]. URL: https://docs.microsoft.com/ruru/cpp/cpp/ (дата обращения 25.09.2024).