**ФГБОУ ВО   
Уфимский университет науки и технологий**

**Кафедра ВМиК**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 100 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| 90 |  |  |  |  |  |  |  |  |  |  |  |  |
| 80 |  |  |  |  |  |  |  |  |  |  |  |  |
| 70 |  |  |  |  |  |  |  |  |  |  |  |  |
| 60 |  |  |  |  |  |  |  |  |  |  |  |  |
| 50 |  |  |  |  |  |  |  |  |  |  |  |  |
| 40 |  |  |  |  |  |  |  |  |  |  |  |  |
| 30 |  |  |  |  |  |  |  |  |  |  |  |  |
| 20 |  |  |  |  |  |  |  |  |  |  |  |  |
| 10 |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |

Построение хеш-таблицы

**ОТЧЕТ**

**по лабораторной работе**

**по** Структурам и алгоритмам

компьютерной обработки данных

(*наименование дисциплины*)

|  |
| --- |
| Лабораторная работа 2 (Вар. 11) |
| (обозначение документа) |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Группа |  |  | Фамилия, И., О. | Подпись | Дата | Оценка |
| МО-325Б |  |
|  |  |
| Студент | | | Шарыгин М.С. |  |  |  |
| Преподаватель | | | Верхотурова Г.Н. |  |  |  |
| Принял | | |  |  |  |  |

**Уфа 2024 г****.**

Содержание

[1 Цель работы 3](#_Toc184077197)

[2 Практическая часть 4](#_Toc184077198)

[2.1 Задание на лабораторную работу 4](#_Toc184077199)

[2.2 Входные и выходные данные 4](#_Toc184077200)

[2.3 Интерфейс работы с хеш-таблицей 5](#_Toc184077201)

[2.4 Алгоритм поиска 6](#_Toc184077202)

[2.5 Алгоритм удаления 7](#_Toc184077203)

[2.6 Алгоритм добавления 8](#_Toc184077204)

[2.7 Алгоритм замены 9](#_Toc184077205)

[2.8 Формулы расчета параметров хеш-таблицы 11](#_Toc184077206)

[3 Вывод 13](#_Toc184077207)

# Цель работы

Целью работы является добавление функций поиска, добавления, удаления и замены в построенную на предыдущей лабораторной работе хеш-таблицу, содержащую заданную последовательность элементов (ключей).

# Практическая часть

## Задание на лабораторную работу

Реализовать в построенной ранее хеш-таблице возможность поиска, добавления, удаления и замены элементов.

## Входные и выходные данные

Входные данные:

* table[size\_table] – исходная хеш-таблица;
* size\_table – размер таблицы, который равен количеству элементов, умноженных в полтора раза;
* size\_sample – количество элементов в таблице;
* r – вводимое пользователем значение для поиска, добавления, удаления и замены.

Промежуточные данные:

* transition – число шагов, на котором произойдет замена метода устранения коллизий с открытой адресации с квадратичным опробыванием на открытую адресацию с линейным опробыванием;
* steps – количество шагов, потраченное на размещение всех элементов в хеш-таблице;
* number\_placements – количество успешных запусков функции добавления элемента;
* key\_value[7] {0, 0 , size\_sample, size\_table, steps, transition, number\_placements} – массив, в котором содержатся все данные для реализации функций; необходим, чтобы не передавать все параметры в функции по отдельность.

Выходные данные:

* table[size\_table] – итоговая хеш-таблица;
* size\_sample / size\_table – коэффициент заполнения таблицы;
* steps / size\_sample – среднее число шагов;
* size\_sample – итоговое количество элементов в таблице.

## Интерфейс работы с хеш-таблицей

Для работы с хеш-таблицей будем использовать бесконечный цикл «while», который позволит выбирать в произвольным порядке любой алгоритм: поиск, добавление, удаление, замена и расчет параметров. Реализация цикла приведена на рисунке 2.1.

Изображение выглядит как текст, снимок экрана, число, Шрифт

Автоматически созданное описание

Рисунок 2.1 – Программная реализация интерфейса

На рисунке 2.2 показан пример вывода интерфейса в консоли.

Изображение выглядит как текст, Шрифт, снимок экрана, алгебра

Автоматически созданное описание

Рисунок 2.2 – Интерфейс программы в консоли

## Алгоритм поиска

Для того, чтобы найти элемент в хеш-таблице, воспользуемся функцией «search»:

* сначала применяем хеш-функцию к элементу;
* после чего проверяем его наличие / отсутствие в полученном хеш-функцией индексе;
* если в нем [индексе] находится проверяемый элемент, то поиск выполнен успешно, следовательно, выводим соответствующее сообщение в консоль (рисунок 2.3);

Изображение выглядит как текст, Шрифт, снимок экрана, белый

Автоматически созданное описание

Рисунок 2.3 – Пример успешного завершения поиска

* если в индексе – 0, то такого элемента в таблице нет, о чем также выводим сообщение в консоль (рисунок 2.4);

Изображение выглядит как текст, Шрифт, снимок экрана, линия

Автоматически созданное описание

Рисунок 2.4 – Пример неуспешного завершения поиска

* иначе применяем метод устранения коллизий – открытая адресация с квадратичным опробованием, как показано на рисунок 2.5, пока не будет найден запрашиваемый элемент или пока не выяснится его отсутствие;

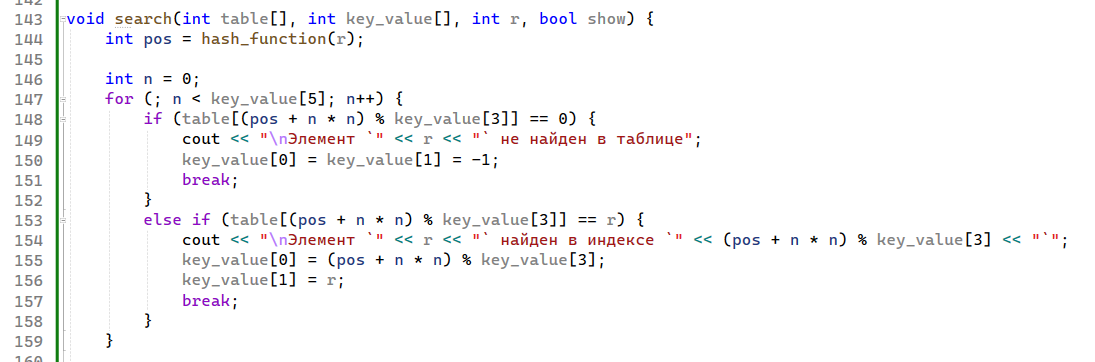


Рисунок 2.5 – Поиск с применением квадратичного опробования

* если количество итераций превысит 50, а нужный элемент не будет найден, то переходим к открытой адресации с линейным опробованием и продолжаем поиск до тех пор, пока не обнаружим элемент или его отсутствие (рисунок 2.6).

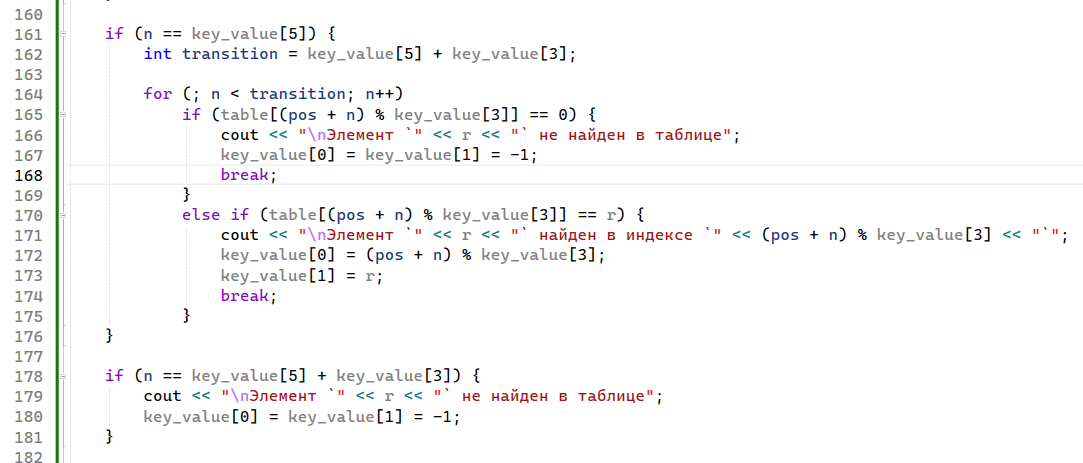


Рисунок 2.6 – Поиск с применением линейного опробования

## Алгоритм удаления

Реализуем удаление элемента следующим образом:

* сначала проверяем с помощью функции поиска[[1]](#footnote-1), описанной ранее, наличие удаляемого элемента в таблице (рисунок 2.7);

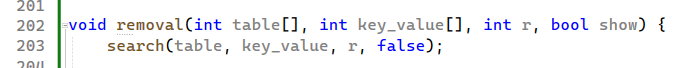


Рисунок 2.7 – Проверка наличия элемента в таблице

* затем, если элемент найден, то в тот индекс, в котором он находится, вписываем «-1» и выводим сообщение об удалении (рисунок 2.8 и рисунок 2.9);

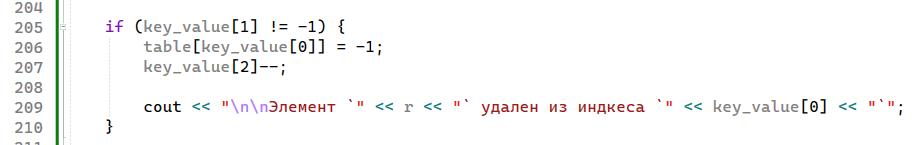


Рисунок 2.8 – Алгоритм удаления элемента из таблицы

Изображение выглядит как текст, Шрифт, снимок экрана, белый

Автоматически созданное описание

Рисунок 2.9 – Пример успешного удаления элемента

* иначе, если элемент не найден, сообщаем об этом пользователю, как приведено на рисунке 2.10.

Изображение выглядит как текст, Шрифт, снимок экрана, белый

Автоматически созданное описание

Рисунок 2.10 – Пример неуспешного удаления элемента

## Алгоритм добавления

Теперь реализуем функцию добавления:

* для начала нужно убедиться, что добавляемого элемента нет в таблице, для чего воспользуемся функцией «search»[[2]](#footnote-2), которая вернет «-1» в данном случае (рисунок 2.11);

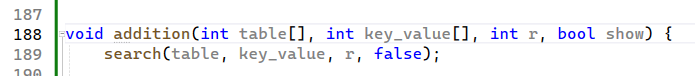


Рисунок 2.11 – Проверка отсутствия элемента

* затем, если элемент найден, то завершаем алгоритм добавления (рисунок 2.12);

Изображение выглядит как текст, Шрифт, снимок экрана, белый

Автоматически созданное описание

Рисунок 2.12 – Пример неуспешного добавления элемента

* в противном случае применяем функцию «add»:
  + сначала для добавляемого элемента применяем хеш-функцию и проверяем полученный индекс;
  + если он пустой, то добавляем элемент в него (рисунок 2.13);

Изображение выглядит как текст, Шрифт, снимок экрана, белый

Автоматически созданное описание

Рисунок 2.13 – Пример успешного добавления элемента

* + иначе применяем метод устранения коллизий, как показано на рисунке 2.14.

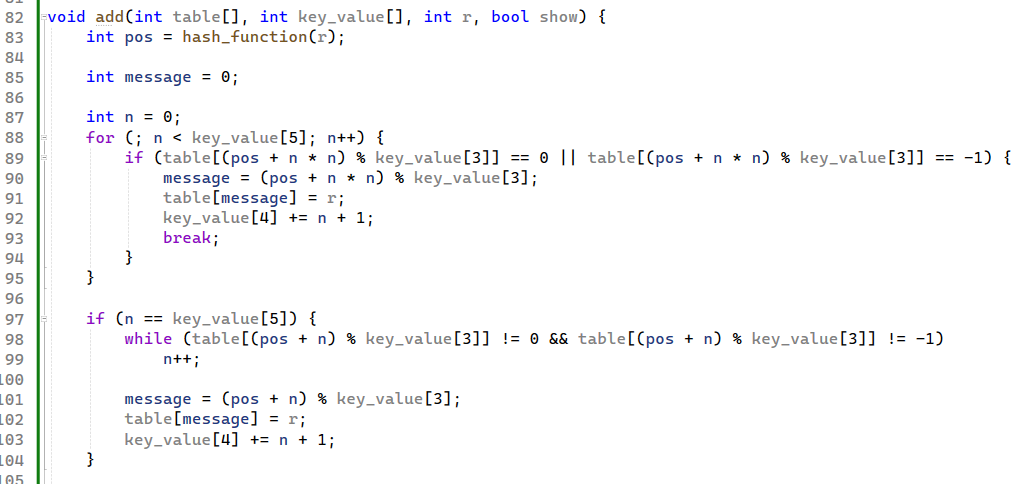


Рисунок 2.14 – Алгоритм добавления элемента в таблицу

## Алгоритм замены

Наконец реализуем замену элементов:

* для начала применим функцию удаления[[3]](#footnote-3) для первого введенного элемента;
* затем применим функцию добавления[[4]](#footnote-4) для второго элемента, как приведено на рисунке 2.9.

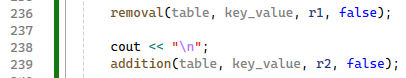


Рисунок 2.15 – Замена элементов в таблице

На рисунках 2.16, 2.17, 2.18 и 2.19 представлены примеры работы алгоритма замены элементов.

Изображение выглядит как текст, Шрифт, снимок экрана, алгебра

Автоматически созданное описание

Рисунок 2.16 – Удаляемый и добавляемый элементы присутствуют в таблице

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 2.17 – Удаляемый элемент присутствует в таблице, а добавляемый отсутствует

Изображение выглядит как текст, Шрифт, снимок экрана, белый

Автоматически созданное описание

Рисунок 2.18 – Удаляемый элемент отсутствует в таблице, а добавляемый присутствует

Изображение выглядит как текст, снимок экрана, Шрифт, алгебра

Автоматически созданное описание

Рисунок 2.19 – Удаляемый и добавляемый элементы отсутствуют в таблице

## Формулы расчета параметров хеш-таблицы

Коэффициент заполнения таблицы рассчитывается делением количества элементов на размер таблицы:

Среднее число шагов, необходимых для размещения некоторого ключа в таблице – это отношение количества шагов к количеству элементов:

Количество элементов высчитывать не нужно, так как переменная «size\_sample» хранит это значение.

В языке программирования «C++» расчёт параметров реализовывается так, как показано на рисунке 2.20. Пример вычисленных параметров изображен на рисунке 2.21.

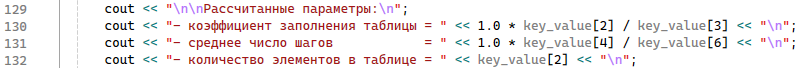


Рисунок 2.20 – Формулы расчета параметров

Изображение выглядит как текст, Шрифт, снимок экрана, белый

Автоматически созданное описание

Рисунок 2.21 – Пример вычисленных параметров

# Вывод

В ходе лабораторной работы мы добавили функции поиска, добавления, удаления и замены в построенную на предыдущей лабораторной работе хеш-таблицу, содержащую заданную последовательность элементов (ключей).

1. Алгоритм поиска была рассмотрена в пункте 2.4. [↑](#footnote-ref-1)
2. Алгоритм поиска была рассмотрена в пункте 2.4. [↑](#footnote-ref-2)
3. Алгоритм удаления была рассмотрена в пункте 2.5. [↑](#footnote-ref-3)
4. Алгоритм добавления была рассмотрена в пункте 2.6. [↑](#footnote-ref-4)