Министерство науки и высшего образования Российской Федерации
Калужский филиал
федерального государственного бюджетного
образовательного учреждения высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(КФ МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ *ИУК "Информатика и управление"*

КАФЕДРА *ИУК5 "Системы обработки информации"*

ЛАБОРАТОРНАЯ РАБОТА №6

«Хеш-таблица и функции хеширования»

ДИСЦИПЛИНА: «Вычислительные алгоритмы»

Выполнил: студент гр.ИУК5-41Б _____ (____Шиндин А.О.___)
(Подпись) (Ф.И.О.)

Проверил: _____ (____ Вершинин В.Е__)
(Подпись) (Ф.И.О.)

Дата сдачи (защиты):

Результаты сдачи (защиты):

- Балльная оценка:
- Оценка:

Калуга, 2023

Цель: изучить построение функции хеширования и алгоритмов хеширования данных и научиться разрабатывать алгоритмы открытого и закрытого хеширования при решении задач.

Задачи:

• изучить словесную постановку задачи, выделив при этом все виды данных;
• сформулировать математическую постановку задачи;
• выбрать метод решения задачи, если это необходимо;
• реализовать алгоритм;

Задание для варианта 13:

Постройте хеш-таблицу для зарезервированных слов, используемого языка программирования (не менее 20 слов), содержащую HELP для каждого слова. Выдайте на экран подсказку по введенному слову. Добавьте подсказку по вновь введенному слову, используя при необходимости реструктуризацию таблицы. Сравните эффективность добавления ключа в таблицу или ее реструктуризацию для различной степени заполненности таблицы.

Ход работы:

Задача состоит в реализации хеш-таблицы для хранения и доступа к данным о зарезервированных словах программирования. Виды данных включают:

• Зарезервированные слова (ключи): строки, представляющие ключевые слова языка программирования.
• Соответствующие значения (значения): строки, представляющие справочную информацию о ключевых словах.

Программа написанная на языке c++:

```
#include <iostream>
#include <string>
#include <vector>
#include <unistd.h>


class HashTable {
private:
struct Node {
std::string key;
std::string value;
long long int hash_value;
Node* next;
```

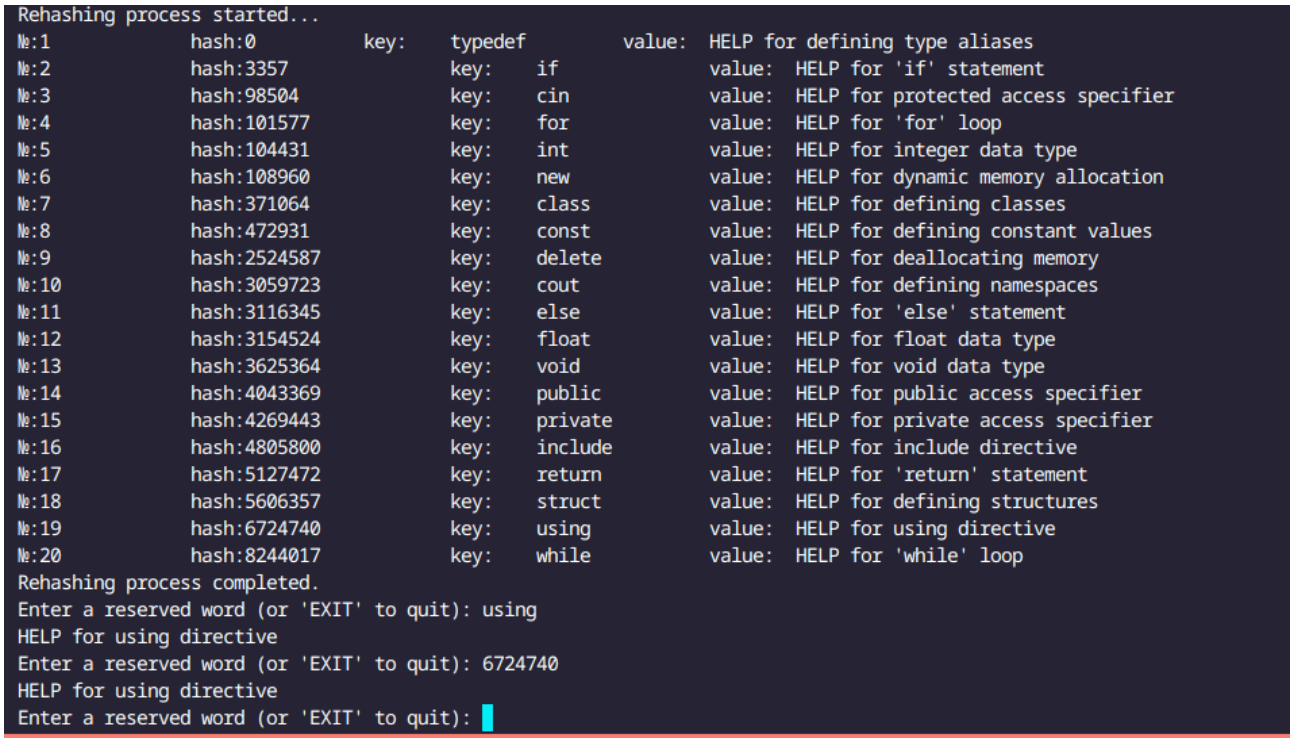
```
Node(const std::string& k, const std::string& v, long long int hash) : key(k), value(v),
hash_value(hash), next(nullptr) {} // Конструктор узла
};

std::vector<Node*> table;
int size;

int hash(const std::string& key) {
return 0;
}

public:
HashTable(int initialSize = 20) : size(0) {
table.resize(initialSize, nullptr);
}

void insert(const std::string& key, const std::string& value) {
long long int hashValue = 0;
Node* newNode = new Node(key, value, hashValue); // Создаем новый узел с
хешем
newNode->next = table[newNode->hash_value]; // Устанавливаем новый узел в
начало цепочки
table[newNode->hash_value] = newNode;
size++;
rehash();
}



std::string get(const std::string& key) {
for (Node* head : table) {
Node* current = head;
while (current != nullptr) {
if (current->key == key) {
return current->value;
}
current = current->next;
}
```

```
}

if (std::to_string(table[std::stoi(key)]->hash_value) == key) { return
table[std::stoi(key)]->value;}
return "Key not found";
}



void rehash() {
std::cout << "Rehashing process started..." << std::endl;
std::vector<Node*> newTable(table.size() * 2, nullptr);
int i{};
i++;

for (Node* head : table) {
Node* current = head;
while (current != nullptr) {
long long int hashValue{};
for (char ch : current->key) {
hashValue = (hashValue * 31 + ch) % newTable.size();
}
Node* newNode = new Node(current->key, current->value, hashValue);
newNode->next = newTable[newNode->hash_value];
newTable[newNode->hash_value] = newNode;
std::cout << "№:" << i << "tthash:"<< current->hash_value <<"ttkey:t" << current-
>key << "ttvalue:t" << current->value << std::endl;
Node* temp = current;
current = current->next;
delete temp;
i++;
}
}
table = std::move(newTable);
std::cout << "Rehashing process completed." << std::endl;
}


};
```

```
int main() {
HashTable reservedWords;
reservedWords.insert("if", "HELP for 'if' statement");
reservedWords.insert("else", "HELP for 'else' statement");
reservedWords.insert("while", "HELP for 'while' loop");
reservedWords.insert("for", "HELP for 'for' loop");
reservedWords.insert("int", "HELP for integer data type");
reservedWords.insert("float", "HELP for float data type");
reservedWords.insert("void", "HELP for void data type");
reservedWords.insert("return", "HELP for 'return' statement");
reservedWords.insert("include", "HELP for include directive");
reservedWords.insert("using", "HELP for using directive");
reservedWords.insert("class", "HELP for defining classes");
reservedWords.insert("public", "HELP for public access specifier");
reservedWords.insert("private", "HELP for private access specifier");
reservedWords.insert("cin", "HELP for protected access specifier");
reservedWords.insert("cout", "HELP for defining namespaces");
reservedWords.insert("const", "HELP for defining constant values");
reservedWords.insert("new", "HELP for dynamic memory allocation");
reservedWords.insert("delete", "HELP for deallocating memory");
reservedWords.insert("struct", "HELP for defining structures");
reservedWords.insert("typedef", "HELP for defining type aliases");


std::string input;
std::string value;
while (true) {
std::cout << "033[2J033[1;1H";
std::cout << "Enter a reserved word (or 'EXIT' to quit): ";
std::getline(std::cin, input);


if (input == "EXIT" || input == "00") {
break;
}


std::string help = reservedWords.get(input);
if (help == "Key not found") {
```

```
std::cout << "Word not found in the reserved words list. Adding it to the table." <<
std::endl;
std::cout << "Enter a help available word (value): t";
std::getline(std::cin, value);
reservedWords.insert(input, (value.size() >= 1)?value:"No help available for this
word yet.");
} else {
std::cout << help << std::endl;
}
// getchar();
}
return 0;


}
```

Результат:



Вывод: в результате выполнения лабораторной работы было изучено построение функции хеширования и алгоритмов хеширования данных и были приобретены практичекие навыки разработки алгоритмов открытого и закрытого хеширования.