

**LAPORAN PRAKTIKUM STRUKTUR DATA DAN
ALGORITMA
MODUL 5
HASH TABLE**



DISUSUN OLEH:

PRIESTY AMEILIANA MAULIDAH

2311102175

S1 IF-11-E

DOSEN:

Muhammad Afrizal Amrustian, S. Kom.

PROGRAM STUDI S1 TEKNIK INFORMATIKA

FAKULTAS INFORMATIKA

INSTITUT TEKNOLOGI TELKOM PURWOKERTO PURWOKERTO

2024

A.DASAR TEORI

1. Open Hashing (Chaining)

Metode chaining mengatasi collision dengan cara menyimpan semua

item data dengan nilai indeks yang sama ke dalam sebuah linked list. Setiap

node pada linked list merepresentasikan satu item data. Ketika ada pencarian

atau penambahan item data, pencarian atau penambahan dilakukan pada

linked list yang sesuai dengan indeks yang telah dihitung dari kunci yang di

hash. Ketika linked list memiliki banyak node, pencarian atau penambahan

item data menjadi lambat, karena harus mencari di seluruh linked list.

Namun, chaining dapat mengatasi jumlah item data yang besar dengan

efektif, karena keterbatasan array dihindari.

2. Closed Hashing

- Linear Probing

Pada saat terjadi collision, maka akan mencari posisi yang kosong di

bawah tempat terjadinya collision, jika masih penuh terus ke bawah,

hingga ketemu tempat yang kosong. Jika tidak ada tempat yang kosong

berarti HashTable sudah penuh.

- Quadratic Probing

Penanganannya hampir sama dengan metode linear, hanya

lompatannya tidak satu-satu, tetapi quadratic (12, 22, 32, 42, ...)

- Double Hashing

Pada saat terjadi collision, terdapat fungsi hash yang kedua untuk

menentukan posisinya kembali.

B.Guided

Guided 1

```
// priesty ameiliana maulidah
// 2311102175

#include <iostream>

using namespace std;

const int MAX_SIZE = 10;

// Fungsi hash sederhana
int hash_func(int key) {
    return key % MAX_SIZE;
}

// Struktur data untuk setiap node
struct Node {
    int key;
    int value;
    Node* next;
    Node(int key, int value) : key(key), value(value),
    next(nullptr) {}
};
```

```

// Class hash table

class HashTable {
private:
    Node** table;
public:
    HashTable() {
        table = new Node*[MAX_SIZE]();
    }
    ~HashTable() {
        for (int i = 0; i < MAX_SIZE; i++) {
            Node* current = table[i];
            while (current != nullptr) {
                Node* temp = current;
                current = current->next;
                delete temp;
            }
        }
        delete[] table;
    }
    // Insertion
    void insert(int key, int value) {
        int index = hash_func(key);
        Node* current = table[index];
        while (current != nullptr) {
            if (current->key == key) {
                current->value = value;
                return;
            }
        }
    }

```

```
current = current->next;
}

Node* node = new Node(key, value);
node->next = table[index];
table[index] = node;
}

// Searching
int get(int key) {
    int index = hash_func(key);
    Node* current = table[index];
    while (current != nullptr) {
        if (current->key == key) {
            return current->value;
        }
        current = current->next;
    }
    return -1;
}
```

```
// Deletion

void remove(int key) {
    int index = hash_func(key);
    Node* current = table[index];
    Node* prev = nullptr;
    while (current != nullptr) {
        if (current->key == key) {
            if (prev == nullptr) {
                table[index] = current->next;
            } else {
                prev->next = current->next;
            }
            delete current;
            return;
        }
        prev = current;
        current = current->next;
    }
}
```

```

// Traversal
void traverse() {
for (int i = 0; i < MAX_SIZE; i++) {
Node* current = table[i];
while (current != nullptr) {
cout << current->key << ": " << current->value

<< endl;

current = current->next;
}
}
}
};

int main() {
HashTable ht;

// Insertion
ht.insert(1, 10);
ht.insert(2, 20);
ht.insert(3, 30);

// Searching
cout << "Get key 1: " << ht.get(1) << endl;
cout << "Get key 4: " << ht.get(4) << endl;

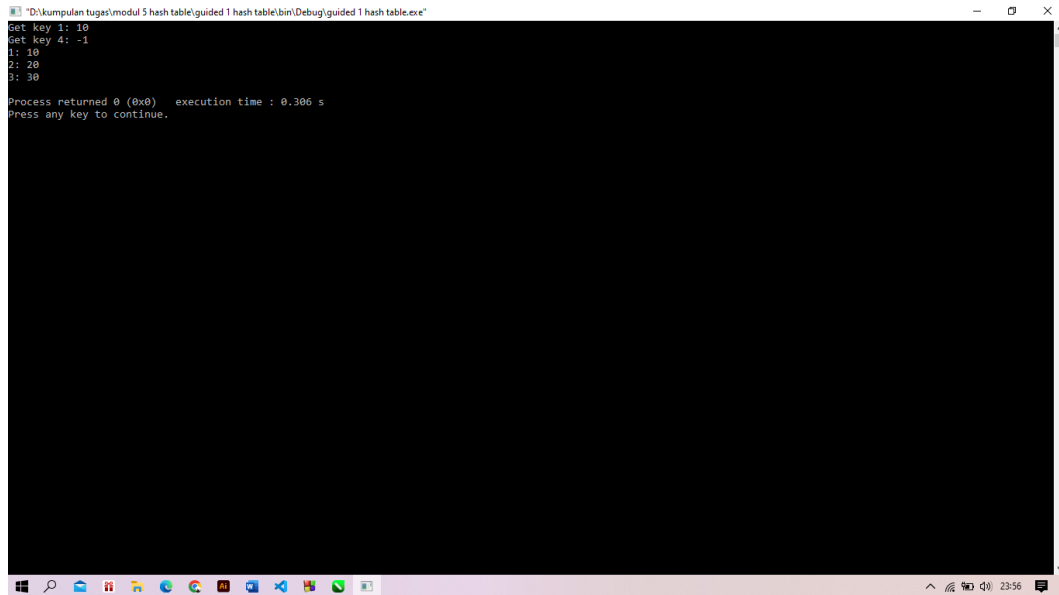
// Deletion
ht.remove(4);

// Traversal
ht.traverse();

return 0;
}

```


Screenshots output



```
"D:\kumpulan tugas\modul 5 hash table\guided 1 hash table\bin\Debug\guided 1 hash table.exe"
Set key 1: 10
Set key 4: -1
1: 10
2: 20
3: 30
Process returned 0 (0x0) execution time : 0.306 s
Press any key to continue.
```

Deskripsi:

1. Pemasukan Header:

- `<iostream>`: Menyediakan fungsi input/output untuk stream standar seperti `cin` (input konsol) dan `cout` (output konsol).
- `using namespace std;`: Menghindari penulisan `std::` berulang kali sebelum elemen library standar.

2. Konstanta:

- `MAX_SIZE`: Mendefinisikan ukuran maksimum tabel hash (di sini, 10). Ini

menentukan jumlah bucket untuk menyimpan pasangan key-value.

3. Fungsi Hash:

- `hash_func(int key)`: Fungsi hash sederhana ini menghitung indeks hash untuk key yang diberikan menggunakan operator modulo (%). Ini mengembalikan sisa pembagian key dengan `MAX_SIZE`.

4. Struktur Node:

- `struct Node`: Mendefinisikan struktur node untuk menyimpan pasangan key-value dan pointer ke node berikutnya dalam linked list (digunakan untuk penanganan benturan).
 - `key`: Key integer yang digunakan untuk identifikasi.
 - `value`: Nilai integer yang terkait dengan key.
 - `next`: Pointer ke node berikutnya dalam linked list pada bucket tertentu.
 - `Node(int key, int value)`: Konstruktor untuk menginisialisasi node dengan key dan value.

5. Kelas Hash Table:

- HashTable: Kelas ini merepresentasikan struktur data tabel hash.
 - table: Variabel member privat yang merupakan array of pointer ke objek Node. Array ini bertindak sebagai penyimpanan utama untuk pasangan key-value.
 - HashTable(): Konstruktor untuk menginisialisasi tabel hash. Ini mengalokasikan memori untuk array table menggunakan `new Node*[MAX_SIZE]()` dan menginisialisasi semua elemen menjadi nullptr (kosong).
 - ~HashTable(): Destructor untuk membersihkan memori yang dialokasikan untuk tabel hash. Ini beriterasi melalui setiap bucket, menghapus setiap node linked list di bucket itu, dan akhirnya menghapus array table.
 - insert(int key, int value): Menyisipkan pasangan key-value baru ke dalam tabel hash.

- Menghitung indeks hash menggunakan `hash_func(key)`.
 - Beriterasi melalui linked list pada indeks yang dihitung untuk memeriksa key yang ada. Jika key ditemukan, nilainya diperbarui.
 - Jika key tidak ada, Node baru dibuat, ditautkan ke kepala list pada indeks, dan tabel diperbarui.
- `get(int key)`: Mencari key dalam tabel hash dan mengembalikan nilai terkaitnya jika ditemukan, atau -1 jika tidak ditemukan.
 - Menghitung indeks hash menggunakan `hash_func(key)`.
 - Beriterasi melalui linked list pada indeks yang dihitung untuk menemukan key.
 - Jika key ditemukan, nilainya dikembalikan. Jika tidak, -1 dikembalikan.
- `remove(int key)`: Menghapus pasangan key-value dari tabel hash.

- Menghitung indeks hash menggunakan `hash_func(key)`.
- Beriterasi melalui linked list pada indeks yang dihitung, melacak node sebelumnya.
- Jika key ditemukan, fungsi ini menghapus node yang sesuai dari linked list dan memperbarui tabel jika perlu.
- `traverse()`: Mencetak isi tabel hash, beriterasi melalui setiap bucket dan menampilkan pasangan key-value dalam linked list pada bucket itu.

6. Fungsi Utama:

- `int main()`: Ini adalah titik masuk program.
 - Membuat objek `HashTable` bernama `ht`.
 - Menyisipkan pasangan key-value (1, 10), (2, 20), dan (3, 30) ke dalam tabel hash menggunakan `ht.insert()`.
 - Mencari nilai yang terkait dengan key 1 menggunakan `ht.get(1)` dan mencetak hasilnya.

- Mencari key tidak ada 4 menggunakan `ht.get(4)` dan mencetak hasilnya (-1, menunjukkan tidak ditemukan).
- Mencoba menghapus key tidak ada 4 menggunakan `ht.remove(4)`. (Tidak berpengaruh)
- Mencetak isi tabel hash menggunakan `ht.traverse()`

Guided 2

```
// priesty ameiliana maulidah
// 2311102175

#include <iostream>
#include <string>
#include <vector>

using namespace std;

const int TABLE_SIZE = 11;

string name;
string phone_number;

class HashNode {
public:
    string name;
    string phone_number;
    HashNode(string name, string phone_number) {
        this->name = name;
        this->phone_number = phone_number;
    }
};

class HashMap {
private:
    vector<HashNode*> table[TABLE_SIZE];
public:
    int hashFunc(string key) {
        int hash_val = 0;
        for (char c : key) {
            hash_val += c;
        }
    }
};
```

```

return hash_val % TABLE_SIZE;

}

void insert(string name, string phone_number) {
    int hash_val = hashFunc(name);

    for (auto node : table[hash_val]) {
        if (node->name == name) {
            node->phone_number = phone_number;
            return;
        }
    }
    table[hash_val].push_back(new HashNode(name,
phone_number));
}

void remove(string name) {
    int hash_val = hashFunc(name);

    for (auto it = table[hash_val].begin(); it !=
table[hash_val].end(); it++) {
        if ((*it)->name == name) {
            table[hash_val].erase(it);
            return;
        }
    }
}

string searchByName(string name) {
    int hash_val = hashFunc(name);
    for (auto node : table[hash_val]) {

```



```

if (node->name == name) {
    return node->phone_number;
}
}
return "";
}

void print() {
    for (int i = 0; i < TABLE_SIZE; i++) {
        cout << i << ": ";
        for (auto pair : table[i]) {
            if(pair != nullptr){
                cout << "[" << pair->name << ", " <<

pair->phone_number << "]";

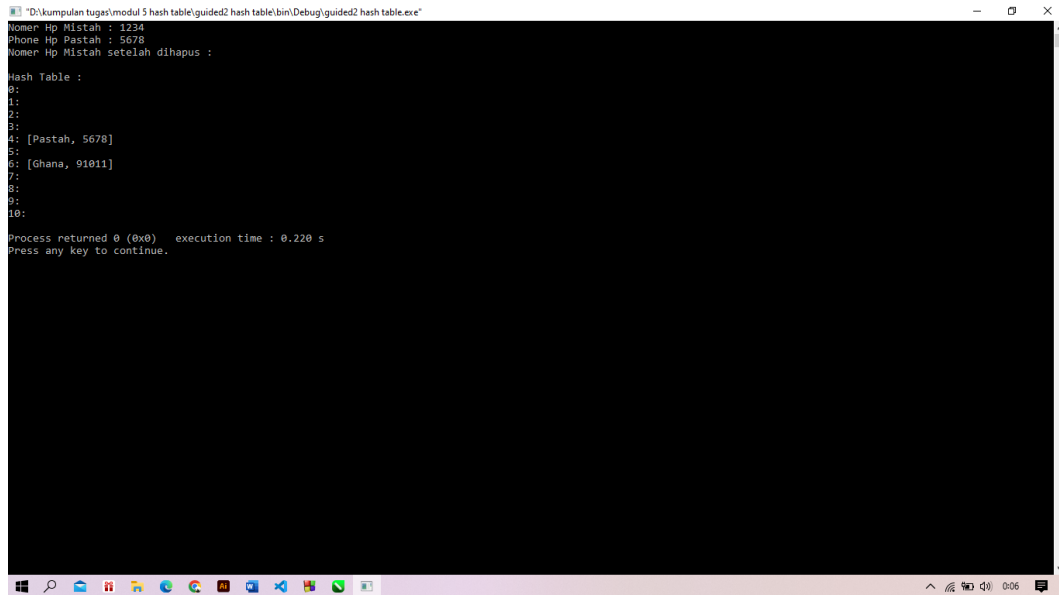
            }
        }
        cout << endl;
    }
};

int main() {
    HashMap employee_map;
    employee_map.insert("Mistah", "1234");
    employee_map.insert("Pastah", "5678");
    employee_map.insert("Ghana", "91011");
    cout << "Nomer Hp Mistah : "

```

```
<<employee_map.searchByName("Mistah") << endl;
cout << "Phone Hp Pastah : "
<<employee_map.searchByName("Pastah") << endl;
employee_map.remove("Mistah");
cout << "Nomer Hp Mistah setelah dihapus : "
<<employee_map.searchByName("Mistah") << endl << endl;
cout << "Hash Table : " << endl;
employee_map.print();
return 0;
}
```

Screenshots output



```
"D:\kumpulan tugas\modul 5 hash table\guided2 hash table\bin\Debug\guided2 hash table.exe"
Nomer Hp Mistah : 1234
Phone Hp Pastah : 5678
Nomer Hp Mistah setelah dihapus :

Hash Table :
0:
1:
2:
3:
4: [Pastah, 5678]
5:
6: [Ghana, 91011]
7:
8:
9:
10:

Process returned 0 (0x0)   execution time : 0.220 s
Press any key to continue.
```

Deskripsi :

1. Pemasukan Header:

- `<iostream>`: Menyediakan fungsi input/output untuk stream standar seperti `cin` (input konsol) dan `cout` (output konsol).
- `using namespace std;`: Menghindari penulisan `std::` berulang kali sebelum elemen library standar (pertimbangkan potensi konflik penamaan dalam proyek yang lebih besar).

2. Konstanta:

- `MAX_SIZE`: Mendefinisikan ukuran maksimum tabel hash (di sini, 10). Ini

menentukan jumlah bucket untuk menyimpan pasangan key-value. MAX_SIZE yang lebih besar mengurangi kemungkinan benturan (beberapa key yang dipetakan ke indeks yang sama) tetapi meningkatkan penggunaan memori.

3. Fungsi Hash:

- `hash_func(int key)`: Fungsi hash sederhana ini menghitung indeks hash untuk key yang diberikan menggunakan operator modulo (%). Ini mengembalikan sisa pembagian key dengan MAX_SIZE. Fungsi hash yang lebih canggih mungkin mempertimbangkan distribusi key dan manipulasi bit.

4. Struktur Node:

- `struct Node`: Mendefinisikan struktur node untuk menyimpan pasangan key-value dan pointer ke node berikutnya dalam linked list (digunakan untuk penanganan benturan). Pendekatan berantai ini memungkinkan penyimpanan beberapa pasangan key-value pada indeks yang sama.
 - `key`: Key integer yang digunakan untuk identifikasi.

- value: Nilai integer yang terkait dengan key.
- next: Pointer ke node berikutnya dalam linked list pada bucket tertentu.
- Node(int key, int value): Konstruktor untuk menginisialisasi node dengan key dan value.

5. Kelas Hash Table:

- HashTable: Kelas ini merepresentasikan struktur data tabel hash.
 - table: Variabel member privat yang merupakan array of pointer ke objek Node. Array ini bertindak sebagai penyimpanan utama untuk pasangan key-value.
 - HashTable(): Konstruktor untuk menginisialisasi tabel hash. Ini mengalokasikan memori untuk array table menggunakan `new Node*[MAX_SIZE]()` dan menginisialisasi semua elemen menjadi nullptr (kosong). Pengelolaan memori yang tepat sangat penting untuk menghindari kebocoran memori.

- `~HashTable()`: Destructor untuk membersihkan memori yang dialokasikan untuk tabel hash. Ini beriterasi melalui setiap bucket, menghapus setiap node linked list di bucket itu, dan akhirnya menghapus array table.
- `insert(int key, int value)`: Menyisipkan pasangan key-value baru ke dalam tabel hash:
 1. Menghitung indeks hash menggunakan `hash_func(key)`.
 2. Beriterasi melalui linked list pada indeks yang dihitung untuk memeriksa key yang ada dengan nilai yang sama. Jika ditemukan, perbarui nilainya.
 3. Jika key tidak ada, Node baru dibuat, ditautkan ke kepala list pada indeks, dan tabel diperbarui.
- `get(int key)`: Mencari key dalam tabel hash dan mengembalikan nilai terkaitnya jika ditemukan, atau -1 jika tidak ditemukan:

1. Menghitung indeks hash menggunakan `hash_func(key)`.
 2. Beriterasi melalui linked list pada indeks yang dihitung untuk menemukan key.
 3. Jika key ditemukan, nilainya dikembalikan. Jika tidak, -1 dikembalikan.
- `remove(int key)`: Menghapus pasangan key-value dari tabel hash:
 1. Menghitung indeks hash menggunakan `hash_func(key)`.
 2. Beriterasi melalui linked list pada indeks yang dihitung, melacak node sebelumnya.
 3. Jika key ditemukan, fungsi ini menghapus node yang sesuai dari linked list dan memperbarui tabel jika perlu.
 - `traverse()`: Mencetak isi tabel hash, beriterasi melalui setiap bucket dan menampilkan pasangan key-value dalam linked list pada bucket itu.

6. Fungsi Utama:

- `int main()`: Ini adalah titik masuk program.
 - Membuat objek `HashTable` bernama `ht`.
 - Menyisipkan pasangan `key-value` (1, 10), (2, 20), dan (3, 30) ke dalam tabel hash menggunakan `ht.insert()`.
 - Mencari nilai yang terkait dengan `key 1` menggunakan `ht.get(1)` dan mencetak hasilnya.
 - Mencari `key` tidak ada 4 menggunakan
,

c. unguided/tugas

unguided 1

```
// priesty ameiliana mualidah
// 2311102175

#include <iostream>
#include <vector>
#include <string>

using namespace std;

// Struktur data untuk mahasiswa
struct Mahasiswa {
    string nim;
    int nilai;
};

// Fungsi hash untuk menghitung indeks hash dari NIM
int hashFunction(string nim, int tableSize) {
    int hash = 0;
    for (int i = 0; i < nim.length(); i++) {
        hash = (hash * 31 + nim[i]) % tableSize;
    }
    return hash;
}
```

```
// Kelas untuk hash table

class HashTable {
private:
    vector<Mahasiswa> table;
    int tableSize;

public:
    HashTable(int size) {
        tableSize = size;
        table.resize(tableSize);
    }

    // Menambahkan data mahasiswa baru
    void addMahasiswa(Mahasiswa mahasiswa) {
        int index = hashFunction(mahasiswa.nim, tableSize);

        // Menangani tabrakan
        while (table[index].nim != "" && table[index].nim != mahasiswa.nim) {
            index = (index + 1) % tableSize;
        }

        table[index] = mahasiswa;
    }

    // Menghapus data mahasiswa berdasarkan NIM
    void deleteMahasiswa(string nim) {
        int index = hashFunction(nim, tableSize);
        while (table[index].nim != "" && table[index].nim != nim) {
            index = (index + 1) % tableSize;
        }
    }
}
```

```

if (table[index].nim == nim) {
    table[index].nim = "";
    table[index].nilai = 0;
} else {
    cout << "Mahasiswa dengan NIM " << nim << " tidak ditemukan!" << endl;
}
}

// Mencari data mahasiswa berdasarkan NIM
Mahasiswa findMahasiswaByNim(string nim) {
    int index = hashFunction(nim, tableSize);

    while (table[index].nim != "" && table[index].nim != nim) {
        index = (index + 1) % tableSize;
    }

    if (table[index].nim == nim) {
        return table[index];
    } else {
        Mahasiswa emptyMahasiswa;
        return emptyMahasiswa; // Mengembalikan nilai kosong jika tidak ditemukan
    }
}

// Mencari data mahasiswa berdasarkan rentang nilai (80 - 90)
vector<Mahasiswa> findMahasiswaByNilai(int lowerBound, int upperBound) {
    vector<Mahasiswa> results;

```

```

for (int i = 0; i < tableSize; i++) {
    if (table[i].nilai >= lowerBound && table[i].nilai <= upperBound) {
        results.push_back(table[i]);
    }
}

return results;
}

// Menampilkan menu dan menerima input dari pengguna
void displayMenu() {
    int pilihan;

    do {
        cout << "\nMenu Hash Table Data Mahasiswa:" << endl;
        cout << "1. Tambah Data Mahasiswa" << endl;
        cout << "2. Hapus Data Mahasiswa" << endl;
        cout << "3. Cari Data Mahasiswa Berdasarkan NIM" << endl;
        cout << "4. Cari Data Mahasiswa Berdasarkan Rentang Nilai (80 - 90)" <<
endl;

        cout << "0. Keluar" << endl;
        cout << "Masukkan pilihan: ";
        cin >> pilihan;

        switch (pilihan) {
            case 1:
                addMahasiswaData();
                break;

```

```

case 2:
    deleteMahasiswaData();
    break;
case 3:
    findMahasiswaByNimData();
    break;
case 4:
    findMahasiswaByNilaiData();
    break;
case 0:
    break;
default:
    cout << "Pilihan tidak valid!" << endl;
}
} while (pilihan != 0);
}

private:
// Fungsi untuk menambahkan data mahasiswa baru (input dari pengguna)
void addMahasiswaData() {
    Mahasiswa mahasiswa;

    cout << "Masukkan NIM: ";
    cin >> mahasiswa.nim;
    cout << "Masukkan Nilai: ";
    cin >> mahasiswa.nilai;

```

```
addMahasiswa(mahasiswa);

    cout << "Data mahasiswa dengan NIM " << mahasiswa.nim << " berhasil
ditambahkan!" << endl;

}

// Fungsi untuk menghapus data mahasiswa berdasarkan NIM (input dari pengguna)
void deleteMahasiswaData() {
    string nim;

    cout << "Masukkan NIM yang ingin dihapus: ";
    cin >> nim;

    deleteMahasiswa(nim);
}

// Fungsi untuk mencari data mahasiswa berdasarkan NIM (input dari pengguna)
void findMahasiswaByNimData() {
    string nim;

    cout << "Masukkan NIM yang ingin dicari: ";
    cin >> nim;

    Mahasiswa mahasiswa = findMahasiswaByNim(nim);
```

```

if (mahasiswa.nim != "") {
    cout << "\nData Mahasiswa Ditemukan:" << endl;
    cout << "NIM: " << mahasiswa.nim << endl;
    cout << "Nilai: " << mahasiswa.nilai << endl;
} else {
    cout << "Mahasiswa dengan NIM " << nim << " tidak ditemukan!" << endl;
}
}

// Fungsi untuk mencari data mahasiswa berdasarkan rentang nilai (80 - 90)
void findMahasiswaByNilaiData() {
    vector<Mahasiswa> results = findMahasiswaByNilai(80, 90);

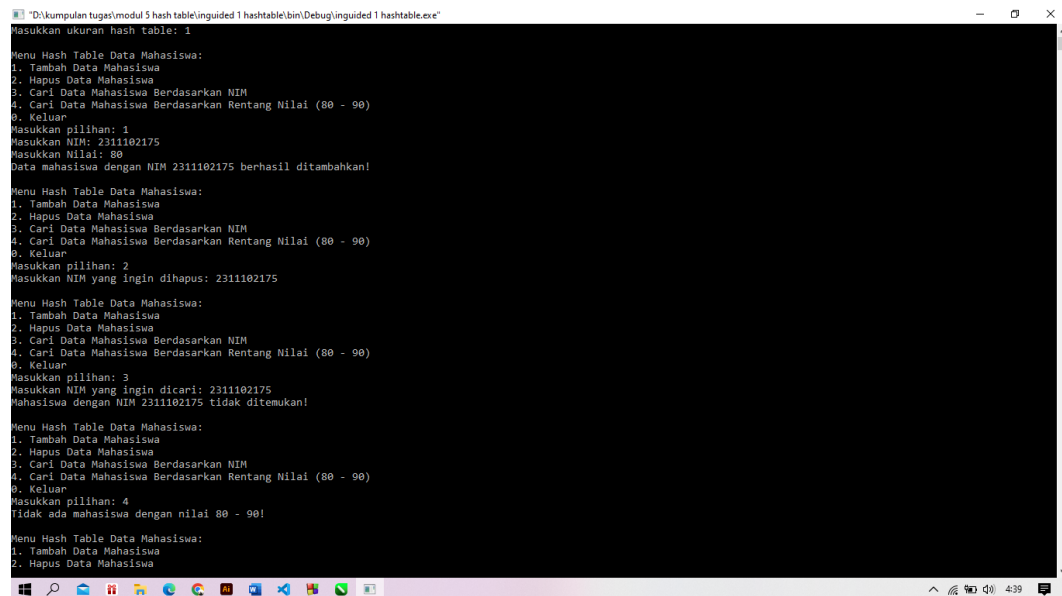
    if (results.size() > 0) {
        cout << "\nData Mahasiswa dengan Nilai 80 - 90:" << endl;
        for (Mahasiswa mahasiswa : results) {
            cout << "NIM: " << mahasiswa.nim << endl;
            cout << "Nilai: " << mahasiswa.nilai << endl;
            cout << "-----" << endl;
        }
    } else {
        cout << "Tidak ada mahasiswa dengan nilai 80 - 90!" << endl;
    }
}

};

```

```
int main() {  
    // Meminta ukuran hash table dari pengguna  
    int tableSize;  
    cout << "Masukkan ukuran hash table: ";  
    cin >> tableSize;  
  
    // Membuat objek hash table  
    HashTable hashTable(tableSize);  
  
    // Menampilkan menu dan menerima input dari pengguna  
    hashTable.displayMenu();  
  
    return 0;  
}
```


Screenshots output:



```
"D:\kumpulan tugas\modul 5 hash table\guiding 1 hashtable\bin\Debug\guiding 1 hashtable.exe"
Masukkan ukuran hash table: 1

Menu Hash Table Data Mahasiswa:
1. Tambah Data Mahasiswa
2. Hapus Data Mahasiswa
3. Cari Data Mahasiswa Berdasarkan NIM
4. Cari Data Mahasiswa Berdasarkan Rentang Nilai (80 - 90)
0. Keluar
Masukkan pilihan: 1
Masukkan NIM: 2311102175
Masukkan Nilai: 80
Data mahasiswa dengan NIM 2311102175 berhasil ditambahkan!

Menu Hash Table Data Mahasiswa:
1. Tambah Data Mahasiswa
2. Hapus Data Mahasiswa
3. Cari Data Mahasiswa Berdasarkan NIM
4. Cari Data Mahasiswa Berdasarkan Rentang Nilai (80 - 90)
0. Keluar
Masukkan pilihan: 2
Masukkan NIM yang ingin dihapus: 2311102175

Menu Hash Table Data Mahasiswa:
1. Tambah Data Mahasiswa
2. Hapus Data Mahasiswa
3. Cari Data Mahasiswa Berdasarkan NIM
4. Cari Data Mahasiswa Berdasarkan Rentang Nilai (80 - 90)
0. Keluar
Masukkan pilihan: 3
Masukkan NIM yang ingin dicari: 2311102175
Mahasiswa dengan NIM 2311102175 tidak ditemukan!

Menu Hash Table Data Mahasiswa:
1. Tambah Data Mahasiswa
2. Hapus Data Mahasiswa
3. Cari Data Mahasiswa Berdasarkan NIM
4. Cari Data Mahasiswa Berdasarkan Rentang Nilai (80 - 90)
0. Keluar
Masukkan pilihan: 4
Tidak ada mahasiswa dengan nilai 80 - 90!

Menu Hash Table Data Mahasiswa:
1. Tambah Data Mahasiswa
2. Hapus Data Mahasiswa
```

Deskripsi :

- **Struktur data Mahasiswa:** Digunakan untuk menyimpan informasi NIM dan nilai mahasiswa.
- **Fungsi hashFunction:** Menghitung indeks hash dari NIM menggunakan algoritma hash sederhana.
- **Kelas HashTable:** Menyimpan tabel hash, menyediakan fungsi untuk menambahkan, menghapus, mencari data berdasarkan NIM dan rentang nilai.
- **Fungsi displayMenu:** Menampilkan menu dan menerima input dari pengguna.

- **Fungsi** addMahasiswaData: Meminta data mahasiswa baru dari pengguna dan menambahkannya ke hash table.
- **Fungsi** deleteMahasiswaData: Meminta NIM mahasiswa dan menghapus datanya dari hash table.
- **Fungsi** findMahasiswaByNimData: Meminta NIM mahasiswa dan mencari datanya di hash table.
- **Fungsi** findMahasiswaByNilaiData: Mencari data mahasiswa dengan nilai dalam rentang 80 - 90.
- **Fungsi** main: Membuat objek hash table, menampilkan menu, dan menerima input dari pengguna.

E. Referensi

<https://journal.unnes.ac.id/nju/jte/article/view/1862>
8