LAPORAN PRAKTIKUM

MODUL 5
HASH TABLE

Disusun oleh:
Nofita Fitriyani
NIM: 2311102001

Dosen Pengampu:
Wahyu Andi Saputra, S.Pd., M.Eng

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS INFORMATIKA
INSTITUT TEKNOLOGI TELKOM PURWOKERTO
PURWOKERTO
2024

BAB I

TUJUAN PRAKTIKUM

a. Mahasiswa mampu menjelaskan definisi dan konsep dari Hash Code
b. Mahasiswa mampu menerapkan Hash Code kedalam pemrograman

BAB II

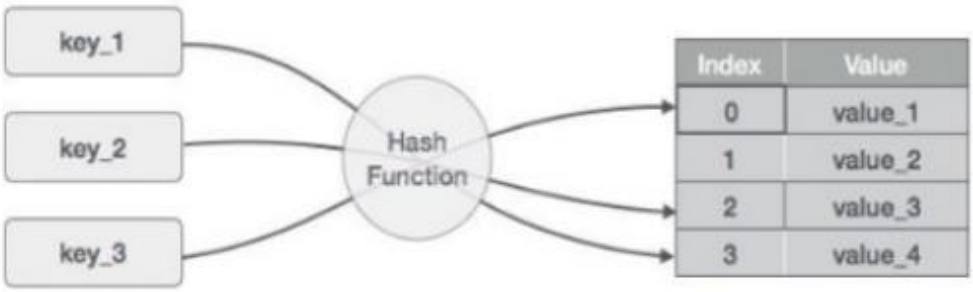
DASAR TEORI

a. Pengertian Hash Table

Hash Table adalah struktur data yang mengorganisir data ke dalam pasangan kunci-nilai. Hash table biasanya terdiri dari dua komponen utama: array (atau vektor) dan fungsi hash. Hashing adalah teknik untuk mengubah rentang nilai kunci menjadi rentang indeks array.

Array menyimpan data dalam slot-slot yang disebut bucket. Setiap bucket dapat menampung satu atau beberapa item data. Fungsi hash digunakan untuk menghasilkan nilai unik dari setiap item data, yang digunakan sebagai indeks array. Dengan cara ini, hash table memungkinkan pencarian data dalam waktu yang konstan (O(1)) dalam kasus terbaik.

Sistem hash table bekerja dengan cara mengambil input kunci dan memetakkannya ke nilai indeks array menggunakan fungsi hash. Kemudian, data disimpan pada posisi indeks array yang dihasilkan oleh fungsi hash. Ketika data perlu dicari, input kunci dijadikan sebagai parameter untuk fungsi hash, dan posisi indeks array yang dihasilkan digunakan untuk mencari data. Dalam kasus hash collision, di mana dua atau lebih data memiliki nilai hash yang sama, hash table menyimpan data tersebut dalam slot yang sama dengan Teknik yang disebut chaining.



b. Fungsi Hash Table

Fungsi hash membuat pemetaan antara kunci dan nilai, hal ini dilakukan melalui penggunaan rumus matematika yang dikenal sebagai fungsi hash. Hasil dari fungsi hash disebut sebagai nilai hash atau hash. Nilai hash adalah representasi dari string karakter asli tetapi biasanya lebih kecil dari aslinya.

c. Operasi Hash Table

1. **Insertion :**

   Memasukkan data baru ke dalam hash table dengan memanggil fungsi hash untuk menentukan posisi bucket yang tepat, dan kemudian menambahkan data ke bucket tersebut.

2. **Deletion :**

   Menghapus data dari hash table dengan mencari data menggunakan fungsi hash, dan kemudian menghapusnya dari bucket yang sesuai.

3. **Searching :**

   Mencari data dalam hash table dengan memasukkan input kunci ke fungsi hash untuk menentukan posisi bucket, dan kemudian mencari data di dalam bucket yang sesuai.

4. **Update :**

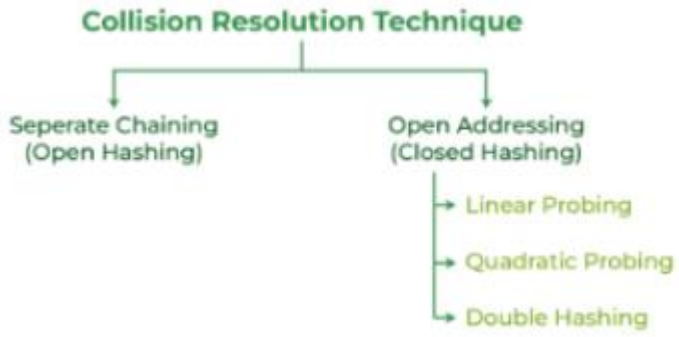
   Memperbarui data dalam hash table dengan mencari data menggunakan fungsi hash, dan kemudian memperbarui data yang ditemukan.

5. **Traversal :**

   Melalui seluruh hash table untuk memproses semua data yang ada dalam table.

d. Collision Resolution

Keterbatasan tabel hash adalah jika dua angka dimasukkan ke dalam fungsi hash menghasilkan nilai yang sama. Hal ini disebut dengan collision. Ada dua teknik untuk menyelesaikan masalah ini diantaranya :



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.

BAB III

GUIDED

1. Guided 1

Source code

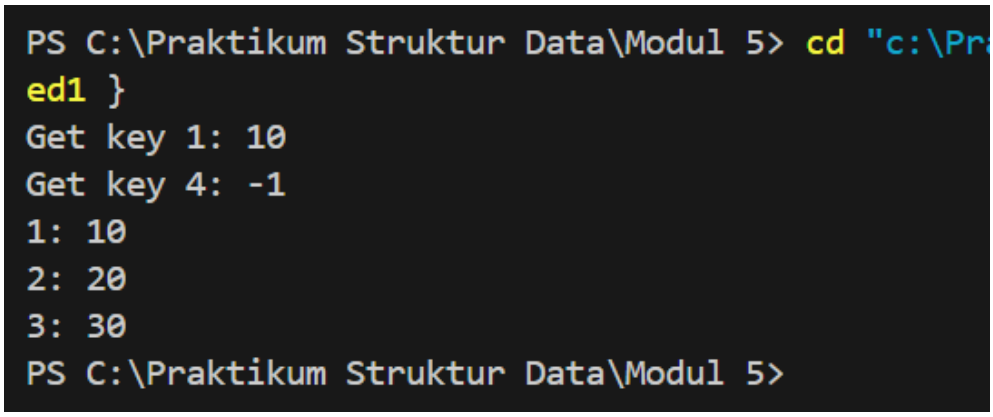
```
#include <iostream>
using namespace std;
const int MAX_SIZE = 10;
// Fungsi hash sederhana
int hash_func(int key)
{
    return key % MAX_SIZE;
}
// Struktur data untuk setiap node
struct Node
{
    int key;
    int value;
    Node *next;
    Node(int key, int value) : key(key), value(value),
                               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;
}
```

Screenshoot program



Deskripsi program

Kode di atas menggunakan array dinamis “table” untuk menyimpan bucket dalam hash table. Setiap bucket diwakili oleh sebuah linked list dengan setiap node merepresentasikan satu item data. Fungsi hash sederhana hanya menggunakan modulus untuk memetakan setiap input kunci ke nilai indeks array.

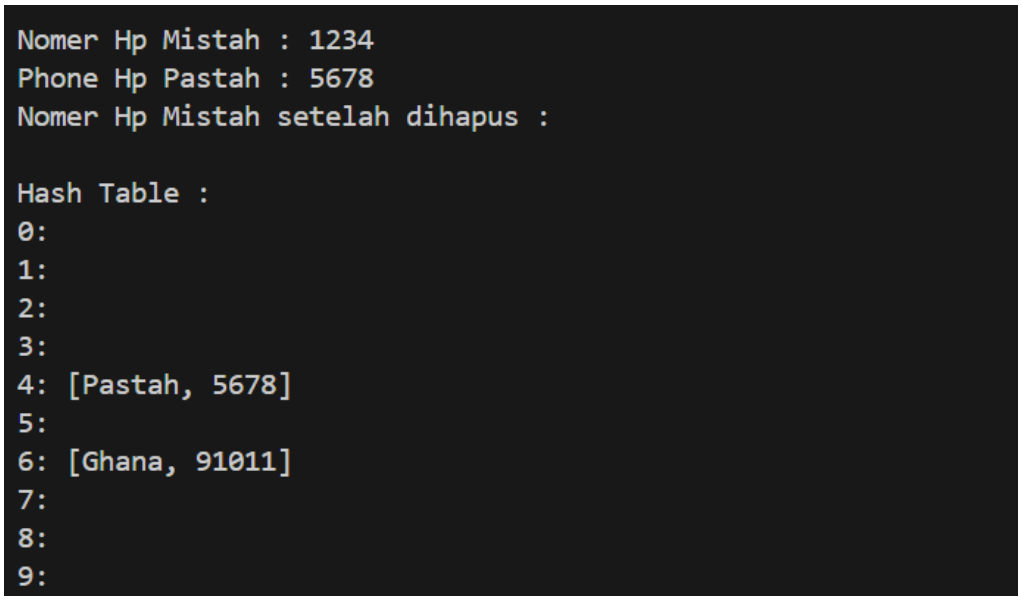
2. Guided 2

```
#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;
}
```

Screenshoot program



Deskripsi program

Pada program di atas, class HashNode merepresentasikan setiap node dalam hash table, yang terdiri dari nama dan nomor telepon karyawan. Class HashMap digunakan untuk mengimplementasikan struktur hash table dengan menggunakan vector yang menampung pointer ke HashNode. Fungsi hashFunc digunakan untuk menghitung nilai hash dari nama karyawan yang diberikan, dan fungsi insert digunakan untuk menambahkan data baru ke dalam hash table. Fungsi remove digunakan untuk menghapus data dari hash table, dan fungsi searchByName digunakan untuk mencari nomor telepon dari karyawan dengan nama yang diberikan.

LATIHAN KELAS - UNGUIDED

1. Implementasikan hash table untuk menyimpan data mahasiswa. Setiap mahasiswa memiliki NIM dan nilai. Implementasikan fungsi untuk menambahkan data baru,menghapus data, mencari data berdasarkan NIM, dan mencari data berdasarkan nilai. Dengan ketentuan :
   a. Setiap mahasiswa memiliki NIM dan nilai.
   b. Program memiliki tampilan pilihan menu berisi poin C.
   c. Implementasikan fungsi untuk menambahkan data baru, menghapus data, mencari data berdasarkan NIM, dan mencari data berdasarkan rentang nilai (80 – 90).

Source code

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

using namespace std;

// Class Mahasiswa untuk merepresentasikan entitas mahasiswa
class Mahasiswa {
public:
    string nim;
    int nilai;

    Mahasiswa(string nim, int nilai) {
        this->nim = nim;
        this->nilai = nilai;
    }
};

// Class HashTable untuk implementasi hash table
class HashTable {
private:
    static const int table_size = 10; // Ukuran tabel hash
```

```
    vector<Mahasiswa *> table[table_size];

public:
    // Fungsi hash
    int hash_function(string nim) {
        int sum = 0;
        for (char c : nim) {
            sum += c;
        }
        return sum % table_size;
    }

    // Menambah data mahasiswa
    void addData2311102001(Mahasiswa *mahasiswa) {
        int index = hash_function(mahasiswa->nim);
        table[index].push_back(mahasiswa);
    }

    // Menghapus data mahasiswa berdasarkan NIM
    void delData(string nim) {
        int index = hash_function(nim);
        for (auto it = table[index].begin(); it != table[index].end(); ++it) {
            if ((*it)->nim == nim) {
                delete *it;
                table[index].erase(it);
                break;
            }
        }
    }

    // Mencari data mahasiswa berdasarkan NIM
    Mahasiswa *searchDataByNIM(string nim) {
        int index = hash_function(nim);
```

```
        for (Mahasiswa *mahasiswa : table[index]) {
            if (mahasiswa->nim == nim) {
                return mahasiswa;
            }
        }
        return nullptr;
    }

    // Mencari data mahasiswa berdasarkan rentang nilai (80 - 90)
    vector<Mahasiswa *> searchDataByRange(int nilai_min, int nilai_max) {
        vector<Mahasiswa *> hasil_pencarian;
        for (int i = 0; i < table_size; i++) {
            for (Mahasiswa *mahasiswa : table[i]) {
                if (mahasiswa->nilai >= nilai_min && mahasiswa->nilai <= nilai_max) {
                    hasil_pencarian.push_back(mahasiswa);
                }
            }
        }
        return hasil_pencarian;
    }
};

// Fungsi main
int main() {
    HashTable hash_table;
    while (true) {
        cout << "\nPilihan Menu:" << endl;
        cout << "1. Tambah Data Mahasiswa" << endl;
        cout << "2. Hapus Data Mahasiswa" << endl;
        cout << "3. Cari Mahasiswa Berdasarkan NIM" << endl;
```

```
        cout << "4. Cari Mahasiswa Berdasarkan Rentang Nilai (80-
90)" << endl;
        cout << "5. Keluar" << endl;
        int pilihan;
        cout << "Masukkan pilihan Anda: ";
        cin >> pilihan;

        if (pilihan == 1) {
            string nim;
            int nilai;
            cout << "Masukkan NIM: ";
            cin >> nim;
            cout << "Masukkan nilai: ";
            cin >> nilai;
            Mahasiswa  *mahasiswa_baru  =  new  Mahasiswa(nim,
nilai);
            hash_table.addData2311102001(mahasiswa_baru);
            cout  << "Data mahasiswa berhasil ditambahkan." <<
endl;
        }
        else if (pilihan == 2) {
            string nim;
            cout << "Masukkan NIM : ";
            cin >> nim;
            hash_table.delData(nim);
            cout  << "Data mahasiswa dengan NIM " << nim << "
telah dihapus." << endl;
        }
        else if (pilihan == 3) {
            string nim;
            cout << "Masukkan NIM : ";
            cin >> nim;
            Mahasiswa                *mahasiswa                =
hash_table.searchDataByNIM(nim);
```

```
            if (mahasiswa != nullptr) {
                cout << "Mahasiswa dengan NIM " << nim << "
ditemukan. Nilai: " << mahasiswa->nilai << endl;
            } else {
                cout << "Mahasiswa dengan NIM " << nim << " tidak
ditemukan." << endl;
            }
        }
        else if (pilihan == 4) {
            vector<Mahasiswa    *>    mahasiswa_ditemukan    =
hash_table.searchDataByRange(80, 90);
            if (!mahasiswa_ditemukan.empty()) {
                cout << "Mahasiswa dengan nilai antara 80 dan
90:" << endl;
                for (Mahasiswa *mahasiswa : mahasiswa_ditemukan)
{
                    cout << "NIM: " << mahasiswa->nim << " Nilai:
" << mahasiswa->nilai << endl;
                }
            } else {
                cout << "Tidak ada mahasiswa dengan nilai antara
80 dan 90." << endl;
            }
        }
        else if (pilihan == 5) {
            cout << "Program selesai." << endl;
            break;
        }
        else {
            cout << "Pilihan tidak valid. Silakan masukkan
pilihan yang benar." << endl;
        }
    }
    return 0;
```

```
}
```

Screenshoot program

- **Tampilan Menu**

```
PS C:\Praktikum Struktur Data\Modul 5> cd "c:\Praktikum St
unguided1 }

Pilihan Menu:
1. Tambah Data Mahasiswa
2. Hapus Data Mahasiswa
3. Cari Mahasiswa Berdasarkan NIM
4. Cari Mahasiswa Berdasarkan Rentang Nilai (80-90)
5. Keluar
Masukkan pilihan Anda:
```

- **Tambah Data Mahasiswa**

```
Pilihan Menu:
1. Tambah Data Mahasiswa
2. Hapus Data Mahasiswa
3. Cari Mahasiswa Berdasarkan NIM
4. Cari Mahasiswa Berdasarkan Rentang Nilai (80-90)
5. Keluar
Masukkan pilihan Anda: 1
Masukkan NIM: 2311102001
Masukkan nilai: 90
Data mahasiswa berhasil ditambahkan.
```

- **Hapus Data Mahasiswa**

```
Pilihan Menu:
1. Tambah Data Mahasiswa
2. Hapus Data Mahasiswa
3. Cari Mahasiswa Berdasarkan NIM
4. Cari Mahasiswa Berdasarkan Rentang Nilai (80-90)
5. Keluar
Masukkan pilihan Anda: 2
Masukkan NIM : 2311102005
Data mahasiswa dengan NIM 2311102005 telah dihapus.
```

- **Cari Mahasiswa Berdasarkan NIM**

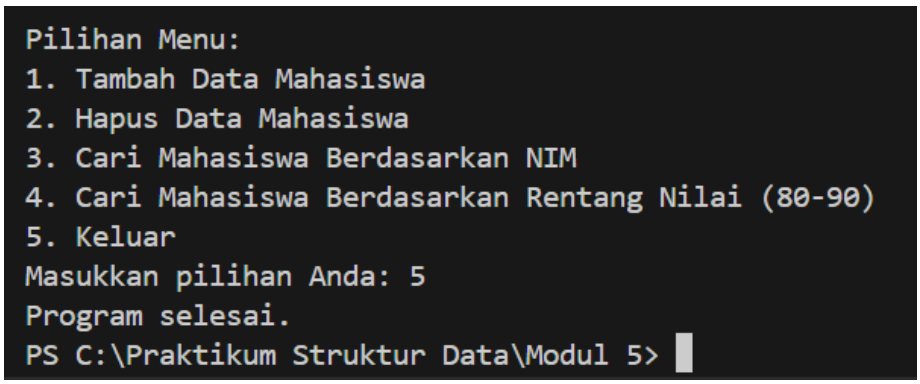
```
Pilihan Menu:
1. Tambah Data Mahasiswa
2. Hapus Data Mahasiswa
3. Cari Mahasiswa Berdasarkan NIM
4. Cari Mahasiswa Berdasarkan Rentang Nilai (80-90)
5. Keluar
Masukkan pilihan Anda: 3
Masukkan NIM : 2311102005
Mahasiswa dengan NIM 2311102005 tidak ditemukan.
```

```
Pilihan Menu:
1. Tambah Data Mahasiswa
2. Hapus Data Mahasiswa
3. Cari Mahasiswa Berdasarkan NIM
4. Cari Mahasiswa Berdasarkan Rentang Nilai (80-90)
5. Keluar
Masukkan pilihan Anda: 3
Masukkan NIM : 2311102001
Mahasiswa dengan NIM 2311102001 ditemukan. Nilai: 90
```

- **Cari Mahasiswa Berdasarkan Rentang Nilai (80-90)**

```
Pilihan Menu:
1. Tambah Data Mahasiswa
2. Hapus Data Mahasiswa
3. Cari Mahasiswa Berdasarkan NIM
4. Cari Mahasiswa Berdasarkan Rentang Nilai (80-90)
5. Keluar
Masukkan pilihan Anda: 4
Mahasiswa dengan nilai antara 80 dan 90:
NIM: 2311102001 Nilai: 90
NIM: 2311102001 Nilai: 89
```

- **Keluar**



Deskripsi program

Program di atas adalah implementasi sederhana dari tabel hash untuk menyimpan, menghapus, dan mencari data mahasiswa berdasarkan NIM (Nomor Induk Mahasiswa) dan rentang nilai. Program ini memiliki kelas Mahasiswa yang merepresentasikan entitas mahasiswa dengan atribut nim dan nilai. Selain itu, terdapat kelas HashTable yang mengimplementasikan tabel hash dengan ukuran tetap 10. Kelas ini menyediakan fungsi untuk menambah data mahasiswa addData2311102001 menghapus data mahasiswa berdasarkan NIM delData mencari data mahasiswa berdasarkan NIM searchDataByNIM dan mencari mahasiswa berdasarkan rentang nilai searchDatabyRange.

Dalam fungsi main, program menampilkan menu interaktif kepada pengguna untuk memilih salah satu dari lima opsi: menambah data mahasiswa, menghapus data mahasiswa, mencari mahasiswa berdasarkan NIM, mencari mahasiswa berdasarkan rentang nilai 80-90, dan keluar dari program. Berdasarkan pilihan pengguna, program akan meminta input yang relevan (seperti NIM dan nilai) dan melakukan operasi yang sesuai pada tabel hash, menampilkan hasilnya ke layar. Program ini berjalan dalam loop hingga pengguna memilih untuk keluar.

BAB IV

KESIMPULAN

Hash table adalah struktur data yang efektif untuk mengorganisir data ke dalam pasangan kunci-nilai. Dengan menggunakan fungsi hash, hash table memetakan kunci ke indeks array, memungkinkan operasi penyisipan, penghapusan, pencarian, pembaruan, dan traversing data dilakukan dengan cepat. Fungsi hash mengubah kunci menjadi nilai hash yang digunakan sebagai indeks untuk menyimpan atau mencari data dalam array.

Dalam implementasi hash table, ada tantangan yang dikenal sebagai hash collision, yaitu ketika dua atau lebih kunci menghasilkan nilai hash yang sama. Untuk mengatasi masalah ini, ada dua metode utama: open hashing (chaining) dan closed hashing. Open hashing mengatasi collision dengan menyimpan data yang memiliki nilai hash yang sama dalam linked list, sementara closed hashing (seperti linear probing, quadratic probing, dan double hashing) mencoba menemukan slot kosong dalam array untuk menyimpan data.

Hash table menawarkan efisiensi tinggi dalam operasi pencarian dan penyimpanan data, namun memerlukan teknik penanganan collision yang tepat untuk memastikan performa optimal. Dengan memahami konsep dasar, fungsi, operasi, dan metode resolusi collision, hash table dapat digunakan secara efektif dalam berbagai aplikasi pemrograman untuk mengelola data dengan cepat dan efisien.