LAPORAN PRAKTIKUM

MODUL V
HASH TABLE

Disusun oleh:
Maulana Ghani Rolanda
NIM: 2311102012

Dosen Pengampu:
Muhammad Afrizal Amrustian, S. Kom., M. Kom

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS INFORMATIKA
INSTITUT TEKNOLOGI TELKOM PURWOKERTO
PURWOKERTO
2023

BAB I

TUJUAN PRAKTIKUM

1. Mahasiswa mampu menjelaskan definisi dan konsep dari Hash Code
2. Mahasiswa mampu menerapkan Hash Code kedalam pemrograman

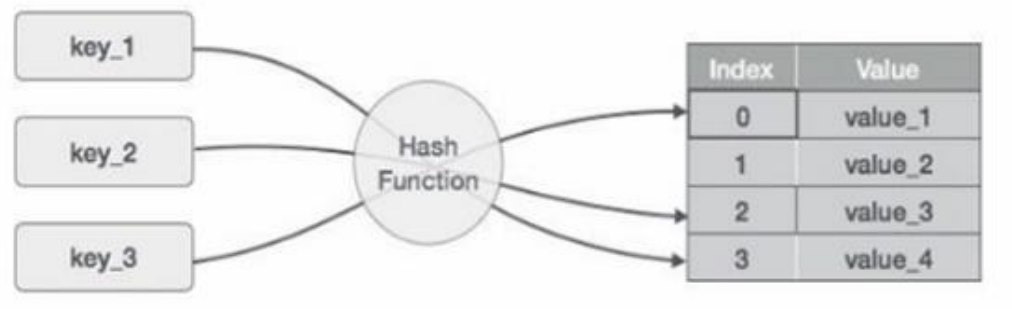
BAB II

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:

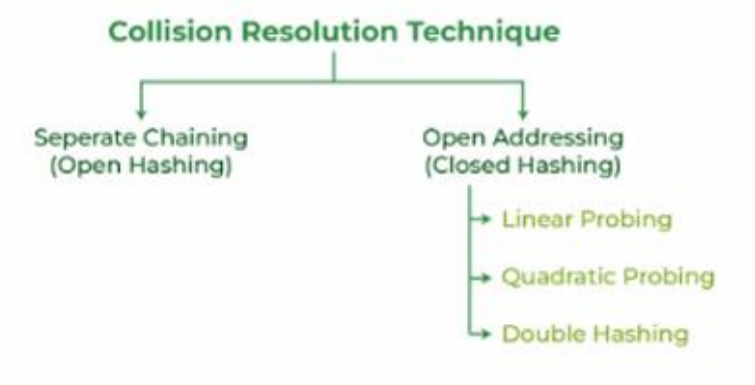
Memperbarui data dalam hash table dengan mencari data menggunakan fungsi hash, dan kemudian memperbarui data yang ditemukan.

5. Traversal:

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

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. OpenHashing (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, ...)

   ● DoubleHashing Pada saat terjadi collision, terdapat fungsi hash yang kedua untuk menentukan posisinya kembali.

BAB III

GUIDED

1. Guided 1

Source code

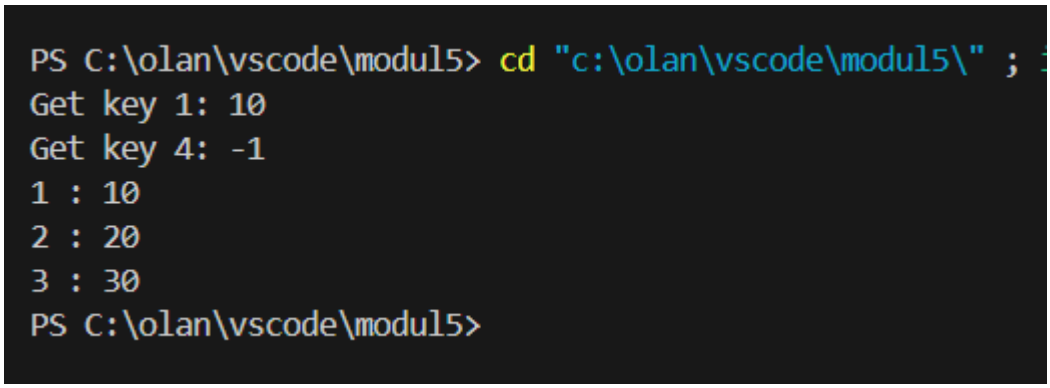
```
#include <iostream>

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

Program di atas adalah implementasi sederhana dari tabel hash (hash table) menggunakan chaining untuk mengatasi konflik (collision).

Komponen Program

1. **Konstanta MAX_SIZE**:
    - Konstanta **MAX_SIZE** ditetapkan menjadi 10. Ini adalah ukuran dari tabel hash, yaitu jumlah bucket atau slot yang tersedia dalam tabel.
2. **Fungsi hash_func**:
    - Fungsi ini adalah fungsi hash sederhana yang mengambil kunci (key) sebagai input dan mengembalikan indeks dalam tabel hash dengan cara mengambil sisa pembagian (**key % MAX_SIZE**).
3. **Struktur Node**:
    - Struktur ini merepresentasikan elemen di dalam tabel hash. Setiap **Node** memiliki kunci (key), nilai (value), dan pointer **next** yang menunjuk ke node berikutnya dalam list chaining.
4. **Kelas HashTable**:
    - Kelas ini mengimplementasikan tabel hash dengan array pointer ke node. Metode-metodenya termasuk:

- **Constructor**: Menginisialisasi tabel hash dengan array pointer yang berukuran **MAX_SIZE**.
- **Destructor**: Menghapus semua node untuk mencegah kebocoran memori.
- **insert**: Menambahkan key-value pair ke tabel hash.
- **get**: Mengambil nilai yang sesuai dengan kunci yang diberikan.
- **remove**: Menghapus key-value pair dari tabel hash berdasarkan kunci yang diberikan.
- **traverse**: Melakukan traversal dan mencetak semua key-value pair dalam tabel hash.

Penjelasan Fungsi Utama

1. **Insertion (insert)**:
    - Fungsi ini menambahkan key-value pair ke dalam tabel hash. Jika kunci sudah ada, nilai diperbarui. Jika tidak, node baru dibuat dan ditambahkan ke awal list chaining pada indeks yang sesuai.
2. **Searching (get)**:
    - Fungsi ini mencari nilai berdasarkan kunci yang diberikan. Jika kunci ditemukan, nilai dikembalikan. Jika tidak, mengembalikan **-1**.
3. **Deletion (remove)**:
    - Fungsi ini menghapus key-value pair dari tabel hash. Jika kunci ditemukan, node dihapus dan pointer diatur ulang untuk mempertahankan list chaining.
4. **Traversal (traverse)**:
    - Fungsi ini mencetak semua key-value pair dalam tabel hash dengan melakukan traversal pada setiap bucket dan node di dalam list chaining.

Fungsi main

- Pada fungsi **main**, contoh penggunaan **HashTable** ditunjukkan:
    - **Insertion**: Menambahkan beberapa pasangan kunci-nilai.
    - **Searching**: Mencari nilai untuk kunci tertentu.
    - **Deletion**: Mencoba menghapus kunci (dalam contoh ini kunci **4** yang tidak ada).
    - **Traversal**: Mencetak semua pasangan kunci-nilai yang ada dalam tabel hash.

2. Guided 2

Source code

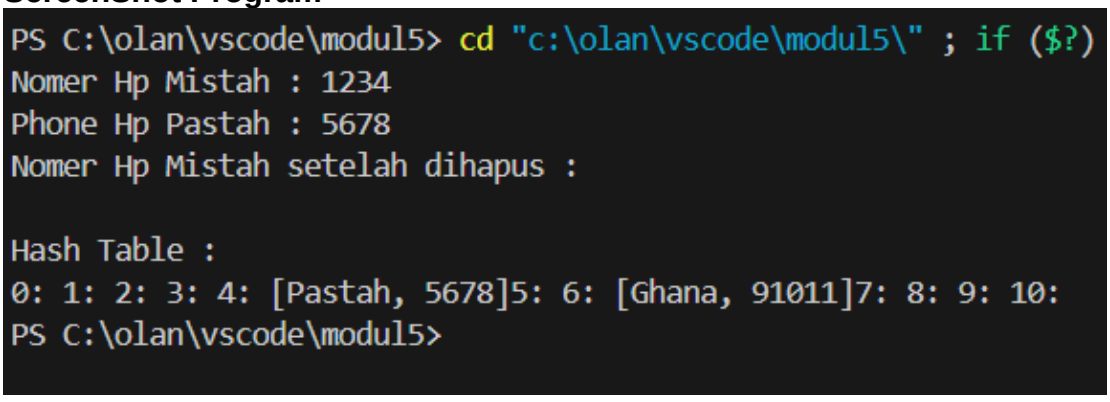
```
#include <iostream>
#include <string>
#include <vector>

using namespace std;

// ukuran tabel hash
const int TABLE_SIZE = 11;

string name; //deklarasi variabel string name
string phone_number; //deklarasi variabel string phone_number

// Struktur Data Untuk Setiap Node
class HashNode
{
//deklarasi variabel name dan phone_number
public:
    string name;
    string phone_number;

    HashNode(string name, string phone_number)
    {
        this->name = name;
        this->phone_number = phone_number;
    }
};
```

```
// Class HashMap
class HashMap
{
private:
    vector<HashNode*> table[TABLE_SIZE];

public:
    // Fungsi Hash Sederhana
    int hashFunc(string key)
    {
        int hash_val = 0;
        for (char c : key)
        {
            hash_val += c;
        }
        return hash_val % TABLE_SIZE;
    }

    // Tambah data
    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));
    }

    // Hapus data
    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;
            }
        }
    }

    // Cari data berdasarkan nama
    string searchByName(string name)
    {
        int hash_val = hashFunc(name);
        for (auto node : table[hash_val])
        {
            if (node->name == name)
            {
                return node->phone_number;
            }
        }
        return "";
    }

    // Cetak data
    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 << "]";
                }
            }
        }
    }
};

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;
}
```

ScreenShot Program



Deskripsi Program

Program di atas adalah implementasi tabel hash (hash table) untuk menyimpan dan mengelola data pasangan nama dan nomor telepon. Program ini menggunakan teknik chaining untuk mengatasi konflik (collision) dengan tabel hash yang berukuran 11.

Komponen Program

1. **Konstanta TABLE_SIZE**:
    - Konstanta ini menentukan ukuran tabel hash, yaitu 11.
2. **Kelas HashNode**:
    - Kelas ini mewakili setiap node dalam tabel hash dengan dua atribut: **name** dan **phone_number**.
    - Konstruktor **HashNode** digunakan untuk menginisialisasi objek node dengan nama dan nomor telepon.
3. **Kelas HashMap**:
    - Kelas ini mengimplementasikan tabel hash dengan array **vector** dari pointer ke **HashNode**.
    - Metode-metode yang tersedia dalam kelas ini adalah:
        - **hashFunc**: Menghitung nilai hash berdasarkan jumlah karakter ASCII dari string kunci.
        - **insert**: Menambahkan pasangan nama dan nomor telepon ke dalam tabel hash. Jika nama sudah ada, nomor telepon diperbarui.
        - **remove**: Menghapus pasangan nama dan nomor telepon dari tabel hash berdasarkan nama.
        - **searchByName**: Mencari nomor telepon berdasarkan nama. Jika tidak ditemukan, mengembalikan string kosong.
        - **print**: Mencetak semua pasangan nama dan nomor telepon dalam tabel hash.

Penjelasan Fungsi Utama

1. **Hash Function (hashFunc)**:

- Fungsi ini menghitung nilai hash dengan menjumlahkan nilai ASCII dari setiap karakter dalam string kunci, kemudian mengambil sisa pembagian dengan ukuran tabel (**TABLE_SIZE**).

2. **Insertion (insert)**:
    - Fungsi ini menambahkan pasangan nama dan nomor telepon ke tabel hash. Jika nama sudah ada, nomor telepon diperbarui. Jika tidak, node baru dibuat dan ditambahkan ke vektor pada indeks yang sesuai.
3. **Deletion (remove)**:
    - Fungsi ini menghapus pasangan nama dan nomor telepon dari tabel hash. Jika nama ditemukan, node dihapus dari vektor pada indeks yang sesuai.
4. **Searching (searchByName)**:
    - Fungsi ini mencari nomor telepon berdasarkan nama. Jika nama ditemukan, nomor telepon dikembalikan. Jika tidak, mengembalikan string kosong.
5. **Traversal (print)**:
    - Fungsi ini mencetak semua pasangan nama dan nomor telepon dalam tabel hash. Setiap indeks tabel dicetak, diikuti dengan pasangan nama dan nomor telepon yang ada pada indeks tersebut.

Fungsi main

- Pada fungsi **main**, contoh penggunaan **HashMap** ditunjukkan:
    - **Insertion**: Menambahkan beberapa pasangan nama dan nomor telepon.
    - **Searching**: Mencari nomor telepon untuk nama tertentu.
    - **Deletion**: Menghapus nama tertentu dari tabel hash.
    - **Traversal**: Mencetak semua pasangan nama dan nomor telepon yang ada dalam tabel hash.

LATIHAN KELAS - UNGUIDED

1. **Unguided 1**

   Source code

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

using namespace std;

// Struktur Data Untuk Setiap Node
struct Student {
    string nim;
    string name;
    int nilai;
    Student(string nim, string name, int nilai) : nim(nim),
name(name), nilai(nilai) {}
};

// Class HashMap
class HashMap {
private:
    static const int TABLE_SIZE = 10; // Ukuran tabel hash
    vector<Student*> table[TABLE_SIZE];

public:
    // Fungsi Hash Sederhana
    int hashFunc(string key) {
        int hash_val = 0;
        for (char c : key) {
            hash_val += c;
        }
        return hash_val % TABLE_SIZE;
    }

    // Tambah data mahasiswa baru
```

```
    void insert(string nim, string name, int nilai) {
        int hash_val = hashFunc(nim);
        table[hash_val].push_back(new Student(nim, name, nilai));
    }

    // Hapus data mahasiswa
    void remove(string nim) {
        int hash_val = hashFunc(nim);
        for (auto it = table[hash_val].begin(); it != 
table[hash_val].end(); it++) {
            if ((*it)->nim == nim) {
                table[hash_val].erase(it);
                return;
            }
        }
    }

    // Cari data mahasiswa berdasarkan NIM
    Student* searchByNIM(string nim) {
        int hash_val = hashFunc(nim);
        for (auto student : table[hash_val]) {
            if (student->nim == nim) {
                return student;
            }
        }
        return nullptr; // Mengembalikan nullptr jika tidak 
ditemukan
    }

    // Cari data mahasiswa berdasarkan rentang nilai (80-90)
    vector<Student*> searchByRange(int min, int max) {
        vector<Student*> result;
        for (int i = 0; i < TABLE_SIZE; i++) {
            for (auto student : table[i]) {
                if (student->nilai >= min && student->nilai <= 
max) {
                    result.push_back(student);
                }
            }
        }
        return result;
```

```
    }
};

int main() {
    HashMap mahasiswa_map;
    int choice;
    string nim, name;
    int nilai;
    while (true) {
        cout << "Menu:\n";
        cout << "1. Tambah data mahasiswa\n";
        cout << "2. Hapus data mahasiswa\n";
        cout << "3. Cari data mahasiswa berdasarkan NIM\n";
        cout << "4. Cari data mahasiswa berdasarkan rentang nilai
(80-90)\n";
        cout << "5. Keluar\n";
        cout << "Pilih: ";
        cin >> choice;
        switch (choice) {
            case 1:
                cout << "Masukkan NIM: ";
                cin >> nim;
                cout << "Masukkan nama: ";
                cin.ignore(); // Clear input buffer
                getline(cin, name);
                cout << "Masukkan nilai: ";
                cin >> nilai;
                mahasiswa_map.insert(nim, name, nilai);
                break;
            case 2:
                cout << "Masukkan NIM yang akan dihapus: ";
                cin >> nim;
                mahasiswa_map.remove(nim);
                break;
            case 3:
                cout << "Masukkan NIM yang dicari: ";
                cin >> nim;
                {
                    Student* student =
mahasiswa_map.searchByNIM(nim);
                    if (student != nullptr) {
```

```
                        cout << "Nama: " << student->name << ",
NIM: " << student->nim << ", Nilai: " << student->nilai << endl;
                    } else {
                        cout << "Mahasiswa dengan NIM " << nim <<
" tidak ditemukan.\n";
                    }
                }
                break;
            case 4:
                cout << "Mahasiswa dengan nilai antara 80 -
90:\n";
                {
                    vector<Student*> students =
mahasiswa_map.searchByRange(80, 90);
                    if (students.empty()) {
                        cout << "Tidak ada mahasiswa dengan nilai
dalam rentang tersebut.\n";
                    } else {
                        for (const auto& student : students) {
                            cout << "Nama: " << student->name <<
", NIM: " << student->nim << ", Nilai: " << student->nilai <<
endl;
                        }
                    }
                }
                break;
            case 5:
                cout << "Terima kasih!\n";
                return 0;
            default:
                cout << "Pilihan tidak valid.\n";
        }
    }
}
```

Screenshoot program

```
PS C:\olan\vscode\modul5> cd "c:\olan\vscode\modul5\" ; i
Menu:
1. Tambah data mahasiswa
2. Hapus data mahasiswa
3. Cari data mahasiswa berdasarkan NIM
4. Cari data mahasiswa berdasarkan rentang nilai (80-90)
5. Keluar
Pilih: 1
Masukkan NIM: 2311102012
Masukkan nama: Maulana Ghani Rolanda
Masukkan nilai: 83
```

```
Menu:
1. Tambah data mahasiswa
2. Hapus data mahasiswa
3. Cari data mahasiswa berdasarkan NIM
4. Cari data mahasiswa berdasarkan rentang nilai (80-90)
5. Keluar
Pilih: 3
Masukkan NIM yang dicari: 2311102012
Nama: Maulana Ghani Rolanda, NIM: 2311102012, Nilai: 83
```

```
Menu:
1. Tambah data mahasiswa
2. Hapus data mahasiswa
3. Cari data mahasiswa berdasarkan NIM
4. Cari data mahasiswa berdasarkan rentang nilai (80-90)
5. Keluar
Pilih: 4
Mahasiswa dengan nilai antara 80 - 90:
Nama: Maulana Ghani Rolanda, NIM: 2311102012, Nilai: 83
```

```
Menu:
1. Tambah data mahasiswa
2. Hapus data mahasiswa
3. Cari data mahasiswa berdasarkan NIM
4. Cari data mahasiswa berdasarkan rentang nilai (80-90)
5. Keluar
Pilih: 2
Masukkan NIM yang akan dihapus: 2311102012
```

Deskripsi program

Program di atas adalah implementasi sederhana dari sebuah aplikasi untuk mengelola data mahasiswa menggunakan tabel hash

Komponen Program:

1. **Struktur Student**:
    - Struktur data ini merepresentasikan setiap mahasiswa dengan atribut **nim**, **name**, dan **nilai**.
2. **Kelas HashMap**:
    - Kelas ini menerapkan tabel hash untuk menyimpan data mahasiswa.
    - Memiliki atribut array **vector** dari pointer ke **Student** untuk menampung data.
    - Metode-metode yang dimilikinya adalah:
        - **hashFunc**: Menghitung nilai hash dari sebuah string (NIM) untuk menentukan indeks di dalam tabel hash.
        - **insert**: Menambahkan data mahasiswa baru ke dalam tabel hash.
        - **remove**: Menghapus data mahasiswa dari tabel hash berdasarkan NIM.
        - **searchByNIM**: Mencari data mahasiswa berdasarkan NIM.
        - **searchByRange**: Mencari data mahasiswa berdasarkan rentang nilai tertentu.
3. **Fungsi main**:
    - Fungsi utama program yang menampilkan pilihan menu dan menerima input dari pengguna.

- Memungkinkan pengguna untuk melakukan operasi seperti menambahkan data mahasiswa baru, menghapus data, mencari data berdasarkan NIM, dan mencari data berdasarkan rentang nilai.
- Setiap pilihan menu memanggil metode yang sesuai dari kelas **HashMap**.

Fungsionalitas Utama:

- **Menambah Data Mahasiswa**:
  - Pengguna diminta untuk memasukkan NIM, nama, dan nilai mahasiswa baru.
  - Data mahasiswa tersebut kemudian ditambahkan ke dalam tabel hash menggunakan metode **insert** dari kelas **HashMap**.
- **Menghapus Data Mahasiswa**:
  - Pengguna diminta untuk memasukkan NIM mahasiswa yang ingin dihapus.
  - Data mahasiswa dengan NIM tersebut kemudian dihapus dari tabel hash menggunakan metode **remove** dari kelas **HashMap**.
- **Mencari Data Mahasiswa berdasarkan NIM**:
  - Pengguna diminta untuk memasukkan NIM yang ingin dicari.
  - Program mencari data mahasiswa dengan NIM tersebut menggunakan metode **searchByNIM** dari kelas **HashMap**, dan jika ditemukan, mencetak detailnya (nama, NIM, nilai).
- **Mencari Data Mahasiswa berdasarkan Rentang Nilai**:
  - Program mencari dan mencetak data mahasiswa yang memiliki nilai dalam rentang tertentu (80-90) menggunakan metode **searchByRange** dari kelas **HashMap**.
- **Keluar dari Program**:
  - Pengguna dapat memilih untuk keluar dari program dengan memilih opsi keluar dari pilihan menu.

Dengan program ini, pengguna dapat dengan mudah mengelola data mahasiswa seperti menambahkan, menghapus, dan mencari data berdasarkan NIM atau rentang nilai tertentu.

BAB IV

KESIMPULAN

Praktikum kali ini bertujuan untuk memperkenalkan konsep dan implementasi Hash Table dalam pemrograman. Melalui praktikum kali ini, kita dapat mempelajari konsep dasar Hash Table, fungsi hash, operasi utama seperti penambahan, penghapusan, pencarian, dan pembaruan data, serta teknik penyelesaian collision.

Dalam praktikum kali ini, telah dilakukan implementasi Hash Table menggunakan chaining dan probing (dalam contoh quadratic probing) untuk mengatasi collision. Selain itu, juga dilakukan implementasi program yang lebih kompleks untuk mengelola data mahasiswa menggunakan Hash Table. Program-program ini memungkinkan pengguna untuk melakukan berbagai operasi seperti penambahan, penghapusan, dan pencarian data mahasiswa berdasarkan NIM atau rentang nilai tertentu.

Dengan demikian, praktikum ini memberikan pemahaman yang kuat tentang konsep Hash Table dan kemampuan untuk mengimplementasikannya dalam pemrograman. Ini merupakan landasan yang penting dalam pengembangan aplikasi dan sistem yang membutuhkan manajemen data yang efisien dan cepat.

DAFTAR PUSTAKA

Vijay Krishna, Bradley Kouchi , 2023. How to Implement a Sample Hash Table in C/C++ , Diakses pada 14 Mei 2024 dari https://www.digitalocean.com/community/tutorials/hash-table-in-c-plus-plus