LAPORAN PRAKTIKUM

MODUL 5
HASH TABLE

Disusun oleh:
Bayu Kuncoro Adi
NIM: 2311102031

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. TUJUAN PRAKTIKUM

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

BAB II
DASAR TEORI

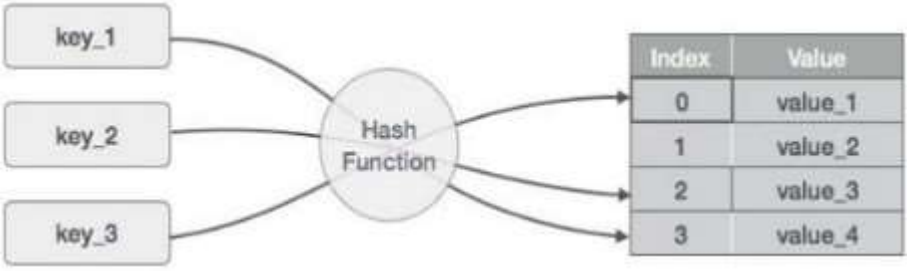
A. DASAR TEORI

1. 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 disebutchaining.



2. 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.

3. **Operasi Hash Table**

   a. **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.

   b.Deletion:

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

   c. 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.

   d.Update:

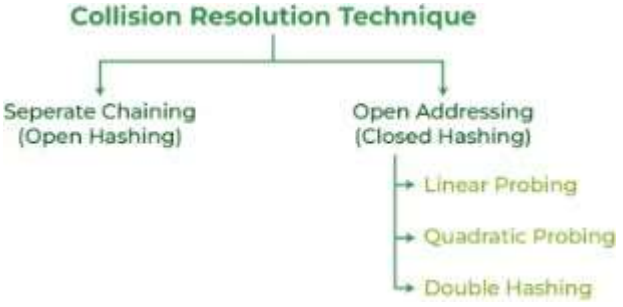
   Memperbarui data dalam hash table dengan mencari data menggunakan fungsi hash, dan kemudian memperbarui data yang ditemukan.

   e. Traversal:

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

4. 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 :



a. 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.

b. Closed Hashing

1. **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.

2. **Quadratic Probing**

   Penanganannya hampir sama dengan metode linear, hanya lompatannya tidak satu-satu, tetapi quadratic (12, 22, 32, 42, ...)

3. **Double Hashing**

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

BAB III
GUIDED

1. Guided 1

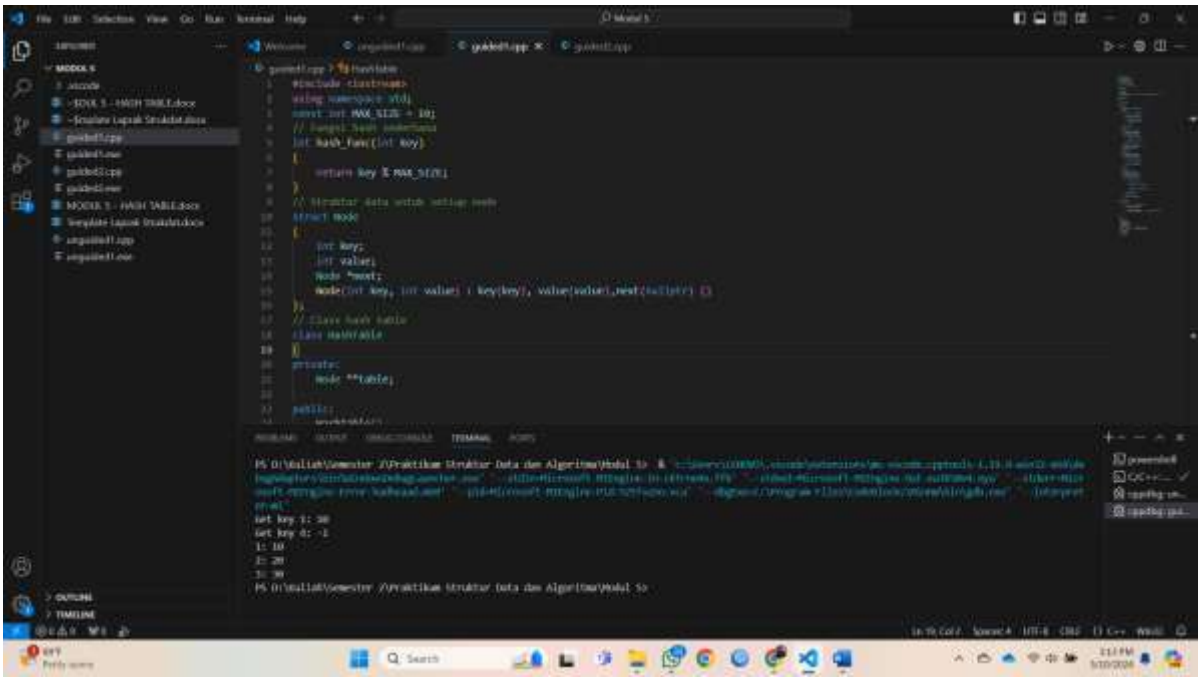
Program:

```
#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();
}
```

Screenshoot Program:



Deskripsi Program:

Program diatas merupkan contoh program C++ yang menggunakan Hash Table. Pada program tersebut terdapat array, struct, get, insert, remove dan lainnya. Program tersebut menghasilkan output yaitu Get key 1: 10, Get key 2: -1, 1: 10, 2: 20, 3: 30. Dalam program tersebut terdapat menu untuk mencari, menginput, dan juga untuk menghapus data pada program tersebut.

Program di atas adalah implementasi dari struktur data hash table menggunakan chaining untuk menangani tabrakan (collision) menggunakan fungsi hash sederhana. Program ini memiliki kelas `HashTable` yang menggunakan array `table` untuk menyimpan pointer ke linked list dari `Node` untuk setiap slot dalam tabel hash. Fungsi hash `hash_func` digunakan untuk menentukan indeks penyimpanan berdasarkan kunci (key) yang dimasukkan. Setiap `Node` dalam linked list memiliki dua nilai, yaitu `key` (kunci) dan `value` (nilai), serta pointer `next` untuk menghubungkan ke node selanjutnya dalam kasus terjadinya tabrakan. Metode `insert` digunakan untuk memasukkan pasangan kunci-nilai baru ke dalam hash table. Jika kunci sudah ada, nilai yang terkait dengan kunci tersebut diperbarui. Metode `get` digunakan untuk mencari nilai yang terkait dengan kunci tertentu. Metode `remove` digunakan untuk menghapus entri yang terkait dengan kunci tertentu dari hash table. Metode `traverse` digunakan untuk menelusuri semua entri dalam hash table dan mencetak pasangan kunci-nilai. Dalam `main`, program menunjukkan contoh penggunaan dengan menyisipkan beberapa entri, mencari nilai dengan kunci tertentu, menghapus entri yang tidak ada, dan menelusuri semua entri yang tersisa dalam hash table.

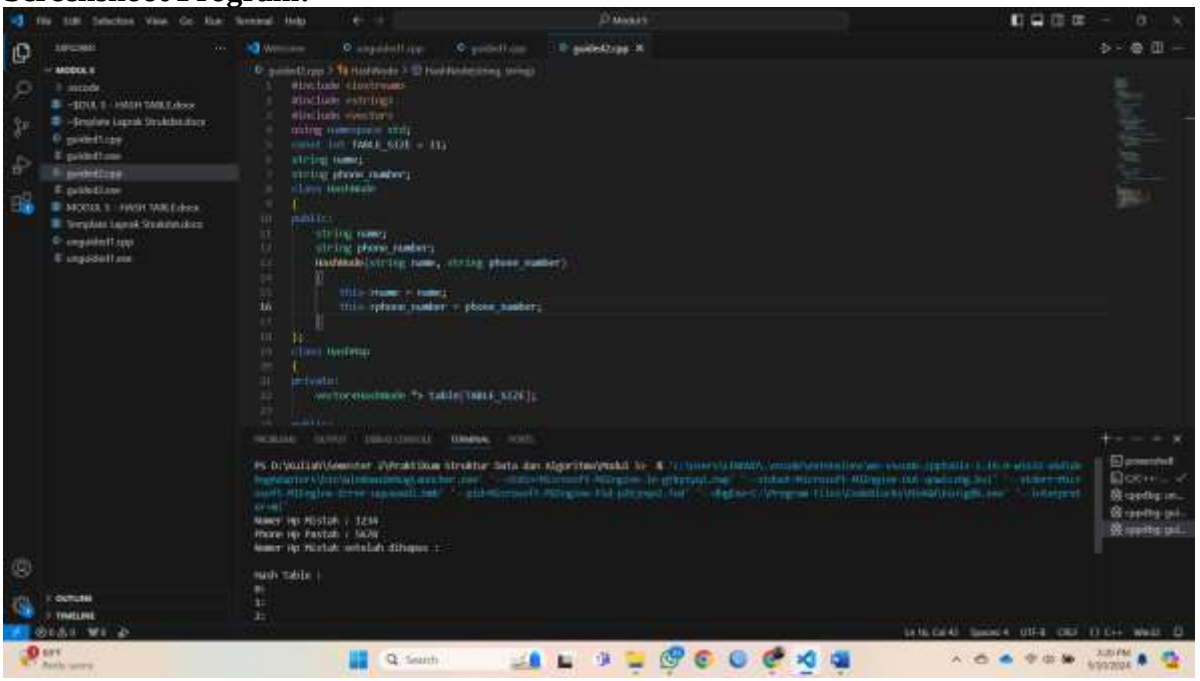
2. **Guided 2**
Program:

```
#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:

Program diatas merupakan program C++ yang menggunakan Hash Table. Didalam program tersebut terdapat string, class, cout, cin, hash table, void, array. Program diatas menjalankan bagaimana sebuah nomor hp yang bernama Mistah, yang kemudian dihapus kemudian program tersebut menampilkan hasil penempatan setelah nomor tersebut dihapus. Program di atas adalah implementasi sederhana dari hash map (tabel hash) menggunakan teknik chaining untuk menangani tabrakan. Program ini menggunakan kelas `HashMap` yang memiliki array `table` yang berisi vektor dari `HashNode *` untuk menyimpan pasangan nama dan nomor telepon. Fungsi hash `hashFunc` digunakan untuk menghitung nilai hash berdasarkan nama. Metode `insert` digunakan untuk menambahkan atau mengupdate entri dengan pasangan nama dan nomor telepon ke dalam hash map. Metode `remove` digunakan untuk menghapus entri berdasarkan nama. Metode `searchByName` digunakan untuk mencari nomor telepon berdasarkan nama. Metode `print` digunakan untuk mencetak seluruh isi hash map. Dalam `main`, program menunjukkan contoh penggunaan dengan menyisipkan beberapa entri, mencari nomor telepon berdasarkan nama, menghapus entri, dan mencetak isi hash map untuk memperlihatkan struktur data yang terbentuk setelah operasi-operasi tersebut.

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).

Program:

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

using namespace std;

const int TABLE_SIZE = 100;

struct Mahasiswa
{
    string nama;
    string nim;
    int nilai;
};
class HashNode
{
public:
    string name_147;
    string nim_147;
    int nilai_147;

    HashNode(string name, string nim, int nilai)
    {
        this->name_147 = name;
        this->nim_147 = nim;
        this->nilai_147 = nilai;
    }
};
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 nim, int nilai)
    {
        int hash_val = hashFunc(name);
        for (auto node : table[hash_val])
        {
            if (node->name_147 == name)
            {
                node->nim_147 = nim;
                node->nilai_147 = nilai;
                cout << "Data Mahasiswa dengan Nama " << name << "
berhasil ditambahkan." << endl;
                return;
            }
        }
        table[hash_val].push_back(new HashNode(name, nim, nilai));
    }
    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_147 == name)
            {
                delete *it;
                table[hash_val].erase(it);
                cout << "Data Mahasiswa dengan nama " << name << "
berhasil dihapus." << endl;
                return;
            }
        }
    }
    void searchByNIM(string nim)
    {
        bool found = false;
```

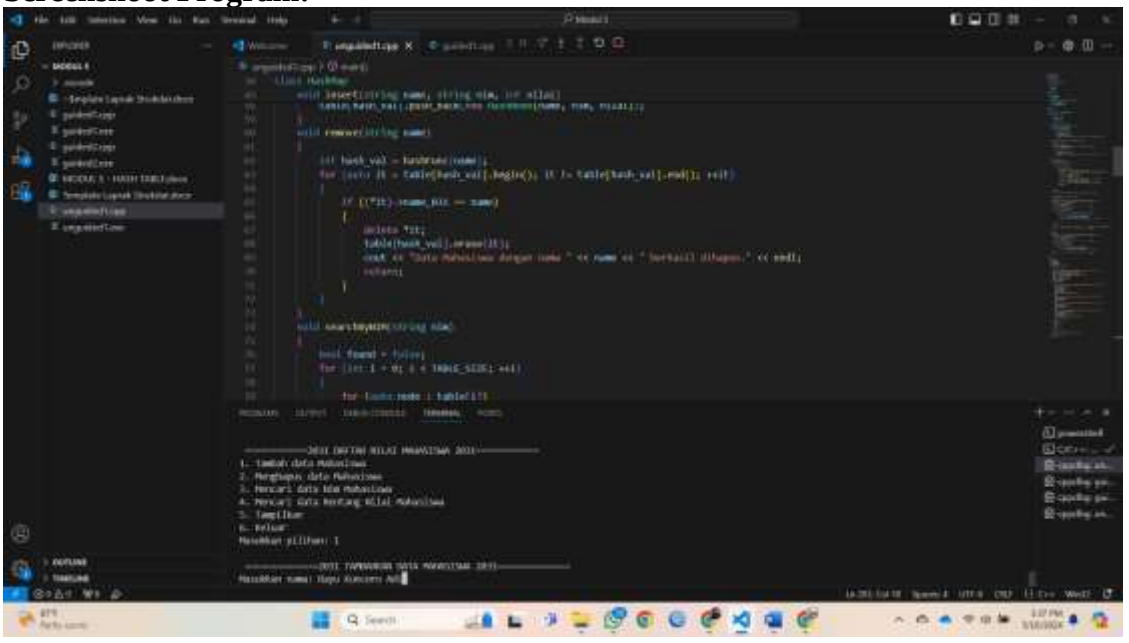
```
        for (int i = 0; i < TABLE_SIZE; ++i)
        {
            for (auto node : table[i])
            {
                if (node->nim_147 == nim)
                {
                    // Menampilkan data mahasiswa pakai setw agar rapi
                    cout << "\n ============2147 MENCARI DATA NIM
MAHASISWA 2147============" << endl;
                    cout << " ------------------------------------------
--------" << endl;
                    cout << left << "| " << setw(15) << "Nama"
                         << "| " << setw(20) << "NIM"
                         << "| " << setw(10) << "Nilai"
                         << " |" << endl;
                    cout << " ------------------------------------------
--------" << endl;
                    cout << "| " << left << setw(15) << node->name_147
                         << "| " << left << setw(20) << node->nim_147
                         << "| " << left << setw(10) << node->nilai_147
<< " |" << endl;
                    cout << " ------------------------------------------
--------" << endl;
                    found = true;
                    return;
                }
            }
        }
        if (!found)
        {
            cout << "Mahasiswa dengan NIM " << nim << " tidak ditemukan."
<< endl;
        }
    }

    void CariRentangNilai(int NilaiMin, int NilaiMax)
    {
        bool found = false;
        cout << "\n =======2147 MENCARI DATA RENTANG NILAI MAHASISWA
2147=======" << endl;
        cout << " ------------------------------------------------" <<
endl;
        cout << left << "| " << setw(15) << "Nama"
             << "| " << setw(20) << "NIM"
             << "| " << setw(10) << "Nilai"
             << " |" << endl;
        cout << " ------------------------------------------------" <<
endl;
```

```
        for (int i = 0; i < TABLE_SIZE; ++i)
        {
            for (auto node : table[i])
            {
                if (node->nilai_147 >= NilaiMin && node->nilai_147 <=
NilaiMax)
                {
                    cout << "| " << left << setw(15) << node->name_147
                         << "| " << left << setw(20) << node->nim_147
                         << "| " << left << setw(10) << node->nilai_147
<< " |" << endl;
                    found = true;
                }
            }
        }
        if (!found)
        {
            cout << "Tidak ada data Mahasiswa dengan nilai antara " <<
NilaiMin << " dan " << NilaiMax << "." << endl;
        }
        cout << " ----------------------------------------------------" <<
endl;
    }

    void print()
    {
        cout << "\n ==================2147 DATA MAHASISWA
2147=================" << endl;
        cout << " ----------------------------------------------------" <<
endl;
        cout << left << "| " << setw(15) << "Nama"
             << "| " << setw(20) << "NIM"
             << "| " << setw(10) << "Nilai"
             << " |" << endl;
        cout << " ----------------------------------------------------" <<
endl;
        for (int i = 0; i < TABLE_SIZE; ++i)
        {
            for (auto pair : table[i])
            {
                cout << "| " << left << setw(15) << pair->name_147
                     << "| " << left << setw(20) << pair->nim_147
                     << "| " << left << setw(10) << pair->nilai_147 << "
|" << endl;
            }
        }
        cout << " ----------------------------------------------------" <<
endl;
```

```
    }
};

int main()
{
    HashMap map;
    int choice_147;
    string name_147;
    string nim_147;
    int nilai_147;
    int search1_147, search2_147;
    do
    {
        // Menampilkan menu
        cout << "\n ============2147 DAFTAR NILAI MAHASISWA
2147============" << endl;
        cout << "1. Tambah data Mahasiswa" << endl;
        cout << "2. Menghapus data Mahasiswa" << endl;
        cout << "3. Mencari data Nim Mahasiswa" << endl;
        cout << "4. Mencari data Rentang Nilai Mahasiswa" << endl;
        cout << "5. Tampilkan" << endl;
        cout << "6. Keluar" << endl;
        cout << "Masukkan pilihan: ";
        cin >> choice_147;
        switch (choice_147)
        {
        case 1:
            cout << "\n ==============2147 TAMBAHKAN DATA MAHASISWA
2147==============" << endl;
            cout << "Masukkan nama: ";
            cin >> name_147;
            cout << "Masukkan NIM: ";
            cin >> nim_147;
            cout << "Masukkan nilai: ";
            cin >> nilai_147;
            map.insert(name_147, nim_147, nilai_147);
            break;
        case 2:
            cout << "\n ===========2147 MENGHAPUS DATA MAHASISWA
2147===========" << endl;
            cout << "Masukkan nama: ";
            cin >> name_147;
            map.remove(name_147);
            break;
        case 3:
            cout << "\n ============2147 MENCARI DATA NIM MAHASISWA
2147============" << endl;
            cout << "Masukkan NIM: ";
```

```
            cin >> nim_147;
            map.searchByNIM(nim_147);
            break;
        case 4:
            cout << "\n =======2147 MENCARI DATA RENTANG NILAI MAHASISWA
2147=======" << endl;
            cout << "Masukkan nilai minimum: ";
            cin >> search1_147;
            cout << "Masukkan nilai maksimum: ";
            cin >> search2_147;
            map.CariRentangNilai(search1_147, search2_147);
            break;

        // Pilihan 5: Tampilkan data
        case 5:
            map.print();
            break;

        // Pilihan 6: Keluar
        case 6:
            break;

        // Pilihan tidak tersedia
        default:
            cout << "Pilihan tidak tersedia!" << endl;
        }
    } while (choice_147 != 6);

    return 0;
}
```

Screenshoot Program:



Deskripsi Program:
Program di atas merupakan implementasi dari sebuah hash map (tabel hash) menggunakan teknik chaining dengan kelas `HashMap` yang menggunakan array `table` yang berisi vektor dari pointer `HashNode *`. Setiap `HashNode` berisi informasi tentang seorang mahasiswa, termasuk nama, NIM, dan nilai. Program ini menyediakan fungsi `insert` untuk menambahkan data mahasiswa ke hash map, dengan mengecek apakah data mahasiswa sudah ada sebelumnya; jika sudah, data akan diupdate. Fungsi `remove` digunakan untuk menghapus data mahasiswa berdasarkan nama. Fungsi `searchByNIM` digunakan untuk mencari dan menampilkan data mahasiswa berdasarkan NIM. Fungsi `CariRentangNilai` digunakan untuk mencari dan menampilkan data mahasiswa berdasarkan rentang nilai. Terdapat juga fungsi `print` yang digunakan untuk mencetak seluruh data mahasiswa dalam hash map. Dalam `main`, program menampilkan menu interaktif untuk pengguna yang memungkinkan penambahan data mahasiswa, penghapusan data berdasarkan nama, pencarian data berdasarkan NIM atau rentang nilai, serta menampilkan seluruh data mahasiswa yang tersimpan dalam hash map. Pengguna dapat keluar dari program dengan memilih opsi keluar (pilihan 6) dari menu. Program ini memberikan fungsionalitas dasar untuk mengelola dan mencari data mahasiswa menggunakan struktur data hash map.

BAB IV
KESIMPULAN

Implementasi dari hash table untuk menyimpan data mahasiswa adalah suatu pendekatan yang efektif dalam manajemen data. Dengan menggunakan struktur data hash map dan teknik chaining, program dapat menyimpan data mahasiswa berdasarkan kunci unik (dalam hal ini, nama mahasiswa) dan melakukan operasi seperti penambahan, penghapusan, dan pencarian data dengan efisien.

Pada program yang diberikan, setiap entri dalam hash table direpresentasikan oleh objek `HashNode` yang berisi informasi nama, NIM, dan nilai mahasiswa. Fungsi hash digunakan untuk menghitung indeks penyimpanan berdasarkan kunci (nama mahasiswa), dan teknik chaining digunakan untuk menangani tabrakan (collision) dengan menyimpan entri dengan kunci yang sama dalam linked list.

Menu interaktif pada program memungkinkan pengguna untuk melakukan berbagai operasi, termasuk penambahan data mahasiswa baru, penghapusan data berdasarkan nama, pencarian data berdasarkan NIM, dan pencarian data berdasarkan rentang nilai. Fungsi pencarian berdasarkan rentang nilai memberikan kemampuan untuk mencari data mahasiswa dengan nilai tertentu, sehingga memberikan fleksibilitas yang berguna dalam analisis data.

Dengan demikian, implementasi hash table untuk data mahasiswa merupakan contoh yang baik dari penerapan struktur data yang efisien dalam manajemen data, terutama ketika ada kebutuhan untuk mengelola dan mengakses data dengan kecepatan operasi konstan (O(1)) dalam kasus pencarian terbaik.

DAFTAR PUSTAKA

Hash Table UI diakses pada tanggal 10 Mei 2024 dari http://aren.cs.ui.ac.id/sda/resources/sda2010/15_hashtable.pdf

Modul 3 Single and Double Linked List Praktikum Struktur Data dan Algoritma

Modul 5 Hash Table Praktikum Struktur Data dan Algoritma

Modul 6 Single dan Double Linked List diakses dari https://elektro.um.ac.id/wp-content/uploads/2016/04/ASD-Modul-6-Linked-List.pdf

Single dan Double Linked List diakses dari https://www.academia.edu/82277141/SINGLE_LINKED_LIST_DOUBLE_ENDED_LIST_DOUBLY_LINKED_LIST_CIRCULAR_LINKED_LIST_DAN_ITERATOR?f_ri=1292408

Struktur Data diakses dari https://github.com/yunusfebriansyah/struktur-data/tree/main/Circular%20Single%20Linked%20List