LAPORAN PRAKTIKUM STRUKTUR DATA DAN ALGORITMA

MODUL 5

"HASH TABLE"

DISUSUN OLEH :

M.Fathiakmal.L.A

18102096

DOSEN

WAHYU ANDI SAPUTRA, S.PD., M.PD.

PROGRAM STUDI S1 TEKNIK INFORMATIKA

FAKULTAS INFORMATIKA

INSTITUT TEKNOLOGI TELKOM PURWOKERTO

2024

A. Dasar Teori

Hash Table (Tabel Hash) adalah struktur data yang digunakan untuk menyimpan dan mengelola kumpulan data, dimana setiap elemen dalam kumpulan data memiliki kunci (key) yang unik dan nilainya (value) terkait. Konsep utama di balik tabel hash adalah penggunaan fungsi hash untuk mengonversi kunci menjadi alamat atau indeks di dalam tabel, sehingga memungkinkan pencarian dan pengambilan data dengan efisien.

Pada dasarnya, tabel hash berfungsi sebagai penyimpanan asosiatif, di mana data disimpan dalam bentuk pasangan kunci-nilai. Proses ini melibatkan penggunaan fungsi hash untuk menghasilkan indeks unik dari kunci, dan nilai yang sesuai akan disimpan pada indeks tersebut..

Salah satu keuntungan utama dari tabel hash adalah kecepatan akses dan pencarian data yang sangat efisien. Ketika kita ingin mencari nilai berdasarkan kunci, kita hanya perlu melakukan proses hashing pada kunci untuk mengetahui lokasi penyimpanan datanya, sehingga tidak perlu mencari secara berurutan seperti pada struktur data lainnya seperti array atau list.

Namun, perlu diingat bahwa pada situasi tertentu, bisa saja terjadi tabrakan hash (hash collision), yaitu ketika dua kunci berbeda menghasilkan indeks yang sama. Untuk menangani tabrakan ini, metode penyelesaian tabrakan (collision resolution) digunakan, seperti linear probing, chaining, atau double hashing. Hash table digunakan dalam berbagai aplikasi, termasuk dalam basis data, kamus (dictionary), dan kebanyakan algoritma yang membutuhkan pencarian atau pengelompokan data dengan cepat dan efisien.

FUNGSI HASH TABLE

Fungsi utama dari Hash Table (Tabel Hash) adalah untuk menyediakan struktur data yang memungkinkan pencarian, penyisipan, dan penghapusan data dengan efisien. Berikut adalah beberapa fungsi utama dari Hash Table :

1. Pencarian (Search)

Hash Table memungkinkan pencarian data berdasarkan kunci (key) dengan cepat. Saat kita ingin mencari nilai berdasarkan kunci, fungsi hash akan mengonversi kunci menjadi

indeks, dan nilai yang sesuai dapat diakses langsung dari indeks tersebut, menghasilkan waktu akses yang konstan (O(1)).

2. Penyisipan (Insertion)

Hash Table memungkinkan penambahan data baru dengan kunci dan nilainya. Saat kita ingin menyisipkan data baru, fungsi hash akan mengonversi kunci menjadi indeks, dan nilai tersebut akan disimpan pada indeks yang sesuai. Proses ini dapat dilakukan dengan waktu yang konstan (O(1)) pada kebanyakan kasus, membuatnya sangat efisien.

3. Penghapusan (Deletion)

Hash Table memungkinkan penghapusan data berdasarkan kunci. Saat kita ingin menghapus data, hash table akan menggunakan fungsi hash untuk menemukan indeks data yang sesuai dengan kunci dan menghapusnya dari tabel. Seperti pencarian dan penyisipan, operasi penghapusan juga berjalan dengan waktu yang konstan (O(1)) dalam kebanyakan kasus.

4. Asosiasi Kunci-Nilai (Key-Value Association)

Hash Table menyimpan data dalam bentuk pasangan kunci-nilai (key-value). Ini memungkinkan kita untuk mengaitkan kunci dengan nilai tertentu sehingga kita dapat dengan mudah mengakses nilai tersebut ketika diberikan kunci.

5. Kecepatan Akses

Salah satu keunggulan utama dari Hash Table adalah kecepatan aksesnya. Dengan menggunakan fungsi hash, proses mencari dan mengakses data menjadi sangat cepat karena kita dapat langsung menuju lokasi data tanpa perlu mencari secara berurutan.

B. Guided

Guided 1

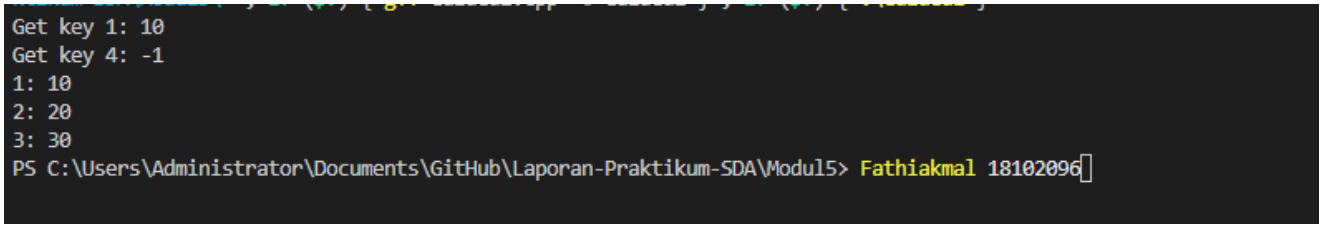
Source Code :

```
#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;
}
```

Screenshots Output:



Deskripsi Program :

Tabel hash dalam kode C++ tersebut adalah struktur data yang memungkinkan penyimpanan pasangan kunci-nilai dengan efisiensi tinggi. Fungsi hash sederhana digunakan untuk menentukan indeks dalam tabel, dan setiap node dalam tabel memiliki kunci, nilai, dan pointer ke node berikutnya. Kelas HashTable menyediakan operasi dasar seperti penyisipan, pencarian, penghapusan, dan penelusuran. Dalam fungsi main, objek HashTable dibuat dan operasi-operasi tersebut diterapkan untuk menunjukkan cara menggunakan tabel hash secara praktis. Dengan menggunakan tabel hash, program dapat mengelola data dengan cepat dan efisien, terutama saat memiliki jumlah data yang besar.

Guided 2

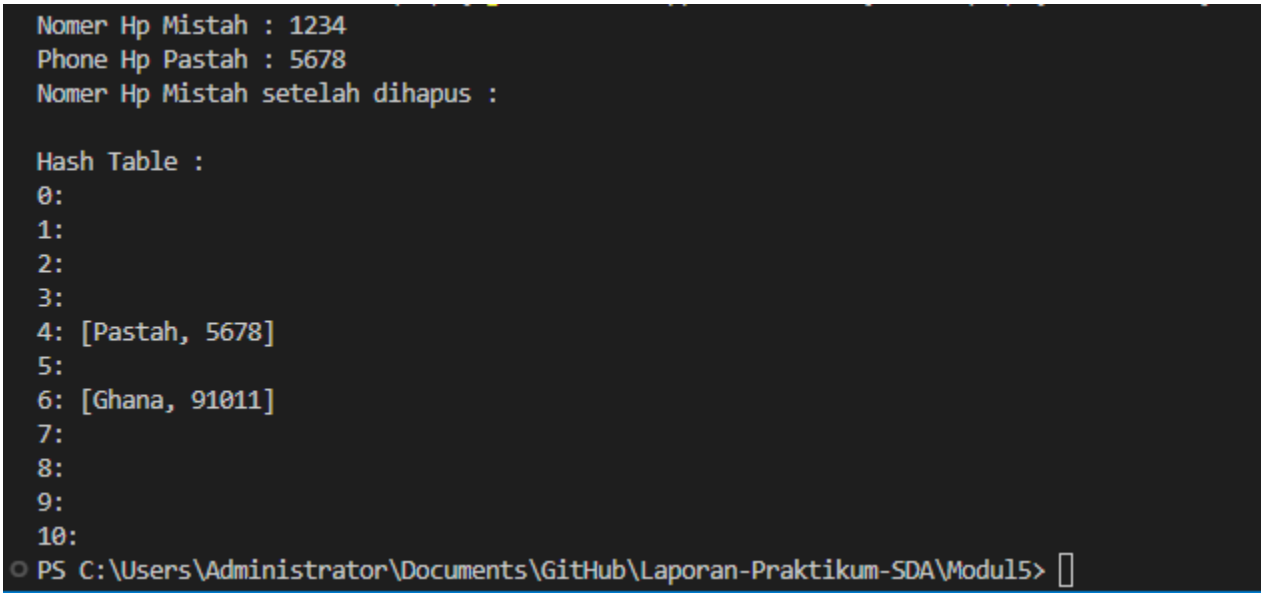
Source Code :

```
#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:



Deskripsi Program :

Kode dimulai dengan beberapa inklusi, definisi konstanta, dan deklarasi variabel global. Kemudian, ada definisi dari dua kelas: HashNode, yang merepresentasikan node dalam tabel hash, dan HashMap, yang mewakili struktur data HashMap itu sendiri. Kelas HashMap memiliki metode untuk melakukan operasi dasar pada HashMap, seperti insert (untuk menambahkan entri), remove (untuk menghapus entri), dan searchByName (untuk mencari nomor telepon berdasarkan nama). Selain itu, ada juga metode print untuk mencetak isi seluruh tabel hash. Di dalam fungsi main, objek employee_map dari kelas HashMap dibuat, dan beberapa entri ditambahkan ke dalamnya dengan menggunakan metode insert. Selanjutnya, nomor telepon dari beberapa nama dicari dan dicetak menggunakan metode searchByName. Salah satu entri dihapus menggunakan metode remove, dan tabel hash dicetak menggunakan metode print.

C. Unguided

Source Code:

```
#include <iostream>
#include <string>
#include <vector>
using namespace std;
const int TABLE_SIZE = 11;
string nim;
int nilai;
class HashNode
{
public:
    string nim;
    int nilai;
    HashNode(string nim, int nilai)
    {
        this->nim = nim;
        this->nilai = 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 nim, int nilai)
    {
        int hash_val = hashFunc(nim);
        for (auto node : table[hash_val])
        {
            if (node->nim == nim)
            {
                node->nilai = nilai;
                return;
            }
        }
```

```
        table[hash_val].push_back(new HashNode(nim,
                                               nilai));
    }
    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;
            }
            else
            {
                cout << "Tidak Ditemukan" << endl;
            }
        }
    }

    int searchByNim(string nim)
    {
        int hash_val = hashFunc(nim);
        for (auto node : table[hash_val])
        {
            if (node->nim == nim)
            {
                // return node->nilai;
                cout << "\nMahasiswa dengan NIM " << node->nim << " memiliki nilai "
<< node->nilai << endl;
            }
        }
        return 0;
    }
    int searchByNilai(int minNilai, int maxNilai)
    {
        for (const auto &bucket : table)
        {
            for (auto node : bucket)
            {
                if (node->nilai >= minNilai && node->nilai <= maxNilai)
                {
                    cout << "[ NIM : " << node->nim << ", NILAI : " << node->nilai <<
" ]" << endl;
                }
            }
        }
        return 0;
```

```
    }

    void print()
    {
        for (int i = 0; i < TABLE_SIZE; i++)
        {
            cout << i << ": ";
            for (auto pair : table[i])
            {
                if (pair != nullptr)
                {
                    cout << "[ NIM : " << pair->nim << ", NILAI : " << pair->nilai <<
" ]";
                }
            }
            cout << endl;
        }
    }
};
int main()
{
    HashMap mahasiswa_map;
    bool menu = true;
    int choice;
    do
    {

        cout << " Data Mahasiswa  " << endl;
        mahasiswa_map.print();
        cout << endl;
        cout << "1. Tambah Data" << endl;
        cout << "2. Hapus Data" << endl;
        cout << "3. Cari berdasarkan NIM" << endl;
        cout << "4. Cari berdasarkan Nilai" << endl;
        cout << "0. Keluar" << endl;
        cout << "Pilihan Anda: ";
        cin >> choice;
        switch (choice)
        {
        case 1:
            cout << "Masukkan NIM: ";
            cin >> nim;
            cout << "Masukkan Nilai: ";
            cin >> nilai;

            mahasiswa_map.insert(nim, nilai);
            break;
        case 2:
            cout << "Masukkan NIM: ";
```

```
            cin >> nim;
            mahasiswa_map.remove(nim);
            break;
        case 3:
            cout << "Masukkan NIM: ";
            cin >> nim;
            mahasiswa_map.searchByNim(nim);

            break;
        case 4:
            int maxNilai, minNilai;
            cout << "Masukkan Nilai Tertinggi: ";
            cin >> maxNilai;
            cout << "Masukkan Nilai Terendah: ";
            cin >> minNilai;
            mahasiswa_map.searchByNilai(minNilai, maxNilai);
            break;
        case 0:
            menu = false;
            break;

        default:
            break;
        }

    } while (menu == true);
    return 0;
}
```

Screenshot output

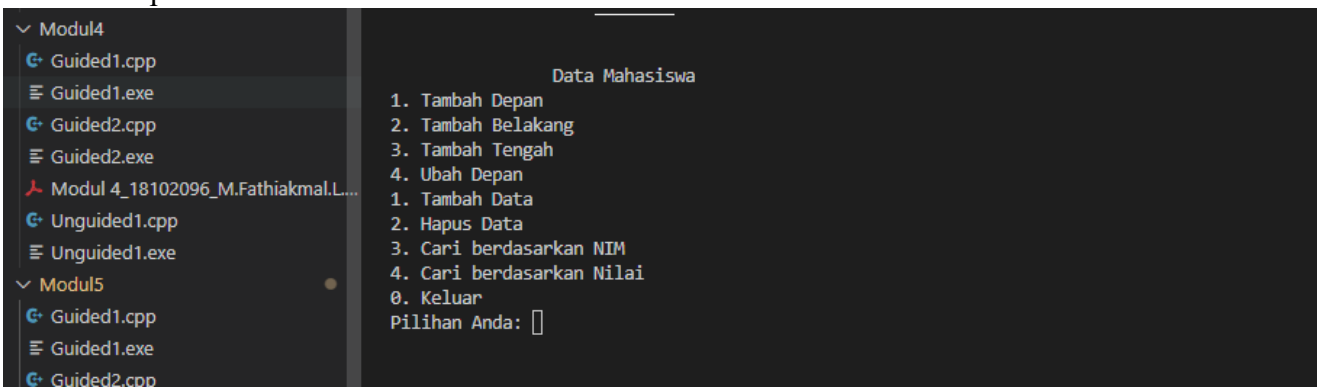
1. Implementasikan hash table untuk menyimpan data mahasiswa. Setiap mahasiswa memiliki NIM dan nilai. Implementasikan fungsi untuk menambahkan data baru, menghapus data, mencari data berdasarkan NIM, dan mencari data berdasarkan nilai. Dengan ketentuan :

a. Setiap Mahasiswa Memiliki NIM dan Nilai

b. Program memiliki tampilan pilihan menu berisi poin C.

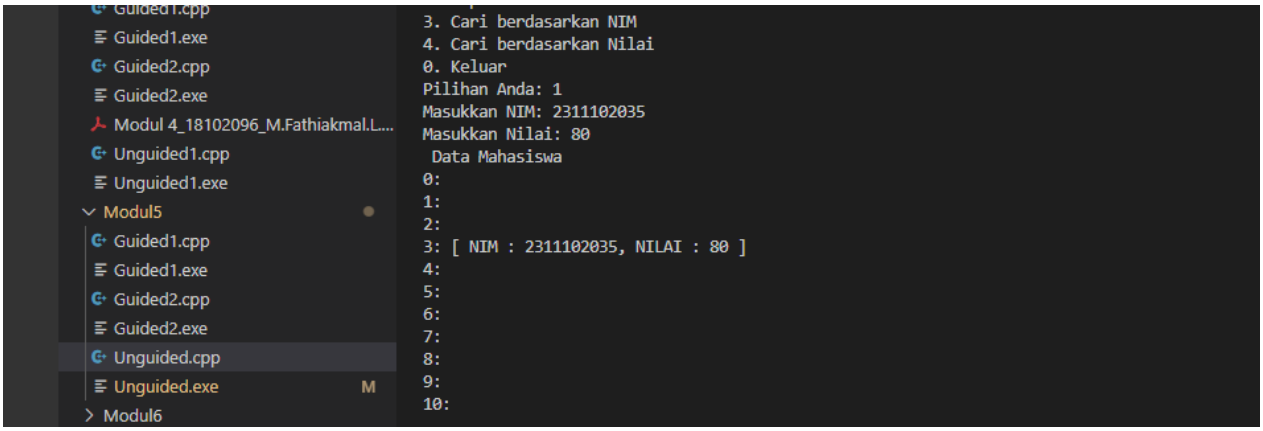
c. Implementasikan fungsi untuk menambahkan data baru, menghapus data, mencari data berdasarkan NIM, dan mencari data berdasarkan rentang nilai (80 – 90)

- Tampilan Menu

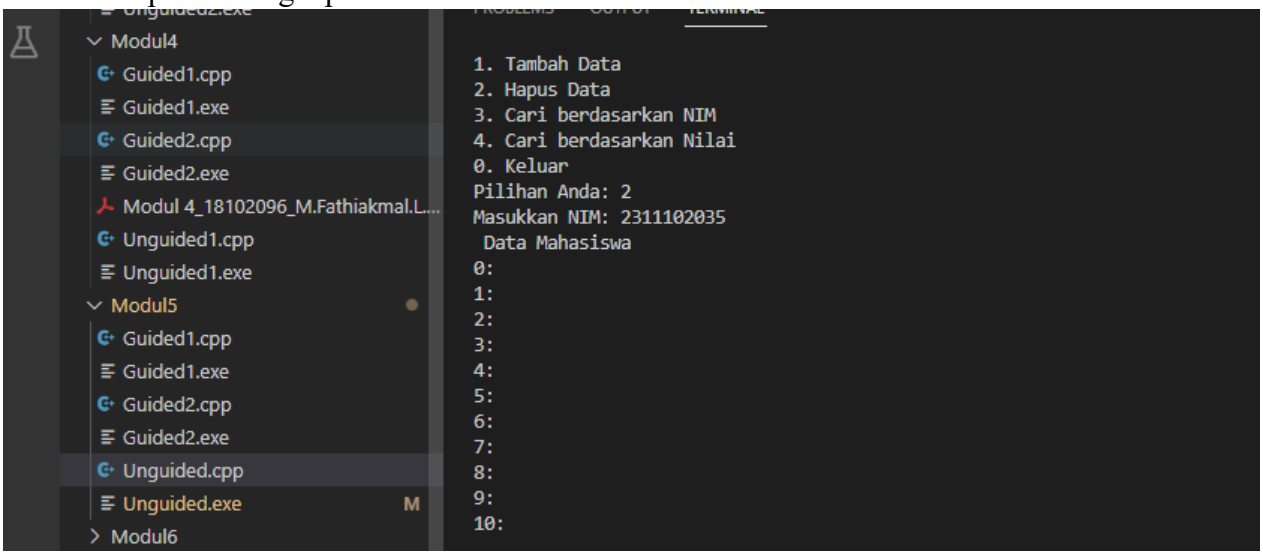


-

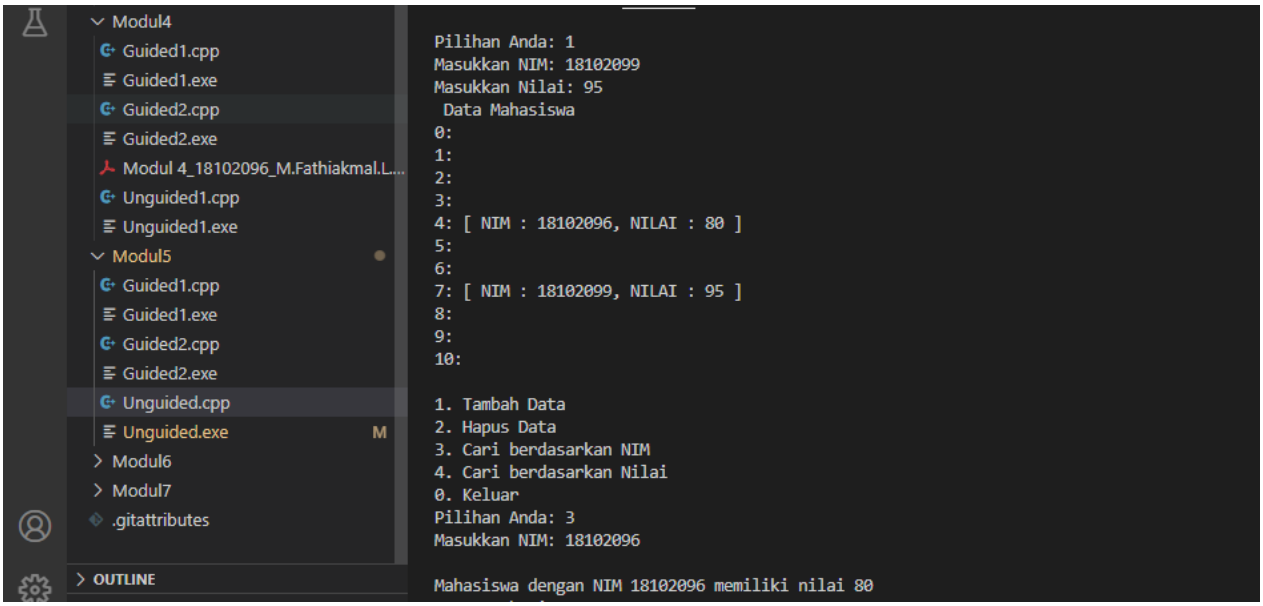
Tampilan Setelah ditambahkan Data baru



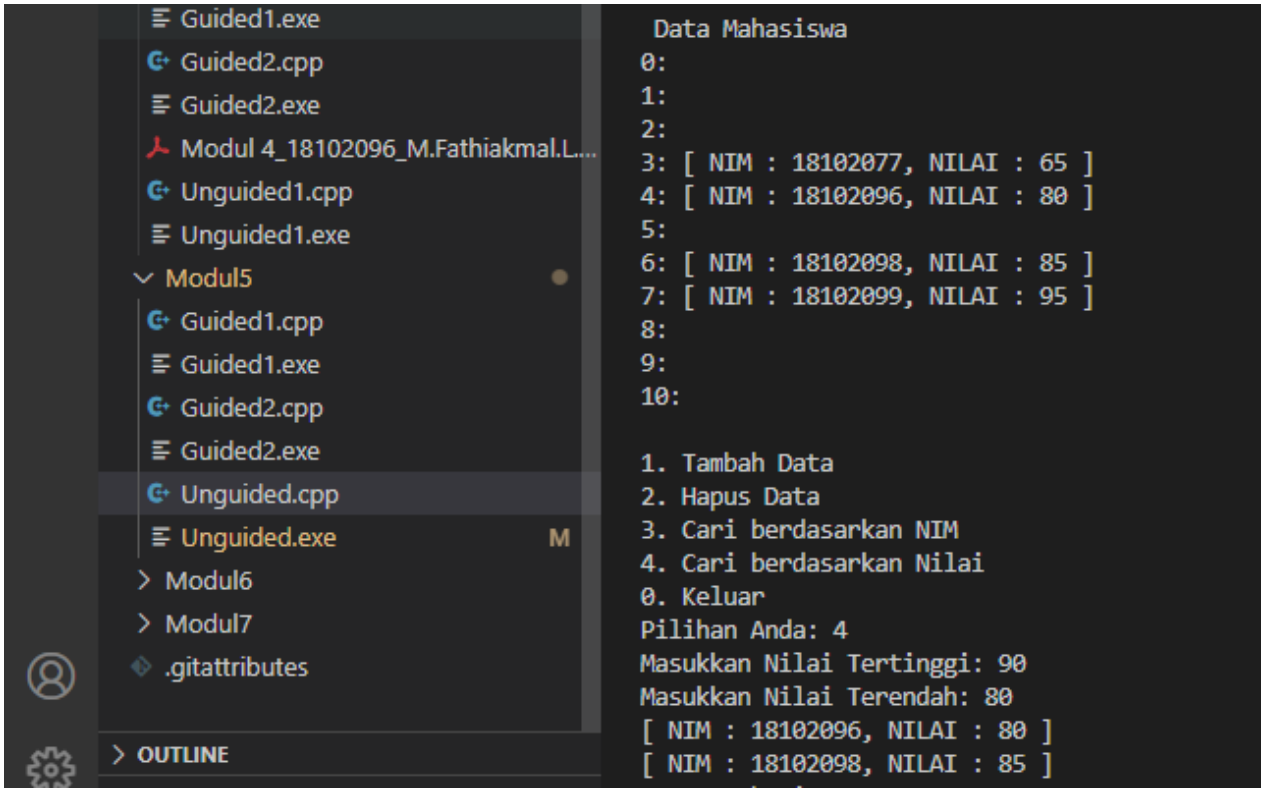
Tampilan Menghapus Data



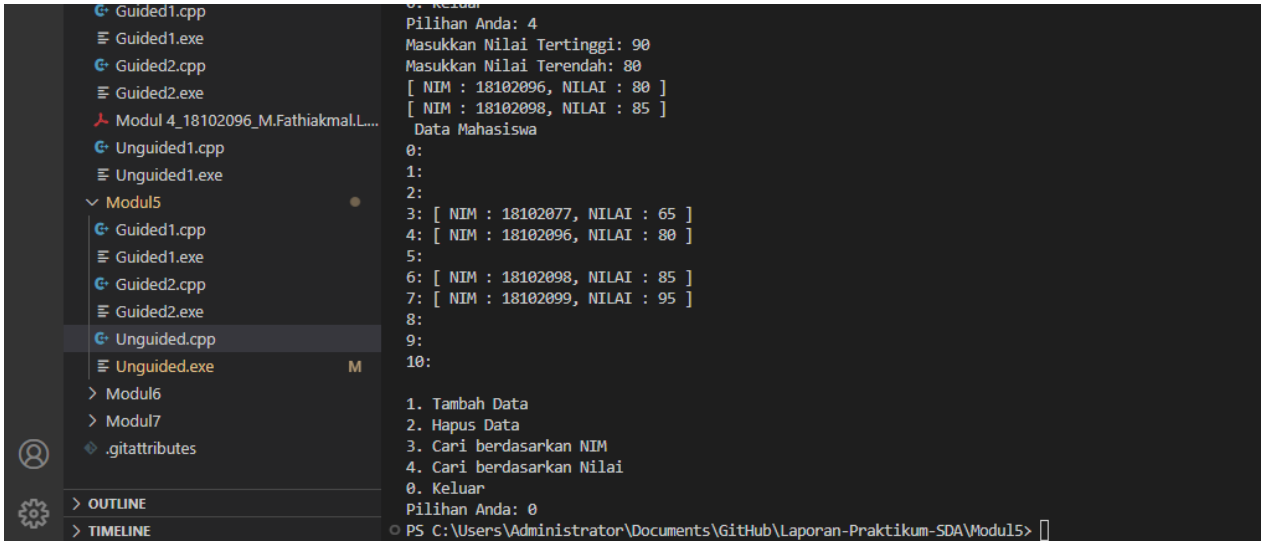
- Tampilan Mencari Data berdasarkan NIM



Tampilan Mencari Data berdasarkan rentang Nilai (80-90)



-
- Tampilan Keluar



Deskripsi Program :

- Library yang di-include:

  iostream: Untuk operasi input-output. unordered_map: Untuk menggunakan

  struktur data hash map yang tidak terurut. vector: Untuk menggunakan struktur

  data vector.

  using namespace std;: Menggunakan namespace std secara keseluruhan.

- Struktur Data:

  struct Mahasiswa: Mendefinisikan struktur data Mahasiswa yang memiliki dua atribut, yaitu NIM (Nomor Induk Mahasiswa) dan nilai.

- Kelas HashTable:

  -unordered_map<string, Mahasiswa> tabel;: Mendefinisikan sebuah hash map dengan kunci berupa string (NIM) dan nilai berupa Mahasiswa.

  -void tambahData(Mahasiswa mahasiswa): Menambahkan data mahasiswa ke dalam hash table.

  -void hapusData(string NIM): Menghapus data mahasiswa dari hash table berdasarkan NIM.

  -Mahasiswa cariDataBerdasarkanNIM(string NIM): Mencari data mahasiswa berdasarkan NIM.

  -vector<Mahasiswa> cariDataBerdasarkanRentangNilai(int nilaiMin, int nilaiMax): Mencari data mahasiswa berdasarkan rentang nilai.

- Fungsi main():

  Membuat objek HashTable.

  Menampilkan menu dan menerima input pilihan dari pengguna.

  Melakukan aksi sesuai dengan pilihan pengguna (menambah data baru, menghapus data, mencari data berdasarkan NIM, mencari data berdasarkan rentang nilai, atau keluar dari program).

  Program akan terus berjalan sampai pengguna memilih untuk keluar (pilihan 5).

D. Kesimpulan

Hash table adalah sebuah struktur data yang sangat efisien digunakan untuk menyimpan data dalam bentuk pasangan kunci-nilai. Keunggulan utama dari hash table adalah kemampuannya untuk menyediakan operasi penyimpanan, pencarian, dan penghapusan data dengan kompleksitas waktu rata-rata yang konstan, yaitu O(1) dalam kasus terbaik, asalkan fungsi hash yang digunakan seimbang dan efisien. Salah satu kelebihan utama dari hash table adalah efisiensi dalam melakukan pencarian data. Dengan fungsi hash yang baik, pencarian data dapat dilakukan dalam waktu konstan, tanpa memperhatikan jumlah data yang tersimpan. Namun, pengelolaan kolisi merupakan salah satu tantangan utama dalam penggunaan hash table. Penanganan kolisi dapat dilakukan melalui berbagai metode seperti chaining dan open addressing. Selain itu, kualitas dari fungsi hash sangat mempengaruhi kinerja dari hash table tersebut. Dengan menggunakan hash table secara bijak, kita dapat mengelola dan memanipulasi data dengan efisien dalam berbagai aplikasi, mulai dari basis data hingga algoritma pencarian.

E. Referensi

[1] Asisten Pratikum “Modul 5 Hash Table”, Learning Management System, 2024.

[2] Educative. Hash Table in C++. 3 Maret 2024.

Diakses Pada 10 Mei 2024, dari

https://www.educative.io/answers/singly-linked-list-in-cpp

[3] DosenIT (2019, juli). “Hash Table”.

Diakses pada 10 Mei 2024, dari

Hash Table: Pengertian, Fungsi dan Cara Membuat - DosenIT.com