LAPORAN PRAKTIKUM
STRUKTUR DATA DAN ALGORITMA
MODUL 5
“HASH TABLE”

DISUSUN OLEH:
TSAQIF KANZ AHMAD
2311102075
IF-11-B

DOSEN:

WAHYU ANDI SAPUTRA S.Pd., M.Eng.

PROGRAM STUDI S1 TEKNIK INFORMATIKA
FAKULTAS INFORMATIKA
INSTITUT TEKNOLOGI TELKOM PURWOKERTO
PURWOKERTO

2024

TUJUAN PRAKTIKUM

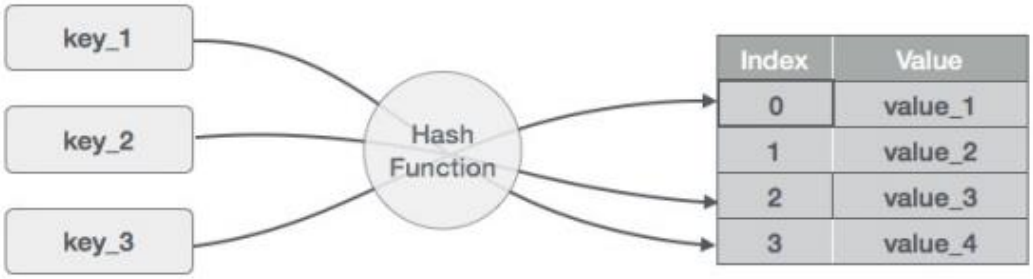
1. Mampu menjelaskan definisi dan konsep dari Hash Code.
2. Mampu menerapkan Hash Code kedalam pemrograman.

DASAR TEORI

1). PENGERTIAN HASH TABLE

Hash table merupakan struktur data yang mengatur data menjadi pasangan nilai kunci. 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 yang disebut bucket. Setiap bucket dapat menampung satu atau beberapa item data.

Fungsi hash digunakan untuk menghasilkan nilai unik untuk setiap item data, yang digunakan sebagai indeks array. Dengan cara ini, hash table memungkinkan pengambilan data dalam waktu yang konstan (O(1)) dalam skenario kasus terbaik. Sistem hash table bekerja dengan mengambil kunci input dan memetakannya ke nilai indeks array menggunakan fungsi hash. Kemudian data disimpan pada posisi indeks array yang dihasilkan oleh fungsi hash. Ketika data perlu dicari, kunci input digunakan sebagai parameter untuk fungsi hash, dan posisi indeks array yang dihasilkan digunakan untuk menemukan data. Dalam kasus tabrakan hash, ketika dua atau lebih data memiliki nilai hash yang sama, tabel hash menyimpan data ini dalam slot yang sama menggunakan teknik yang disebut chaining.



2). FUNGSI HASH TABLE

Fungsi hash menciptakan pemetaan antara kunci dan nilai, yang dicapai melalui penggunaan rumus matematika yang dikenal sebagai fungsi hash. Output dari fungsi hash disebut sebagai nilai hash. Nilai hash merupakan representasi dari string karakter asli, namun biasanya berukuran lebih kecil dari aslinya. Kunci dari fungsi hash harus didistribusikan secara merata ke seluruh array melalui fungsi hash yang layak untuk mengurangi tabrakan dan memastikan kecepatan pencarian yang cepat.

- Asumsi bilangan bulat semesta: Kunci diasumsikan bilangan bulat dalam rentang tertentu sesuai dengan asumsi bilangan bulat semesta. Hal ini memungkinkan penggunaan operasi hashing dasar seperti hashing pembagian atau perkalian.
- Hashing berdasarkan pembagian: Teknik hashing langsung ini menggunakan sisa nilai kunci setelah membaginya dengan ukuran array sebagai indeks. Jika ukuran array adalah bilangan prima dan jarak kuncinya sama, kinerjanya akan baik.
- Hashing dengan perkalian: Operasi hashing langsung ini mengalikan kunci dengan konstanta antara 0 dan 1 sebelum mengambil bagian pecahan dari hasilnya. Setelah itu, indeks ditentukan dengan mengalikan komponen pecahan dengan ukuran array. Selain itu, berfungsi secara efektif bila tombol tersebar secara merata.

Memilih fungsi hash yang layak didasarkan pada properti kunci dan fungsionalitas tabel hash yang diinginkan. Menggunakan fungsi yang mendistribusikan tombol secara merata dan mengurangi benturan sangatlah penting.

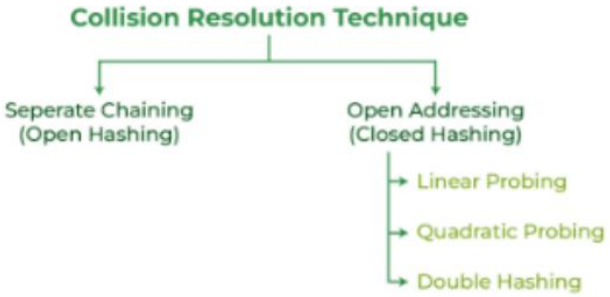
3). OPERASI HASH TABLE

Hash table menyediakan operasi dasar untuk mengelola data yang disimpannya. Berikut adalah beberapa operasi hash table :

1. Insertion (Penyisipan) : Memasukkan data baru ke dalam hash table melibatkan pemanggilan fungsi hash untuk menentukan posisi bucket yang tepat, kemudian menambahkan data ke bucket tersebut.
2. Deletion (Penghapusan) : Menghapus data dari hash table dengan mencari data menggunakan fungsi hash, dan kemudian menghapusnya dari bucket yang sesuai.
3. Searching (Pencarian) : 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 (Pembaruan) : Memperbarui data dalam hash table dengan mencari data menggunakan fungsi hash, dan kemudian memperbarui data yang ditemukan.
5. Traversal (Penjelajahan) : Melalui seluruh hash table untuk memproses semua data yang ada dalam tabel.

4). COLLISION RESOLUTION (Resolusi Tabrakan)

Resolusi tabrakan (collision resolution) adalah teknik yang digunakan untuk menangani situasi di mana dua kunci yang berbeda menghasilkan hash value yang sama dalam hash table. Tabrakan ini dapat terjadi karena fungsi hash tidak sempurna dalam mendistribusikan kunci secara merata ke seluruh bucket (tempat penyimpanan) dalam hash table. Ada dua teknik untuk menyelesaikan masalah ini diantaranya :



- **Open Hashing (Chaining)**

Metode chaining mengatasi tabrakan dengan menyimpan semua item dengan nilai indeks yang sama dalam daftar tertaut. Setiap node dalam daftar tertaut mewakili satu item. Saat mencari atau menambahkan item, operasi dilakukan pada daftar tertaut yang sesuai dengan indeks terhitung dari kunci hash. Ketika daftar tertaut memiliki banyak node, pencarian atau penambahan item menjadi lambat karena perlu mencari seluruh daftar tertaut. Namun, rangkaian secara efektif menangani item dalam jumlah besar, karena menghindari keterbatasan array.

- **Closed Hashing (Addressing)**
  - **Linear Probing**

    Jika terjadi tabrakan, algoritme akan mencoba mencari slot kosong di bawah titik tabrakan. Jika slot sudah terisi, ia akan melanjutkan pencarian lebih jauh ke bawah hingga menemukan ruang yang tersedia. Tidak adanya slot kosong menandakan HashTable telah mencapai kapasitasnya.
  - **Quadratic Probing**

    Penangannya hampir sama dengan metode liniear, 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.

PENJELASAN GUIDED

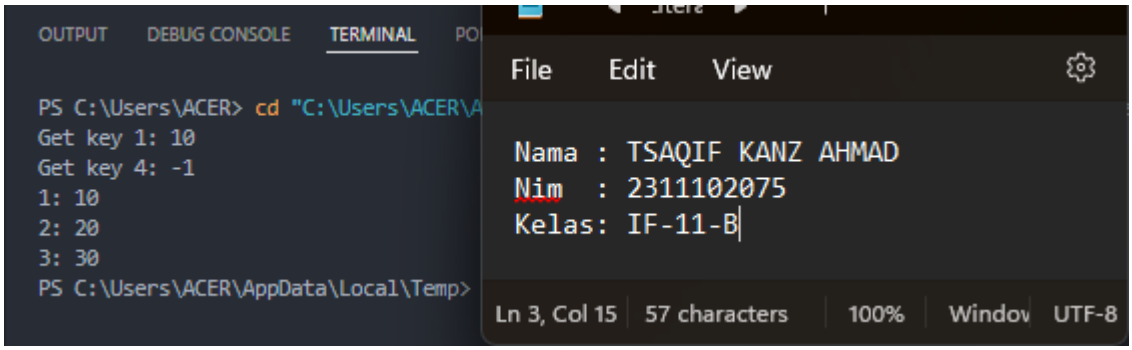
1). **SOURCE CODE**

```
#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;
}
```

SCREENSHOT OUTPUT



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.

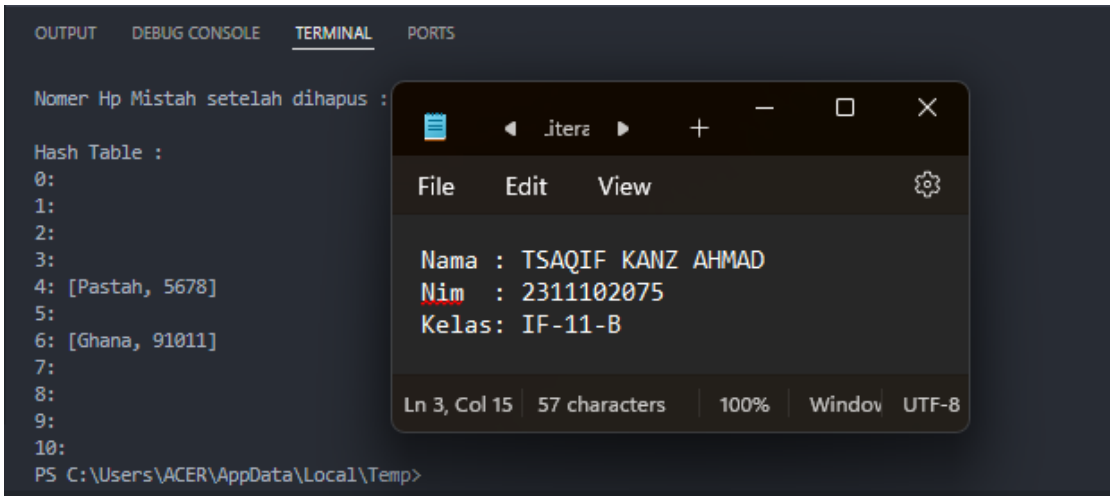
2). **SOURCE CODE**

```
#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;
}
```

SCREENSHOT OUTPUT :



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.

PENJELASAN 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

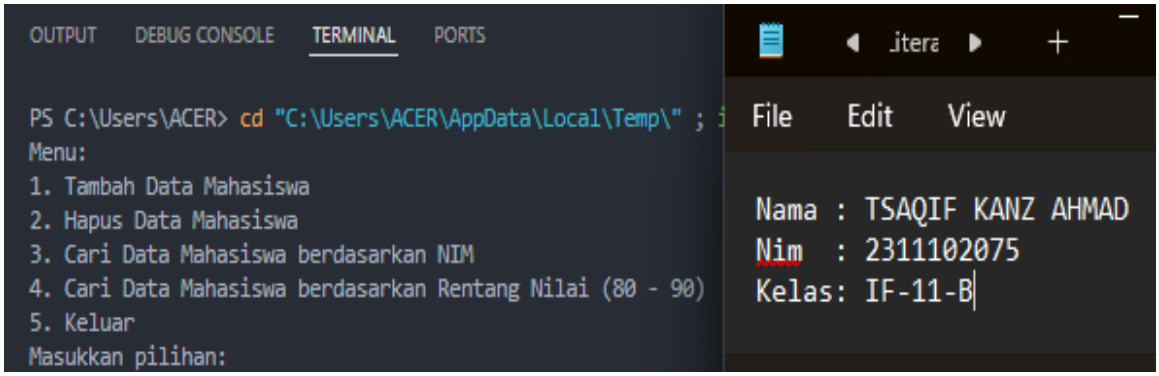
```
##include <iostream>
#include <vector>
using namespace std;

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

// Ukuran hash table
const int SIZE = 10;

// Hash table untuk menyimpan data mahasiswa
vector<Mahasiswa> hashTable[SIZE];

// Fungsi hash sederhana
int hashFunction(int nim) {
    return nim % SIZE;
}

// Fungsi untuk menambahkan data mahasiswa ke hash table
void tambahData(int nim, int nilai) {
    int index = hashFunction(nim);
    hashTable[index].push_back({nim, nilai});
}

// Fungsi untuk menghapus data mahasiswa dari hash table berdasarkan NIM
void hapusData(int nim) {
    int index = hashFunction(nim);
    for (int i = 0; i < hashTable[index].size(); i++) {
        if (hashTable[index][i].nim == nim) {
            hashTable[index].erase(hashTable[index].begin() + i);
            break;
        }
    }
}

// Fungsi untuk mencari data mahasiswa berdasarkan NIM
void cariByNIM(int nim) {
```

```
    int index = hashFunction(nim);
    for (int i = 0; i < hashTable[index].size(); i++) {
        if (hashTable[index][i].nim == nim) {
            cout << "Data ditemukan - NIM: " << hashTable[index][i].nim <<
", Nilai: " << hashTable[index][i].nilai << endl;
            return;
        }
    }
    cout << "Data tidak ditemukan." << endl;
}

// Fungsi untuk mencari data mahasiswa berdasarkan rentang nilai
void cariByNilai(int minNilai, int maxNilai) {
    for (int i = 0; i < SIZE; i++) {
        for (int j = 0; j < hashTable[i].size(); j++) {
            if (hashTable[i][j].nilai >= minNilai && hashTable[i][j].nilai
<= maxNilai) {
                cout << "NIM: " << hashTable[i][j].nim << ", Nilai: " <<
hashTable[i][j].nilai << endl;
            }
        }
    }
}

// Fungsi untuk menampilkan menu
void tampilkanMenu() {
    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";
}

int main() {
    int pilihan;
    do {
        tampilkanMenu();
        cout << "Masukkan pilihan: ";
        cin >> pilihan;

        switch (pilihan) {
            case 1: {
                int nim, nilai;
                cout << "Masukkan NIM: ";
                cin >> nim;
                cout << "Masukkan nilai: ";
                cin >> nilai;
                tambahData(nim, nilai);
                break;
            }
```

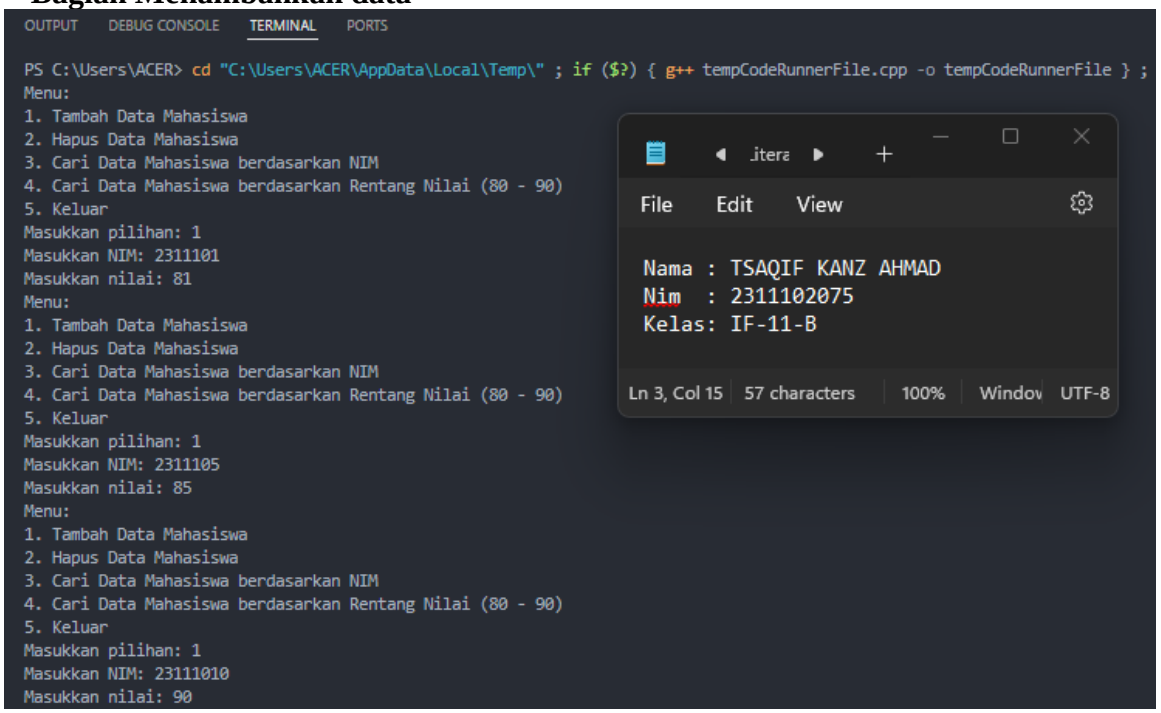
```
            case 2: {
                int nim;
                cout << "Masukkan NIM yang akan dihapus: ";
                cin >> nim;
                hapusData(nim);
                break;
            }
            case 3: {
                int nim;
                cout << "Masukkan NIM yang akan dicari: ";
                cin >> nim;
                cariByNIM(nim);
                break;
            }
            case 4: {
                cariByNilai(80, 90);
                break;
            }
            case 5:
                cout << "Program selesai.\n";
                break;
            default:
                cout << "Pilihan tidak valid.\n";
        }
    } while (pilihan != 5);

    return 0;
}
```

SCREENSHOT OUTPUT

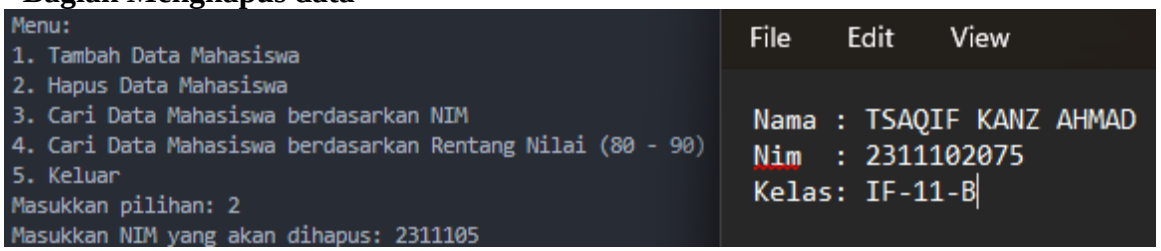
- **Menu**



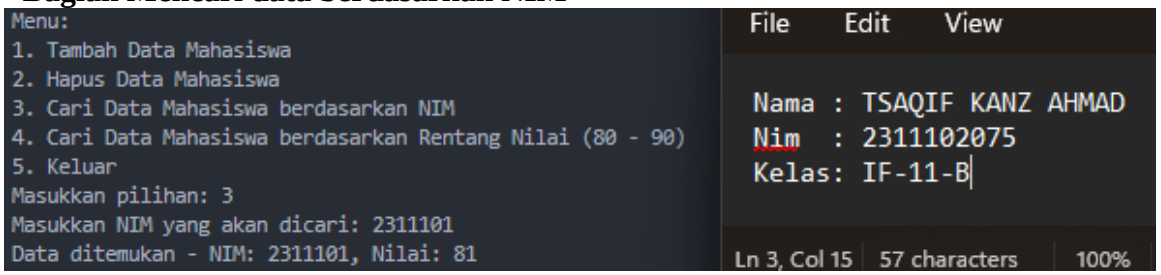
- **Bagian Menambahkan data**



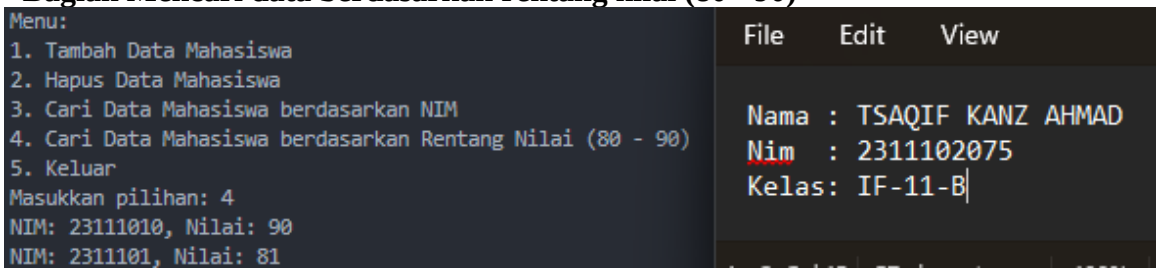
- **Bagian Menghapus data**



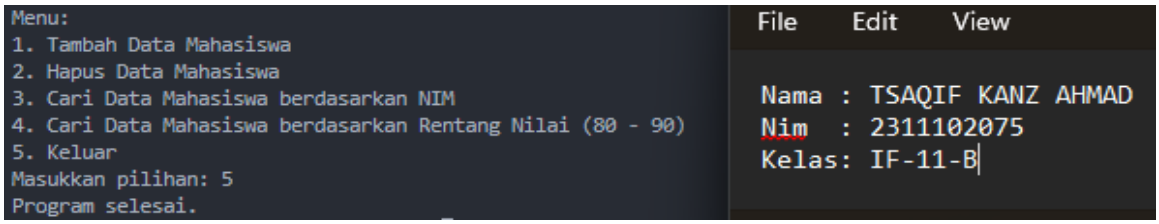
- **Bagian Mencari data berdasarkan NIM**



- **Bagian Mencari data berdasarkan rentang nilai (80 - 90)**



- Keluar



DESKRIPSI PROGRAM

Program diatas adalah Kode yang disediakan pada aplikasi konsol sederhana yang ditulis dalam C++ yang mengimplementasikan tabel hash untuk menyimpan data mahasiswa. Tabel hash memiliki ukuran 10 dan menggunakan fungsi hash sederhana untuk memetakan NIM (nomor induk mahasiswa) ke indeks dalam tabel hash. Tabel hash menyimpan array Mahasiswastruct, yang berisi NIM siswa dan nilainya. Program ini menyediakan menu bagi pengguna untuk berinteraksi dengan hash table diantaranya : Menambahkan data, Menghapus data, Mencari data berdasarkan NIM, Mencari data berdasarkan Rentang Nilai, dan yang terakhir adalah keluar yang menandakan program telah selesai. Program ini menggunakan operasi input/output dasar dan tidak menangani kasus kesalahan apa pun, seperti input tidak valid atau NIM tidak ada. Ini adalah demonstrasi sederhana penggunaan tabel hash untuk operasi dasar dengan data siswa.

KESIMPULAN

Hash table adalah struktur data yang digunakan untuk menyimpan data secara asosiatif, menghubungkan kunci dengan nilainya. Hash table bekerja dengan mengubah kunci menjadi nilai numerik (hash value) menggunakan fungsi hash, kemudian menyimpan data di lokasi tabel hash yang ditentukan oleh hash value. Hash table memiliki banyak aplikasi dalam berbagai bidang seperti basis data, compiler, dan sistem operasi. Memahami hash table penting dalam membentuk fondasi dari berbagai teknologi ini. Namun penting juga dalam memahami kelemahan dan potensi tabrakan untuk menggunakan secara efektif. Dengan memahami kelebihan dan kekurangannya, Pengguna dapat menilai apakah hash table cocok untuk kebutuhan program maupun sebaliknya.

DAFTAR PUSTAKA

[1] Asisten Praktikum, “Modul 5 Hash Table" : MODUL 5 - HASH TABLE.pdf

[2] GeeksforGeeks – Hash Table : https://www.geeksforgeeks.org/hash-table-data-structure/