

**LAPORAN PRAKTIKUM
STRUKTUR DATA DAN ALGORITMA**

MODUL V



DISUSUN OLEH :

Nama: Ibnu Rizal Mutaqim

Nim : (2311102067)

Dosen

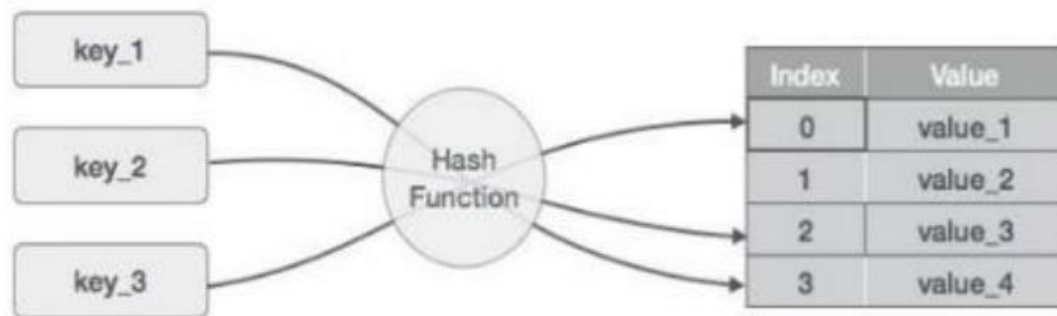
Wahyu Andi Saputra, S.Pd. , M.Eng

**PROGRAM STUDI S1 TEKNIK INFORMATIKA
FAKULTAS INFORMATIKA
INSTITUT TEKNOLOGI TELKOM PURWOKERTO
2024**

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 memetakannya 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



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

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:

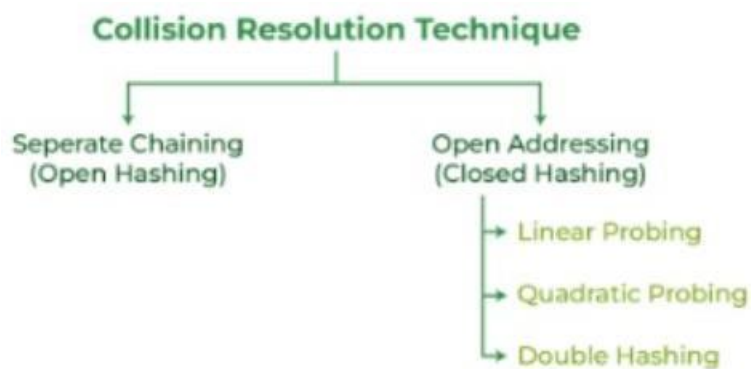
Memperbarui data dalam hash table dengan mencari data menggunakan fungsi hash, dan kemudian memperbarui data yang ditemukan.

5. Traversal:

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

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 :



- 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.

- Closed Hashing

● Linear Probing

Pada saat terjadi collision, maka akan mencari posisi yang kosong di bawah tempat terjadinya collision, jika masih penuh terus ke bawah, hingga ketemu tempat yang kosong. Jika tidak ada tempat yang kosong berarti HashTable sudah penuh.

● Quadratic Probing

Penanganannya hampir sama dengan metode linear, hanya lompatannya tidak satu-satu,

tetapi quadratic (12, 22, 32, 42, ...)

- Double Hashing

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

B. Guided

Guided 1

Sourcode :

```
#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)

```

```

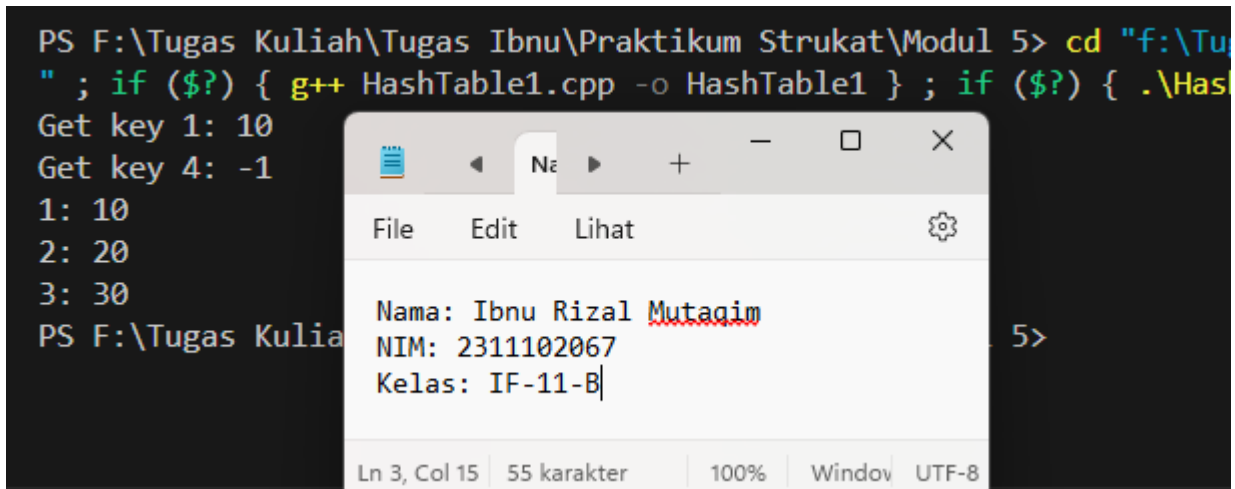
{
    if (current->key == key)
    {
        if (prev == nullptr)
        {
            table[index] = current->next;
        }
        else
        {
            prev->next = current->next;
        }
        delete current;
        return;
    }
    prev = current;
    current = current->next;
}

// 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;  
}
```


Output :



```
PS F:\Tugas Kuliah\Tugas Ibnu\Praktikum Strukat\Modul 5> cd "f:\Tugas Kuliah\Tugas Ibnu\Praktikum Strukat\Modul 5" ; if ($?) { g++ HashTable1.cpp -o HashTable1 } ; if ($?) { .\HashTable1.exe }
Get key 1: 10
Get key 4: -1
1: 10
2: 20
3: 30
PS F:\Tugas Kuliah\Tugas Ibnu\Praktikum Strukat\Modul 5>
```

Nama: Ibnu Rizal Mutaqim
NIM: 2311102067
Kelas: IF-11-B

Deskripsi :

Program ini mengimplementasikan tabel hash dasar dengan chaining terpisah untuk menangani tabrakan. Konstanta `MAX_SIZE` menetapkan ukuran tabel hash menjadi 10, dan fungsi hash menghitung indeks kunci menggunakan operator modulo. Struktur `Node` menyimpan pasangan kunci-nilai dan pointer ke node berikutnya. Kelas `HashTable` mengelola array pointer ke `Node`, dengan metode untuk penyisipan, pencarian, penghapusan, dan traversal. Metode `insert` menambahkan atau memperbarui pasangan kunci-nilai, `get` mengambil nilai berdasarkan kunci, `remove` menghapus pasangan kunci-nilai, dan `traverse` mencetak semua pasangan kunci-nilai dalam tabel. Fungsi `main` mendemonstrasikan penyisipan, pencarian, penghapusan, dan traversal dalam tabel hash.

Guided 2

Sourcode :

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

int search(string nim)
{
    int hash_val = hashFunc(nim);
    for (auto node : table[hash_val])
    {
        if (node->nim == nim)
        {

```

```

        return node->nilai;
    }
}

return -1; // return -1 if NIM is not found
}

void findNimByScore(int minScore, int maxScore)
{
    bool found = false;
    for (int i = 0; i < TABLE_SIZE; i++)
    {
        for (auto pair : table[i])
        {
            if (pair != nullptr && pair->nilai >= minScore && pair->nilai <= maxScore)
            {
                cout << pair->nim << " memiliki nilai " << pair->nilai << endl;
                found = true;
            }
        }
    }

    if (!found)
    {
        cout << "Tidak ada mahasiswa yang memiliki nilai antara " << minScore << " and " << maxScore << endl;
    }
}

void print()
{
    for (int i = 0; i < TABLE_SIZE; i++)
    {
        cout << i << ": ";
        for (auto pair : table[i])
        {
            if (pair != nullptr)
            {

```

```

        cout << "[" << pair->nim << ", " << pair->nilai << "];"
    }
}
cout << endl;
}
}
};

```

```

int main()
{
    HashMap data;
    string NIM;
    int nilai_mhs;
    while (true)
    {
        int menu;
        cout << "\nMenu" << endl;
        cout << "1. Tambah data" << endl;
        cout << "2. Hapus data" << endl;
        cout << "3. Mencari data berdasarkan NIM" << endl;
        cout << "4. Mencari data berdasarkan nilai" << endl;
        cout << "5. Cetak data" << endl;
        cout << "6. Exit" << endl;
        cout << "Masukkkkan pilihan : ";
        cin >> menu;
        switch (menu)
        {
            case 1:
                cout << "Masukkan NIM : ";
                cin >> NIM;
                cout << "Masukkan nilai : ";
                cin >> nilai_mhs;
                data.insert(NIM, nilai_mhs);

```

```

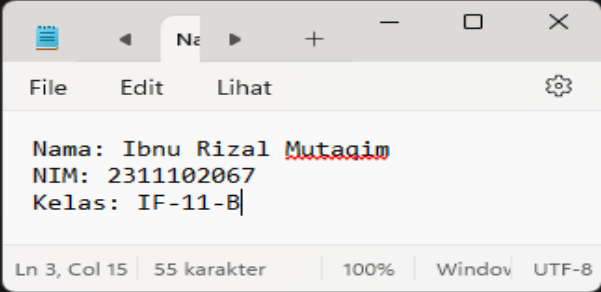
        break;
    case 2:
        cout << "Masukkan NIM yang akan dihapus : ";
        cin >> NIM;
        data.remove(NIM);
        cout << "Data berhasil dihapus" << endl;
        break;
    case 3:
        cout << "Masukkan NIM yang akan dicari : ";
        cin >> NIM;
        cout << "NIM " << NIM << " memiliki nilai " << data.search(NIM) << endl;
        break;
    case 4:
        int a, b;
        cout << "Masukkan rentang nilai minimal : ";
        cin >> a;
        cout << "Masukkan rentang nilai maksimal : ";
        cin >> b;
        data.findNimByScore(a, b);
        break;
    case 5:
        data.print();
        break;
    case 6:
        return 0;
    default:
        cout << "Menu tidak tersedia!" << endl;
        break;
    }
}
}

```

Output:

```
Nomer Hp Mistah : 1234
Phone Hp Pastah : 5678
Nomer Hp Mistah setelah dihapus :

Hash Table :
0:
1:
2:
3:
4: [Pastah, 5678]
5:
6: [Ghana, 91011]
7:
8:
9:
10:
PS F:\Tugas Kuliah\Tugas Ibnu\Praktikum Strukat\Modul 5>
```



Deskripsi :

Program ini mengimplementasikan tabel hash untuk menyimpan data buku telepon dengan nama sebagai kunci dan nomor telepon sebagai nilai. Ukuran tabel hash adalah 11. Kelas `HashNode` menyimpan pasangan `name` dan `phone_number`, sementara kelas `HashMap` mengelola array vektor pointer `HashNode` dan menyediakan metode untuk menyisipkan, menghapus, mencari, dan mencetak data. Metode `hashFunc` menghitung nilai hash dari nama. Metode `insert` menambahkan atau memperbarui pasangan nama dan nomor telepon, `remove` menghapus entri berdasarkan nama, `searchByName` mencari nomor telepon, dan `print` mencetak semua elemen tabel. Fungsi `main` menyisipkan tiga entri, melakukan pencarian, menghapus satu entri, dan mencetak tabel hash.

C. Unguided

Unguided 1

Sourcode :

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

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

```

// 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);

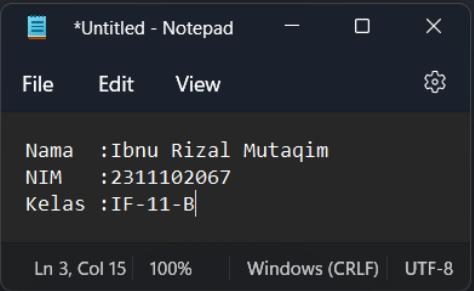
return 0;

}
```

Output :

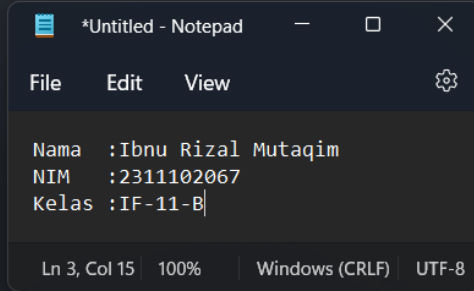
a. Bagian Menu

```
PS E:\Tugas Ibnu\Praktikum Strukat\Modul 5> cd "e:\Tugas Ibnu\Praktikum Strukat\Modul 5\" ; if ($?)
$?) { .\unguided }
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
Masukkan pilihan:
```



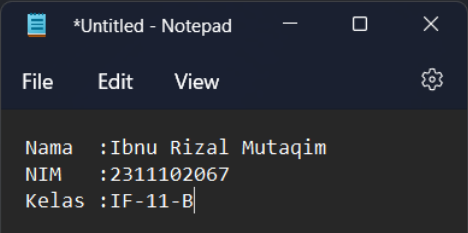
b. Tambah Data Mahasiswa

```
PS E:\Tugas Ibnu\Praktikum Strukat\Modul 5> cd "e:\Tugas Ibnu\Praktikum Strukat\Modul 5\" ; if ($?)
$?) { .\unguided }
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
Masukkan pilihan: 1
Masukkan NIM: 23031
Masukkan nilai: 89
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
Masukkan pilihan: 
```



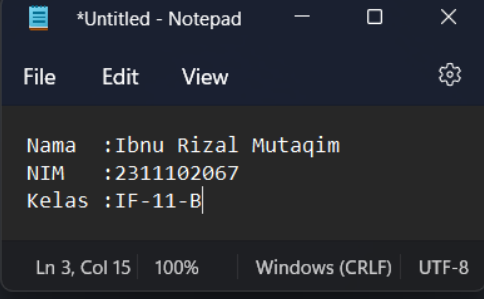
c. Hapus Data Mahasiswa

```
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
Masukkan pilihan: 2
Masukkan NIM yang akan dihapus: 23031
```



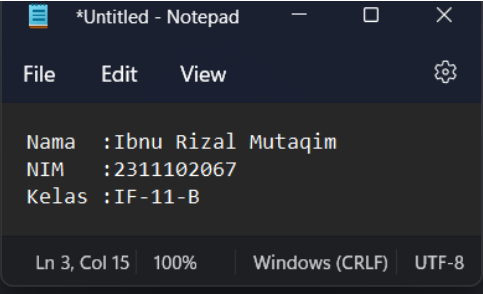
d. Cari Data Mahasiswa Berdasarkan Nim

```
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
Masukkan pilihan: 3
Masukkan NIM yang akan dicari: 2311
Data ditemukan - NIM: 2311, Nilai: 83
```



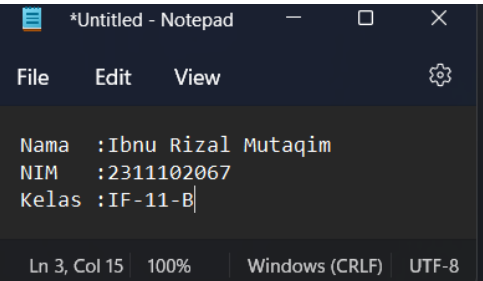
e. Cari Data Mahasiswa Berdasarkan Rentang Nilai (80-90)

```
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
Masukkan pilihan: 4
NIM: 2310, Nilai: 85
NIM: 2311, Nilai: 83
```



f. Keluar

```
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
Masukkan pilihan: 5
Program selesai.
PS E:\Tugas Ibnu\Praktikum Strukt\Modul 5>
```



Deskripsi :

Program ini merupakan implementasi sederhana dari hash table untuk menyimpan data mahasiswa berdasarkan NIM dan nilai. Hash table digunakan untuk mengorganisir data, dengan fungsi hash sederhana untuk menentukan indeks penyimpanan. Ada fungsi untuk menambah, menghapus, dan mencari data berdasarkan NIM atau rentang nilai. Program menampilkan menu operasi kepada pengguna dan menjalankan operasi yang dipilih hingga pengguna memilih untuk keluar. Ini adalah implementasi dasar untuk manajemen data mahasiswa menggunakan hash table.

D. Kesimpulan

Hash table adalah struktur data yang menyimpan data dalam pasangan key-value dan menggunakan fungsi hash untuk memetakan kunci ke indeks dalam tabel hash. Ini memungkinkan operasi penyisipan dan pencarian data yang cepat terlepas dari ukuran data. Teknik hashing mengubah kunci menjadi nilai lain, sementara linear probing digunakan untuk menangani tabrakan dengan mencari lokasi kosong berikutnya dalam tabel hash.

E. Referensi

Karumanchi, N. (2016). Data Structures and algorithms made easy: Concepts, problems, Interview Questions. CareerMonk Publications.

Algoritma

<https://algoritma.blog/hash-table-adalah-2022/>