

**LAPORAN  
STRUKTUR DATA DAN ALGORITMA**

**MODUL V**



**DISUSUN OLEH :**

Nama: Ahmad Titana Nanda Pramudya  
Nim : (2311102042)

**Dosen**

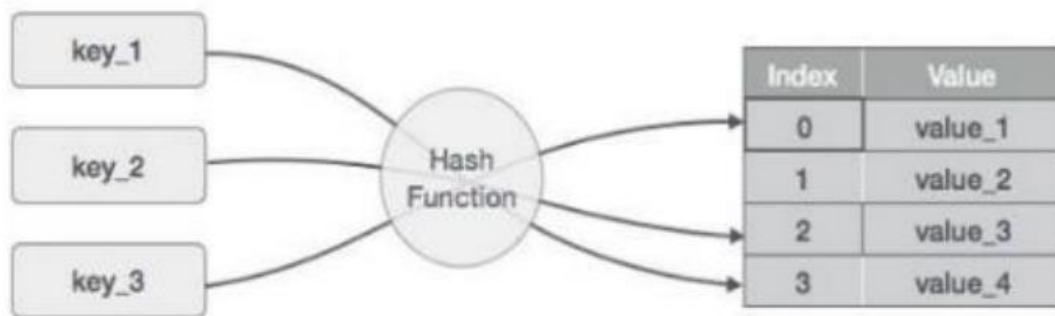
Wahyu Andi Saputra, S.Pd. , M.Eng

**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:

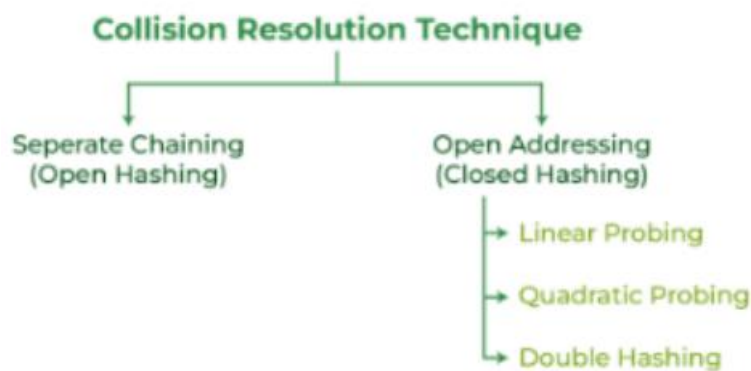
Memperbarui data dalam hash table dengan mencari data menggunakan fungsi hash, dan kemudian memperbarui data yang ditemukan.

#### 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;
        }
    }
}

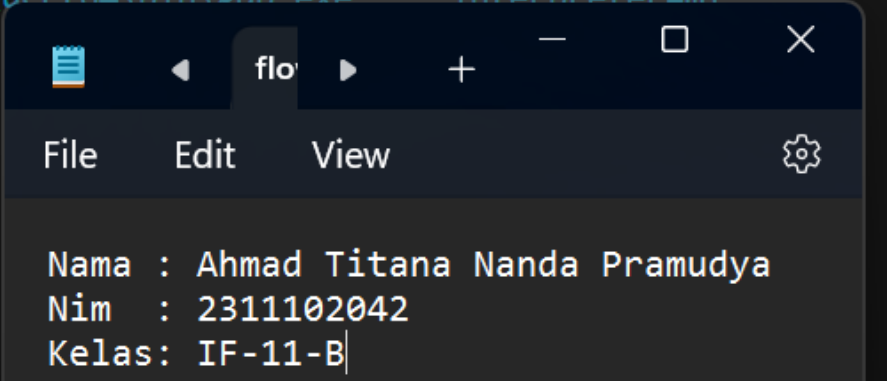
```

```
    }  
}  
};  
  
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 D:\contoh> & 'c:\Users\Eko Puji Susanto\.vscode\extension
s\bin\WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-1
kydr2dj.vcu' '--stderr=Microsoft-MIEngine-Error-qjbgb3yh.h0t
gExe=C:\msys64\ucrt64\bin\gdb.exe' '--interpreter=mi'
Get key 1: 10
Get key 4: -1
1: 10
2: 20
3: 30
PS D:\contoh>
```

A screenshot of a VS Code window. The background shows a terminal window with the output of a C program. The foreground shows a VS Code window with a menu bar (File, Edit, View) and a text area containing the following text:

```
Nama : Ahmad Titana Nanda Pramudya
Nim  : 2311102042
Kelas: IF-11-B|
```

Deskripsi :

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.

## 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;
}
}
};

```

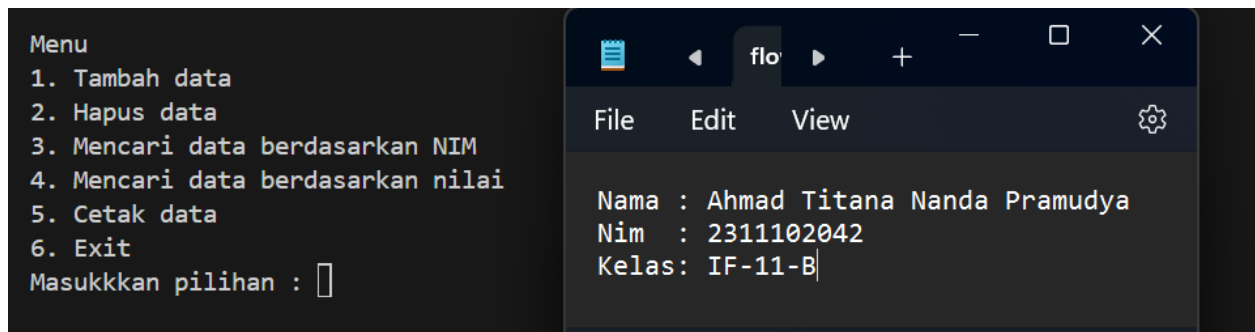
```

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:



The screenshot shows a terminal window with a dark background. On the left, a menu is displayed with six options: 1. Tambah data, 2. Hapus data, 3. Mencari data berdasarkan NIM, 4. Mencari data berdasarkan nilai, 5. Cetak data, and 6. Exit. Below the menu, the prompt 'Masukkan pilihan : ' is followed by a cursor. On the right, a separate window titled 'flow' is open, displaying a form with three fields: 'Nama : Ahmad Titana Nanda Pramudya', 'Nim : 2311102042', and 'Kelas: IF-11-B'.

```
Menu
1. Tambah data
2. Hapus data
3. Mencari data berdasarkan NIM
4. Mencari data berdasarkan nilai
5. Cetak data
6. Exit
Masukkan pilihan : 
```

```
flow
File Edit View
Nama : Ahmad Titana Nanda Pramudya
Nim : 2311102042
Kelas: IF-11-B
```

Deskripsi :

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

### C. Unguided

Unguided 1

Sourcode :

```
#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);

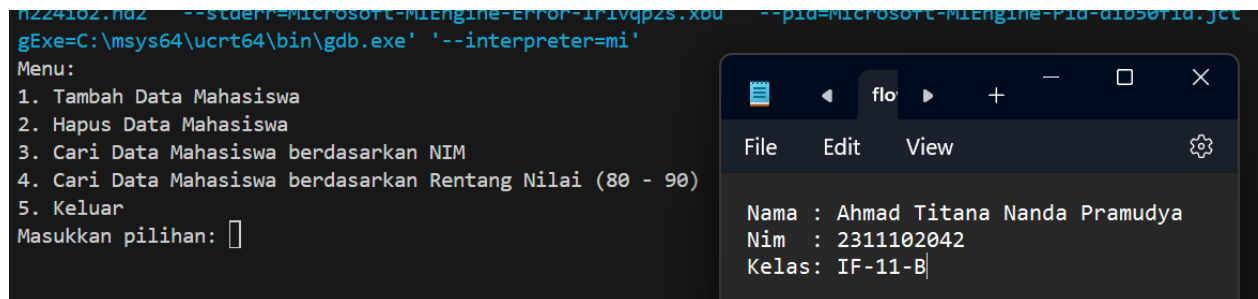
    return 0;

}

```

Output :

#### a. Bagian Menu



The screenshot shows a terminal window with a menu and a student information window. The terminal output is as follows:

```

n224102.nd2 --stderr=Microsoft-MIEngine-Error-11ivqps.xbu --pid=Microsoft-MIEngine-Pid-410501d.jct
gExe=C:\msys64\ucrt64\bin\gdb.exe' '--interpreter=mi'
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: 

```

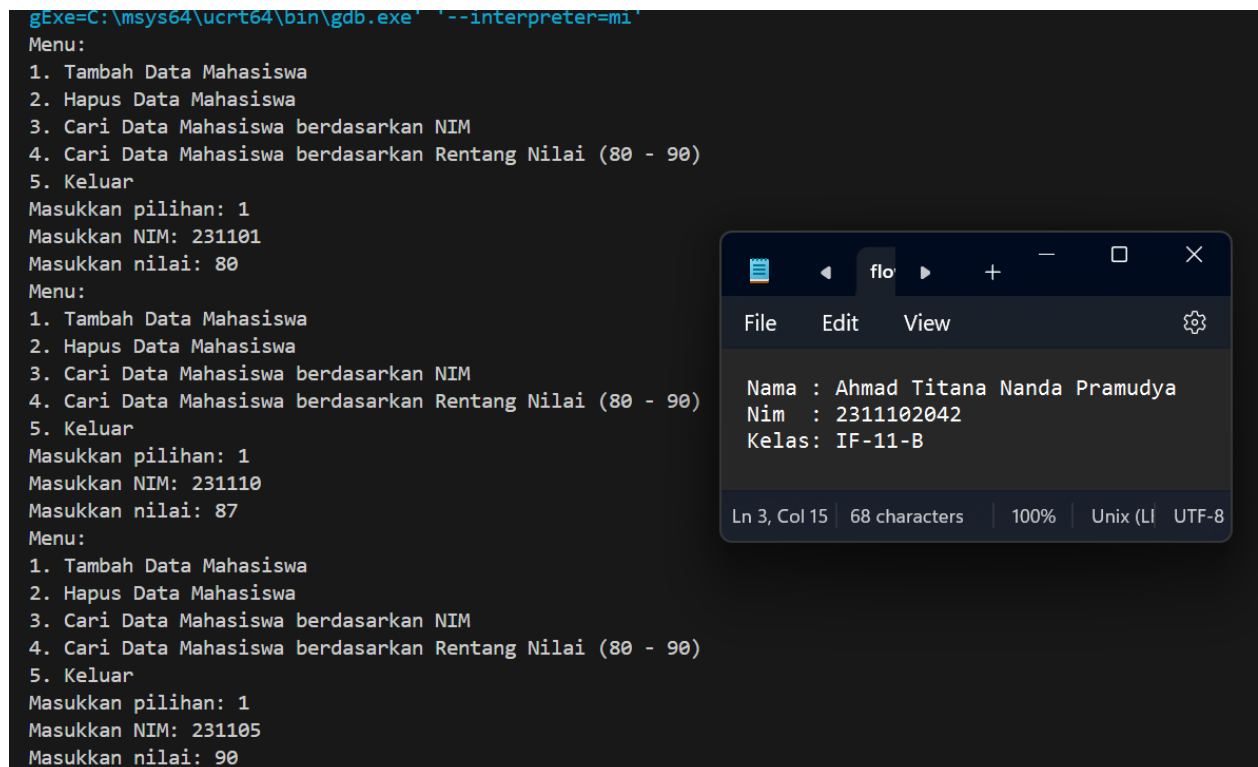
The student information window displays the following details:

```

Nama : Ahmad Titana Nanda Pramudya
Nim  : 2311102042
Kelas: IF-11-B

```

#### b. Tambah Data Mahasiswa



The screenshot shows a terminal window where a new student record is being added. The terminal output is as follows:

```

gExe=C:\msys64\ucrt64\bin\gdb.exe' '--interpreter=mi'
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: 231110
Masukkan nilai: 80
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: 231110
Masukkan nilai: 87
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: 231105
Masukkan nilai: 90

```

The student information window displays the following details:

```

Nama : Ahmad Titana Nanda Pramudya
Nim  : 2311102042
Kelas: IF-11-B

```

### 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: 231110
```

```
File Edit View

Nama : Ahmad Titana Nanda Pramudya
Nim : 2311102042
Kelas: IF-11-B|
```

```
Ln 3, Col 15 | 68 characters | 100% | Unix (LI UTF
```

### 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: 231105
Data ditemukan - NIM: 231105, Nilai: 90
```

```
File Edit View

Nama : Ahmad Titana Nanda Pramudya
Nim : 2311102042
Kelas: IF-11-B|
```

```
Ln 3, Col 15 | 68 characters | 100% | Unix (LI UTF
```

### 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: 231101, Nilai: 80
NIM: 231105, Nilai: 90
```

```
File Edit View

Nama : Ahmad Titana Nanda Pramudya
Nim : 2311102042
Kelas: IF-11-B|
```

### 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 D:\contoh>
```

```
File Edit View

Nama : Ahmad Titana Nanda Pramudya
Nim : 2311102042
Kelas: IF-11-B|
```

```
Ln 3, Col 15 | 68 characters | 100% | Unix (LI UTF-8
```

### Deskripsi :

Program di atas adalah implementasi sederhana dari hash table untuk menyimpan data mahasiswa. Program ini memiliki beberapa fitur: Struktur Data Mahasiswa: Program menggunakan struktur Mahasiswa yang memiliki dua atribut, yaitu nim (Nomor Induk Mahasiswa) dan nilai. Hash Table: Program menggunakan hash table untuk menyimpan data mahasiswa. Hash table ini memiliki ukuran tetap (const int SIZE = 10;) dan diimplementasikan sebagai array dari vektor vector<Mahasiswa> hashTable[SIZE];. Fungsi hash sederhana digunakan untuk menentukan indeks di mana data mahasiswa akan disimpan dalam hash table. Dan ada Beberapa menu yaitu : Menambah data mahasiswa, Menghapus data mahasiswa ,Mencari data mahasiswa berdasarkan NIM, Mencari data mahasiswa berdasarkan rentang nilai (80 - 90). Keluar dari program. Main Function: Program menggunakan loop do-while untuk menampilkan menu dan memproses pilihan pengguna. Loop tersebut berakhir ketika pengguna memilih untuk keluar

#### **D. Kesimpulan**

Hash table adalah struktur data yang memberikan cara efisien untuk menyimpan dan mengakses data dengan cepat melalui penggunaan fungsi hash. Dengan menggunakan hash table, kita dapat mengakses elemen dengan waktu konstan dalam kebanyakan kasus, membuatnya cocok untuk aplikasi di mana kecepatan akses data menjadi prioritas utama. Namun, penting untuk memperhatikan strategi penanganan tabrakan (collision) agar tidak mengurangi kinerja hash table, seperti dengan menggunakan teknik chaining atau probing. Dengan memahami dan menerapkan hash table dengan baik, kita dapat meningkatkan efisiensi dan kinerja program .

#### **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/>