

LAPORAN PRAKTIKUM

MODUL V “HASH TABLE”



Disusun oleh:

Shiva Indah Kurnia
NIM 2311102035

Dosen Pengampu:

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

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

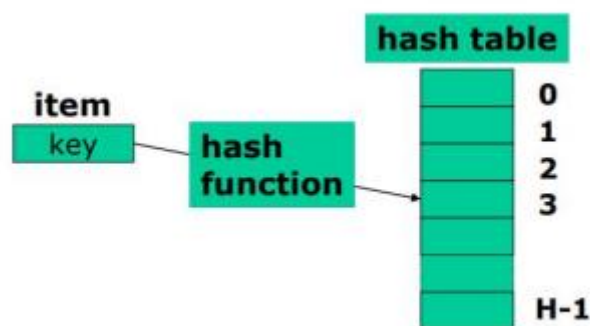
TUJUAN PRAKTIKUM

Setelah melakukan praktikum ini diharapkan mahasiswa dapat:

1. Mahasiswa diharapkan mampu menjelaskan dan memahami konsep dari Hash Table itu sendiri.
2. Mahasiswa diharapkan mampu mengaplikasikan Hash Code kedalam pemrograman C++.
3. Mahasiswa diharapkan mampu menyelesaikan studi kasus menggunakan penyelesaian program Hash Table.

DASAR TEORI

Tabel hash, juga dikenal sebagai peta hash, adalah struktur data yang digunakan untuk menyimpan catatan, di mana setiap data diakses menggunakan kunci yang dihitung oleh fungsi hash. Fungsi hash adalah algoritme yang digunakan untuk membuat indeks unik, yang kemudian digunakan untuk menemukan data yang cocok dengan kunci. Dalam tabel hash, setiap elemen disimpan dalam "ember" atau "slot" yang sesuai dengan indeks yang dikembalikan oleh fungsi hash.



Gambar 1. Struktur Hash Table

Keuntungan menggunakan tabel hash adalah waktu akses yang cepat dan efisien, karena penyajian data tidak memerlukan proses iteratif atau pencarian linier ke seluruh kumpulan data. Akan tetapi penggunaan tabel hash membutuhkan alokasi memori yang cukup besar dan juga memerlukan perhatian khusus dalam merancang dan mengimplementasikan fungsi hash yang baik untuk meminimalisir kemungkinan terjadinya collision (keduanya menggunakan indeks yang sama). Tabel hash sering digunakan dalam implementasi berbagai aplikasi seperti sistem basis data, pencarian kata kunci di mesin pencari, manajemen cache, dan banyak lainnya.

Terdapat beberapa jenis fungsi hash yang dapat digunakan dalam struktur data hash table, antara lain:

1. Fungsi hash modulo, Fungsi ini menghasilkan indeks dengan menggunakan operasi modulo terhadap nilai hash dari kunci data dengan ukuran tabel hash.
2. Fungsi hash perkalian, Fungsi ini menghasilkan indeks dengan melakukan operasi perkalian antara nilai hash dari kunci data dengan sebuah konstanta dan kemudian mengambil sebagian bit dari hasil perkalian tersebut sebagai indeks.

3. Fungsi hash berbasis kunci, Fungsi ini menghasilkan nilai hash dengan melakukan operasi pada setiap karakter kunci data dan menggabungkan hasilnya.
4. Fungsi hash berbasis nilai acak, Fungsi ini menghasilkan nilai hash dengan menggunakan nilai acak sebagai input dan kemudian melakukan beberapa operasi matematika dan logika pada nilai tersebut.

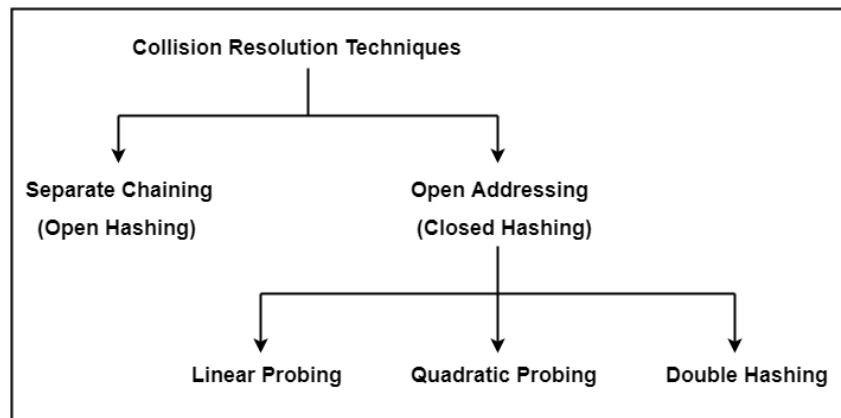
Pemilihan tipe fungsi hash yang tepat tergantung pada jenis data yang akan disimpan dalam hash table, dan juga penggunaan hash table itu sendiri. Sebagai contoh, fungsi hash modulo sering digunakan untuk data numerik, sedangkan fungsi hash berbasis kunci lebih cocok untuk data teks.

Berikut adalah beberapa operasi hash table yang umum digunakan:

1. Insert, Operasi ini digunakan untuk menambahkan elemen baru ke dalam hash table dengan menyimpan kunci dan nilai data yang sesuai ke dalam bucket yang dihasilkan dari fungsi hash.
2. Get, Operasi ini digunakan untuk mengakses nilai data yang terkait dengan kunci tertentu dalam hash table. Proses ini melibatkan penghitungan fungsi hash untuk mendapatkan indeks bucket yang sesuai, dan kemudian mengambil nilai data yang tersimpan di dalam bucket tersebut.
3. Remove, Operasi ini digunakan untuk menghapus elemen dengan kunci tertentu dari hash table. Proses ini melibatkan penghitungan fungsi hash untuk mendapatkan indeks bucket yang sesuai, dan kemudian menghapus elemen yang tersimpan di dalam bucket tersebut.
4. Resize, Operasi ini digunakan untuk menyesuaikan ukuran hash table ketika jumlah elemen dalam hash table terlalu besar atau terlalu kecil. Proses ini melibatkan pembuatan hash table yang lebih besar atau lebih kecil, dan kemudian memindahkan elemen-elemen yang ada dari hash table lama ke hash table yang baru.
5. Iterate, Operasi ini digunakan untuk mengakses seluruh elemen dalam hash table secara berurutan. Proses ini melibatkan iterasi melalui setiap bucket di dalam hash table, dan kemudian mengambil nilai data dari masing-masing bucket.

Operasi-operasi di atas dapat dikombinasikan dengan operasi lain, seperti update, clear, dan lain sebagainya, tergantung pada implementasi hash table yang digunakan.

Collision resolution adalah suatu metode untuk menangani masalah collision atau tumpang tindih pada hash table. Collision terjadi ketika dua atau lebih kunci data menghasilkan indeks hash yang sama dan mencoba untuk disimpan di dalam bucket yang sama. Ketika collision terjadi, hash table harus menentukan bagaimana cara menangani masalah tersebut agar tidak terjadi kehilangan data atau pencarian data yang tidak efisien.



Gambar 2. Teknik Collision

GUIDED

1. 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);
}

```

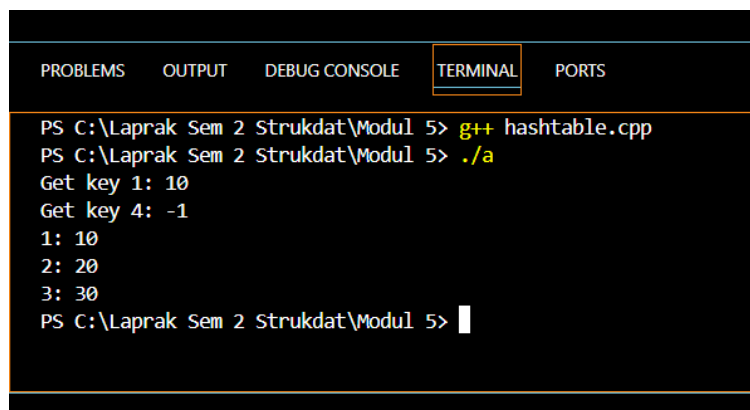


```

    // Traversal
    ht.traverse();
    return 0;
}

```

Screenshot program



```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Laprak Sem 2 Strukdat\Modul 5> g++ hashtable.cpp
PS C:\Laprak Sem 2 Strukdat\Modul 5> ./a
Get key 1: 10
Get key 4: -1
1: 10
2: 20
3: 30
PS C:\Laprak Sem 2 Strukdat\Modul 5>

```

Deskripsi Program

Pada awal program, konstanta MAX_SIZE didefinisikan sebagai 10. Kemudian, struktur node didefinisikan yang menyimpan dua variabel, yaitu kunci dan nilai, dan penunjuk berikutnya yang menunjuk ke node berikutnya. dalam daftar tertaut. Kemudian ada definisi class HashTable dengan variabel array berupa array pointer node dengan ukuran MAX_SIZE. Pada fungsi insert, nilai key dan value menyimpan nilai kunci dan nilai di node baru, setelah itu node ditambahkan ke indeks yang dibuat oleh fungsi hash. Jika sebuah simpul ada di indeks itu, kami memeriksa apakah kunci sudah ada di simpul itu. Jika ya, nilainya diperbarui dengan nilai baru, jika tidak, maka simpul baru ditambahkan ke bagian atas daftar tertaut. Pada fungsi get, nilai kunci di-hash untuk mendapatkan indeks, kemudian penelusuran daftar tertaut dari indeks dilakukan untuk menemukan node dengan kunci yang sama. Jika ditemukan, nilai node dikembalikan, jika tidak, nilai -1 dikembalikan.

Dalam remove, nilai kunci di-hash untuk mendapatkan indeks, setelah itu daftar tertaut dari indeks dilalui untuk menemukan node dengan kunci yang sama. Jika node ditemukan, node tersebut dihapus dari linked list dan memori yang digunakan untuk node dibebaskan. Jika tidak ditemukan, tidak ada tindakan yang diambil. Fungsi transverse melewati semua linked list di setiap indeks dan kemudian

mengembalikan kunci dan nilai dari setiap node. Tabel hash dibuat di fungsi utama, setelah itu fungsi tambahan dilakukan untuk beberapa pasangan nilai kunci. Kemudian operasi get dilakukan pada kunci yang ada dan tidak ada di tabel hash, operasi hapus dilakukan pada kunci yang tidak ada di tabel hash, dan operasi silang mengembalikan seluruh isi hash. Meja.Program kemudian mengembalikan nilai 0 sebagai tanda bahwa program berhasil.

2. 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;
}

```

Screenshot Program

```
PS C:\Laparak Sem 2 Strukdat\Modul 5> g++ hashtableguided2.cpp
PS C:\Laparak Sem 2 Strukdat\Modul 5> ./a
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 C:\Laparak Sem 2 Strukdat\Modul 5> |
```

Deskripsi program

Program diatas adalah program yang menggunakan struktur data hash table untuk menyimpan pasangan key-value berupa nama dan nomor telepon karyawan. Pada awal program, terdapat deklarasi variabel global berupa name dan phone_number yang nantinya akan digunakan pada kelas HashNode.

Class HashNode digunakan untuk merepresentasikan satu node pada hash table, dimana setiap node terdiri dari dua variabel yaitu name dan phone_number. Kelas HashMap adalah kelas utama yang akan digunakan untuk membuat, menghapus, mencari, dan mencetak pasangan key-value pada hash table. Hash table yang digunakan dalam program ini berukuran 11 dan diimplementasikan menggunakan vector.

Fungsi hashFunc digunakan untuk menghitung nilai hash dari suatu string key menggunakan metode simple sum. Fungsi insert digunakan untuk memasukkan pasangan key-value baru ke dalam hash table. Apabila terdapat key yang sama, maka value akan diupdate dengan value yang baru. Fungsi remove digunakan untuk menghapus pasangan key-value dari hash table berdasarkan key yang diberikan.

Fungsi searchByName digunakan untuk mencari value dari suatu key pada hash table. Fungsi print digunakan untuk mencetak seluruh pasangan key-value pada hash table. Di dalam fungsi main, terdapat deklarasi objek dari kelas HashMap dengan nama employee_map. Kemudian, beberapa pasangan key-value dimasukkan ke dalam hash

table menggunakan fungsi insert. Kemudian, dilakukan pencarian nomor telepon karyawan dengan menggunakan fungsi `searchByName` dan mencetak hasilnya. Setelah itu, dilakukan penghapusan pasangan key-value dengan menggunakan fungsi `remove`, dan mencetak kembali hasil pencarian nomor telepon karyawan untuk memastikan bahwa pasangan key-value telah dihapus dari hash table. Terakhir, dilakukan pencetakan seluruh pasangan key-value pada hash table menggunakan fungsi `print`. Program diakhiri dengan mengembalikan nilai 0.

UNGUIDED

1. Unguided 1

```
//Shiva Indah Kurnia
//2311102035

#include <iostream>
#include <unordered_map>
#include <vector>

using namespace std;

// Struktur data untuk mahasiswa
struct Mahasiswa {
    string nim;
    string nama; // Menambahkan variabel nama
    int nilai;
};

// Class HashTable untuk menyimpan data mahasiswa
class HashTable {
private:
    unordered_map<string, Mahasiswa> data;
public:
    // Fungsi untuk menambahkan data mahasiswa
    void tambahData(const Mahasiswa& mhs) {
        data[mhs.nim] = mhs;
    }

    // Fungsi untuk menghapus data mahasiswa berdasarkan NIM
    void hapusData(const string& nim) {
        data.erase(nim);
    }

    // Fungsi untuk mencari data mahasiswa berdasarkan NIM
    Mahasiswa* cariByNIM(const string& nim) {
        if (data.find(nim) != data.end()) {
            return &data[nim];
        }
        return nullptr;
    }
}
```

```

        // Fungsi untuk mencari data mahasiswa berdasarkan rentang
        nilai (80 - 90)
        vector<Mahasiswa> cariByNilai() {
            vector<Mahasiswa> result;
            for (auto& pair : data) {
                if (pair.second.nilai >= 80 && pair.second.nilai <=
90) {
                    result.push_back(pair.second);
                }
            }
            return result;
        }
};

// 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";
    cout << "Pilih : ";
}

int main() {
    HashTable hashTable;
    int pilihan;
    string nim;
    string nama; // Menambahkan variabel nama
    int nilai;

    do {
        tampilkanMenu();
        cin >> pilihan;

        switch (pilihan) {
            case 1: {
                Mahasiswa mhs;
                cout << "Masukkan NIM : ";
                cin >> mhs.nim;
                cout << "Masukkan nama : ";
                cin.ignore(); // Membersihkan buffer sebelum
membaca string

```



```

        getline(cin, mhs.nama); // Menggunakan getline
untuk membaca nama dengan spasi
        cout << "Masukkan nilai : ";
        cin >> mhs.nilai;
        hashTable.tambahData(mhs);
        break;
    }
    case 2: {
        cout << "Masukkan NIM mahasiswa yang akan
dihapus : ";
        cin >> nim;
        hashTable.hapusData(nim);
        break;
    }
    case 3: {
        cout << "Masukkan NIM mahasiswa yang akan dicari
: ";
        cin >> nim;
        Mahasiswa* mhs = hashTable.cariByNIM(nim);
        if (mhs != nullptr) {
            cout << "NIM : " << mhs->nim << ", Nama : "
<< mhs->nama << ", Nilai : " << mhs->nilai << endl;
        } else {
            cout << "Mahasiswa dengan NIM tersebut tidak
ditemukan!\n";
        }
        break;
    }
    case 4: {
        vector<Mahasiswa> result =
hashTable.cariByNilai();
        if (result.empty()) {
            cout << "Tidak ada mahasiswa dengan nilai di
rentang (80 - 90)!\n";
        } else {
            cout << "Mahasiswa dengan nilai di rentang
(80 - 90) :\n";
            for (const Mahasiswa& mhs : result) {
                cout << "NIM : " << mhs.nim << ", Nama :
" << mhs.nama << ", Nilai : " << mhs.nilai << endl;
            }
        }
        break;
    }
    case 5:

```

```

        cout << "Terima kasih!\n";
        break;
    default:
        cout << "Pilihan tidak valid. Silakan pilih
kembali!\n";
    }

    } while (pilihan != 5);

    return 0;
}

```

Screenshot program

```

PS C:\Laparak Sem 2 Strukdat\Modul 5> g++ hashtableunguided2.cpp
PS C:\Laparak Sem 2 Strukdat\Modul 5> ./a

```

```

-----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
Pilih : 1
Masukkan NIM : 2311102035
Masukkan nama : Shiva Indah
Masukkan nilai : 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
Pilih : 3
Masukkan NIM mahasiswa yang akan dicari : 2311102035
NIM : 2311102035, Nama : Shiva Indah, Nilai : 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
Pilih : 4
Mahasiswa dengan nilai di rentang (80 - 90) :
NIM : 2311102035, Nama : Shiva Indah, Nilai : 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

```

```

Pilih : 2
Masukkan NIM mahasiswa yang akan dihapus : 2311102035
-----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
Pilih : █

```

Deskripsi program

Pada fungsi addData(), pengguna diminta untuk memasukkan NIM dan nilai mahasiswa yang akan ditambahkan. Data mahasiswa tersebut kemudian di-hash menggunakan fungsi hashFunction() dan disimpan ke dalam linked list pada indeks hash yang dihasilkan.

Pada fungsi searchDataByNIM(), pengguna diminta untuk memasukkan NIM mahasiswa yang ingin dicari. Data mahasiswa kemudian dicari di linked list pada indeks hash yang dihasilkan. Jika data ditemukan, program akan menampilkan NIM dan nilai mahasiswa tersebut.

Pada fungsi searchDataByRange(), program akan mencari semua data mahasiswa yang memiliki nilai di antara 80 dan 90. Jika ada data yang ditemukan, program akan menampilkan NI dan nilai dari semua data mahasiswa tersebut.

Pada fungsi deleteData(), pengguna diminta untuk memasukkan NIM mahasiswa yang ingin dihapus. Data mahasiswa tersebut kemudian dicari di linked list pada indeks hash yang dihasilkan. Jika data ditemukan, data tersebut akan dihapus dari linked list. Jika data tidak ditemukan, program akan menampilkan pesan bahwa data tidak ditemukan.

Program ini memiliki menu utama dengan empat pilihan: menambahkan data mahasiswa, mencari data mahasiswa berdasarkan NIM, mencari data mahasiswa berdasarkan rentang nilai (80 - 90), dan menghapus data mahasiswa berdasarkan NIM. Pengguna dapat memilih salah satu opsi dengan memasukkan nomor pilihan. Setelah selesai melakukan operasi, pengguna diminta untuk memilih apakah ingin melanjutkan atau tidak dengan memilih y atau n. Program akan terus berjalan sampai pengguna memilih untuk tidak melanjutkan.

KESIMPULAN

Hash table atau tabel hash adalah struktur data yang memungkinkan pengindeksan dan pencarian data secara efisien. Hash table menggunakan fungsi hash untuk mengonversi kunci data menjadi indeks tabel. Pada C++, implementasi hash table dapat menggunakan struktur data berupa array dan linked list. Pada program di atas, hash table diimplementasikan menggunakan array dari linked list. Hash table sangat berguna dalam pengolahan data yang besar, seperti pada database atau aplikasi yang membutuhkan pencarian data yang cepat dan efisien. Selain itu, di modul kali ini kita juga berkesempatan untuk mengaplikasikan beberapa operasi hash table yang umum digunakan seperti Insert (Operasi ini digunakan untuk menambahkan elemen baru), Get (Operasi ini digunakan untuk mengakses nilai data yang terkait dengan kunci tertentu dalam hash table), Remove (Operasi ini digunakan untuk menghapus elemen dengan kunci tertentu dari hash table), dan Iterate (Operasi ini digunakan untuk mengakses seluruh elemen dalam hash table secara berurutan).