LAPORAN PRAKTIKUM

MODUL V
HASH TABLE

Disusun oleh:
Aryo Tegar Sukarno
NIM: 2311102018

Dosen Pengampu:
Wahyu Andi Saputra, S.Pd., M.Eng

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

BAB I

TUJUAN PRAKTIKUM

1. Mahasiswa memahami perbedaan konsep dan definisi Hash Code
2. Mahasiswa mampu menerapkan Hash Code dalam pemrograman

BAB II

DASAR TEORI

a. Pengertian Hash Table

Hash table (tabel hash) adalah struktur data yang digunakan untuk menyimpan data secara efisien berdasarkan prinsip pemetaan nilai kunci (key) menjadi lokasi penyimpanan.

Konsep dasarnya, tabel hash menggunakan sebuah array dan fungsi khusus yang disebut fungsi hash (hash function) untuk menentukan lokasi penyimpanan data.

Berikut komponen penting dalam tabel hash:

- Array: Media penyimpanan data utama.
- Key (nilai kunci): Nilai unik yang digunakan untuk identifikasi data.
- Hash function (fungsi hash): Fungsi yang mengubah key menjadi index (alamat) pada array.
- Value (nilai): Data yang sebenarnya ingin disimpan yang dikaitkan dengan key.

Prinsip kerja tabel hash**:**

1. Ketika data baru hendak disimpan, key dari data tersebut akan diproses oleh fungsi hash untuk menghasilkan index (alamat) pada array.
2. Data kemudian disimpan pada index tersebut pada array.

Keuntungan menggunakan tabel hash**:**

- Pencarian data yang cepat**:** Pencarian data bisa dilakukan dengan waktu rata-rata operasi pencariam O(1) (konstan), asalkan fungsi hash dirancang dengan baik dan jarang terjadi tabrakan (collusion). Tabrakan terjadi ketika fungsi hash menghasilkan index yang sama untuk key yang berbeda.
- Memproses data berdasarkan key secara efisien**:** Cocok digunakan untuk implementasi struktur data seperti dictionary atau map.

Jika tertarik untuk mempelajari lebih lanjut, anda bisa mencari referensi tentang penanganan tabrakan (collision handling) pada tabel hash.

BAB III

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);
    ht.insert(4, 40);

    // 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;
}
```

Screenshoot program

```
PS C:\Users\aryos> cd 'c:\Users\aryos\Downloads\PROKTIKUM SEMESTER 2\M
PS C:\Users\aryos\Downloads\PROKTIKUM SEMESTER 2\MODUL 6\output> & .\
Get key 1: 10
Get key 4: 40
1 : 10
2 : 20
3 : 30
PS C:\Users\aryos\Downloads\PROKTIKUM SEMESTER 2\MODUL 6\output>
```

Deskripsi program

Program hash table yang Anda buat di C++ menunjukkan cara mengimplementasikan tabel hash menggunakan teknik chaining untuk menangani tabrakan (collision).

2. Guided 2

Source code

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

using namespace std;

// ukuran tabel hash
const int TABLE_SIZE = 11;

string name; //deklarasi variabel string name
```

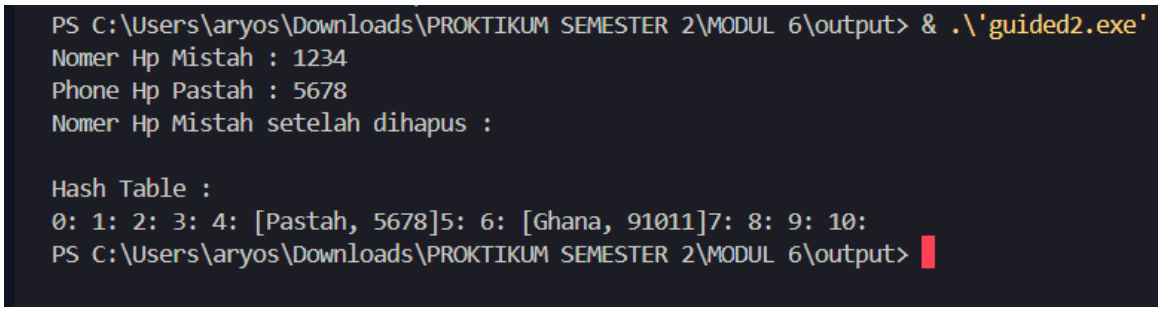
```
string phone_number; //deklarasi variabel string phone_number

// Struktur Data Untuk Setiap Node
class HashNode
{
//deklarasi variabel name dan phone_number
public:
    string name;
    string phone_number;

    HashNode(string name, string phone_number)
    {
        this->name = name;
        this->phone_number = phone_number;
    }
};

// Class HashMap
class HashMap
{
private:
    vector<HashNode*> table[TABLE_SIZE];

public:
    // Fungsi Hash Sederhana
    int hashFunc(string key)
    {
        int hash_val = 0;
        for (char c : key)
        {
            hash_val += c;
        }
        return hash_val % TABLE_SIZE;
    }
```

```
    // Tambah data
    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));
    }

    // Hapus data
    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;
            }
        }
    }

    // Cari data berdasarkan nama
    string searchByName(string name)
```

```
    {
        int hash_val = hashFunc(name);
        for (auto node : table[hash_val])
        {
            if (node->name == name)
            {
                return node->phone_number;
            }
        }
        return "";
    }

    // Cetak data
    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 << "]";
                }
            }
        }
    }
};

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;
}
```

Screenshoot program



Deskripsi program

Program di atas adalah implementasi sederhana dari tabel hash (hash table) di C++ yang digunakan untuk menyimpan dan mengelola data pasangan nama dan nomor telepon.

LATIHAN KELAS - UNGUIDED

1. Unguided 1

Source code

```
#include <iostream>
// Aryo Tegar Sukarno/2211102018/IF-11-A
#include <unordered_map>
#include <vector>
using namespace std;
struct Mahasiswa
{
    string NIM;
    int nilai;
};
class HashTable
{
private:
    unordered_map<string, Mahasiswa> tabel;

public:
    void tambahData(Mahasiswa mahasiswa)
    {
        tabel[mahasiswa.NIM] = mahasiswa;
    }
    void hapusData(string NIM)
    {
        tabel.erase(NIM);
    }
    Mahasiswa cariDataBerdasarkanNIM(string NIM)
    {
        if (tabel.find(NIM) != tabel.end())
        {
            return tabel[NIM];
```

```
        }
        else
        {
            throw "NIM tidak ditemukan";
        }
    }
    vector<Mahasiswa>       cariDataBerdasarkanRentangNilai(int
nilaiMin,  int nilaiMax)
    {
        vector<Mahasiswa> hasil;
        for (auto it = tabel.begin(); it != tabel.end(); it++)
        {
            if   (it->second.nilai   >=   nilaiMin   &&   it   ->
second.nilai <= nilaiMax)
            {
                hasil.push_back(it->second);
            }
        }
        return hasil;
    }
};
int main()
{
    HashTable hashTable;
    int pilihan;
    do
    {
        cout << "Menu:" << endl;
        cout << "1. Tambah data baru" << endl;
        cout << "2. Hapus data" << endl;
        cout << "3. Cari data berdasarkan NIM" << endl;
 cout  <<  "4.  Cari  data  berdasarkan  rentang  nilai  (80-90)" <<
endl;
 cout << "5. Keluar" << endl;
```

```
 cout << "Masukkan pilihan: ";
 cin >> pilihan;
 if (pilihan == 1)
 {
     Mahasiswa mahasiswa;
     cout << "Masukkan NIM: ";
     cin >> mahasiswa.NIM;
     cout << "Masukkan nilai: ";
     cin >> mahasiswa.nilai;
     hashTable.tambahData(mahasiswa);
     cout << "Data berhasil ditambahkan" << endl;
     cout << endl;
 }
 else if (pilihan == 2)
 {
     string NIM;
     cout << "Masukkan NIM yang ingin dihapus: ";
     cin >> NIM;
     hashTable.hapusData(NIM);
     cout << "Data berhasil dihapus" << endl;
     cout << endl;
 }
 else if (pilihan == 3)
 {
     string NIM;
     cout << "Masukkan NIM yang ingin dicari: ";
     cin >> NIM;
     try
     {
         Mahasiswa                    mahasiswa                    =
hashTable.cariDataBerdasarkanNIM(NIM);
         cout << "NIM: " << mahasiswa.NIM << ", Nilai: " <<
mahasiswa.nilai << endl;
     }
```

```
     catch (const char *msg)
     { cerr << msg << endl; }
     cout << endl;
 }
 else if (pilihan == 4)
 {
     int nilaiMin, nilaiMax;
     cout << "Masukkan rentang nilai yang ingin dicari(misal :
80 90) : "; cin >> nilaiMin >> nilaiMax;
     vector<Mahasiswa>                  hasil                  =
hashTable.cariDataBerdasarkanRentangNilai(nilaiMin, nilaiMax);
     for (Mahasiswa mahasiswa : hasil)
     {
         cout << "NIM: " << mahasiswa.NIM << ", Nilai: " <<
mahasiswa.nilai << endl;
     }
     cout << endl;
 }
    } while (pilihan != 5);
    return 0;
}
```

Screenshoot program

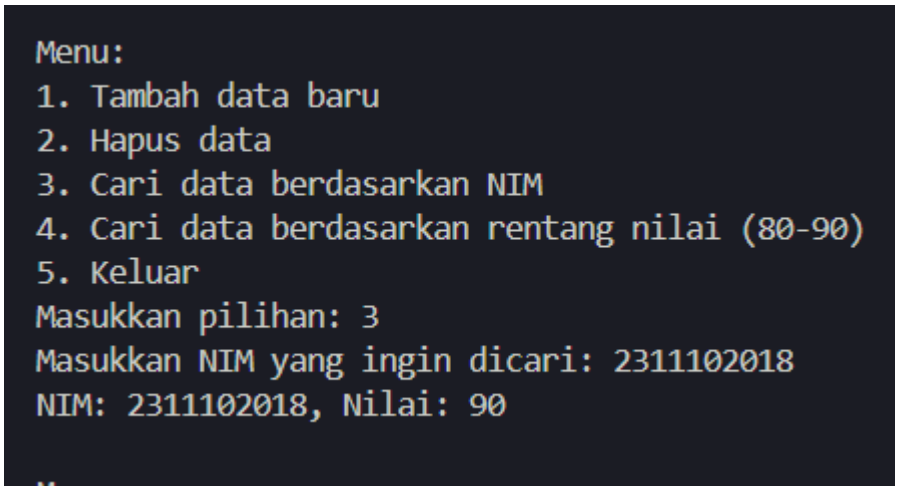
A. Tambah data

```
Menu:
1. Tambah data baru
2. Hapus data
3. Cari data berdasarkan NIM
4. Cari data berdasarkan rentang nilai (80-90)
5. Keluar
Masukkan pilihan: 1
Masukkan NIM: 2311102018
Masukkan nilai: 90
Data berhasil ditambahkan
```

B. Hapus Data

```
Menu:
1. Tambah data baru
2. Hapus data
3. Cari data berdasarkan NIM
4. Cari data berdasarkan rentang nilai (80-90)
5. Keluar
Masukkan pilihan: 2
Masukkan NIM yang ingin dihapus: 2311102018
Data berhasil dihapus
```

C. Temukan dengan Nim



Deskripsi program

Kode Program C++ di atas merupakan program untuk mengelola data mahasiswa menggunakan struktur HashTable. Program ini menyediakan menu interaktif untuk melakukan operasi CRUD (Create, Read, Update, Delete) pada data mahasiswa.

BAB IV

KESIMPULAN

Single dan double linked list adalah dua struktur data penting dalam pemrograman, yang digunakan untuk menyimpan dan mengorganisir kumpulan data secara terurut. Kesimpulannya, pemilihan antara single dan double linked list tergantung pada kebutuhan dan kinerja aplikasi yang diinginkan. Single linked list cenderung lebih hemat ruang memori dan cocok untuk aplikasi yang hanya memerlukan penambahan dan penghapusan di bagian depan atau belakang list. Sementara itu, double linked list lebih fleksibel dalam operasi penambahan, penghapusan, dan traversal maju-mundur, namun membutuhkan lebih banyak ruang memori.