

LAPORAN PRAKTIKUM

MODUL V HASH TABLE



Disusun oleh:
Muhammad Agha Zulfadhli
NIM: 2311102015

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 mampu menjelaskan definisi dan konsep dari Hash Code
2. Mahasiswa mampu menerapkan Hash Code kedalam pemrograman

BAB II

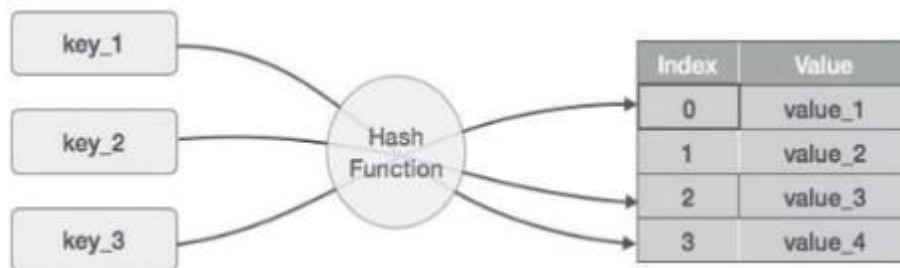
DASAR TEORI

a. **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.



b. 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.

c. 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:

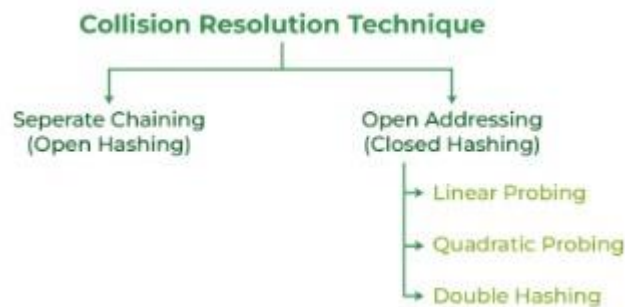
Memperbarui data dalam hash table dengan mencari data menggunakan fungsi hash, dan kemudian memperbarui data yang ditemukan.

5. Traversal:

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

d. 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 :



1. 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.

2. 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.

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;
    }
}

```

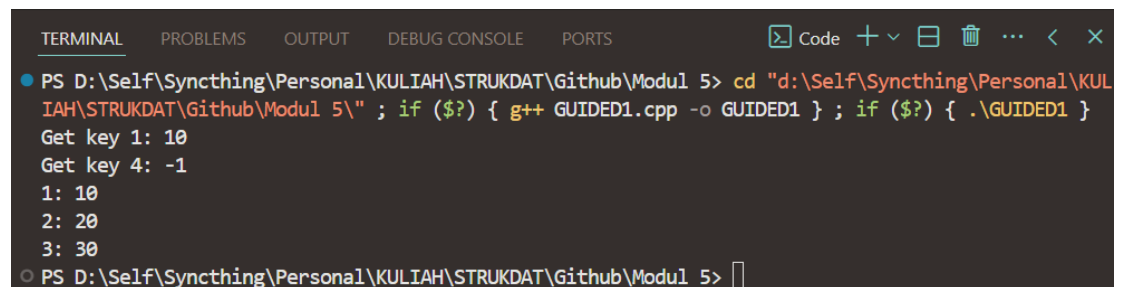


```

        }
        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();
}

```

Screenshoot program



```

TERMINAL  PROBLEMS  OUTPUT  DEBUG CONSOLE  PORTS
● PS D:\Self\Syncthing\Personal\KULIAH\STRUKDAT\Github\Modul 5> cd "d:\Self\Syncthing\Personal\KULIAH\STRUKDAT\Github\Modul 5\" ; if ($?) { g++ GUIDED1.cpp -o GUIDED1 } ; if ($?) { .\GUIDED1 }
Get key 1: 10
Get key 4: -1
1: 10
2: 20
3: 30
● PS D:\Self\Syncthing\Personal\KULIAH\STRUKDAT\Github\Modul 5> 

```

Deskripsi program

Program di atas adalah implementasi dari tabel hash sederhana menggunakan chaining (linked list) untuk menangani tabrakan (collision). Tabel hash ini terdiri dari sejumlah bucket yang ditentukan oleh konstanta `MAX_SIZE`, yang bernilai 10. Fungsi hash yang digunakan adalah fungsi modulus (`key % MAX_SIZE`), yang mengembalikan indeks bucket berdasarkan kunci yang diberikan.

Struktur data utama yang digunakan adalah struct `Node`, yang menyimpan pasangan kunci-nilai (`key-value`) dan pointer ke node berikutnya dalam linked list. Kelas `HashTable` mengelola array pointer ke node-node ini (`Node** table`), yang bertindak sebagai bucket dalam tabel hash.

Konstruktor `HashTable` menginisialisasi array dengan nilai `nullptr`. Destruktor `HashTable` mengiterasi seluruh bucket dan menghapus setiap node yang ada untuk mencegah kebocoran memori.

Metode `insert` menambahkan pasangan kunci-nilai ke dalam tabel hash. Jika kunci sudah ada, nilai yang terkait akan diperbarui. Jika tidak, node baru akan dibuat dan ditambahkan ke awal linked list di bucket yang sesuai.

Metode `get` mencari dan mengembalikan nilai berdasarkan kunci. Jika kunci tidak ditemukan, metode ini mengembalikan -1.

Metode `remove` menghapus node berdasarkan kunci dari tabel hash. Jika kunci ditemukan, node yang terkait akan dihapus dan pointer yang relevan akan diperbarui untuk menjaga integritas linked list.

Metode `traverse` mengiterasi seluruh tabel hash dan mencetak semua pasangan kunci-nilai yang tersimpan dalam tabel.

Di fungsi `main`, beberapa operasi dasar dilakukan pada tabel hash: memasukkan beberapa pasangan kunci-nilai, mencari nilai berdasarkan kunci, mencoba menghapus kunci yang tidak ada, dan menampilkan semua isi tabel hash melalui traversal. Hasil dari pencarian dan traversal dicetak ke konsol.

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");
}

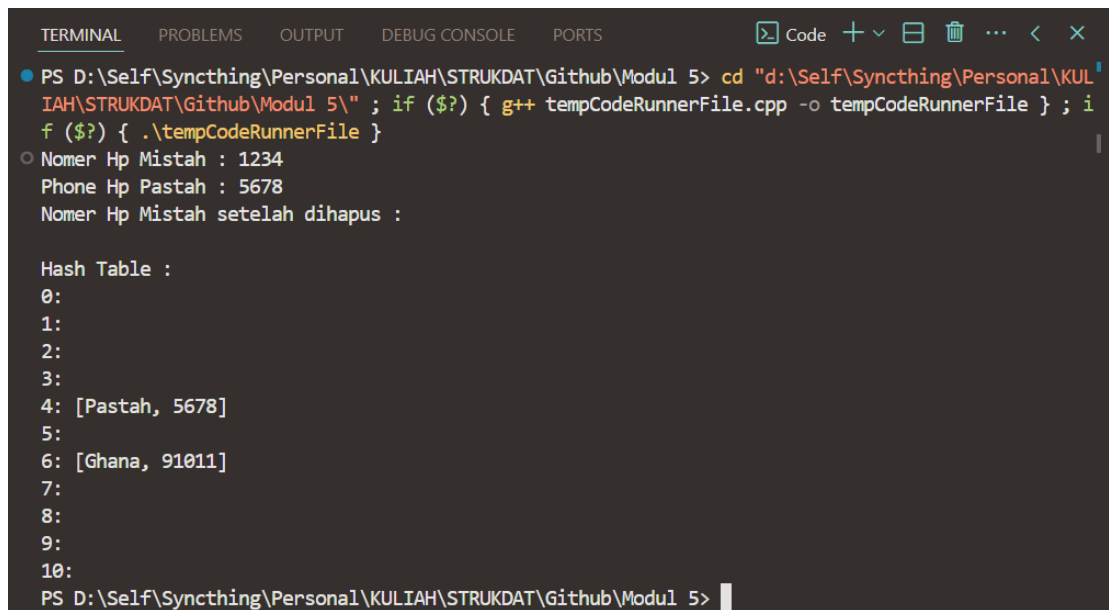
```

```

    cout << "Nomer Hp Mistah setelah dihapus : "
        << employee_map.searchByName("Mistah") << endl
        << endl;
    cout << "Hash Table : " << endl;
    employee_map.print();
    return 0;
}

```

Screenshoot program



```

TERMINAL  PROBLEMS  OUTPUT  DEBUG CONSOLE  PORTS
PS D:\Self\Syncthing\Personal\KULIAH\STRUKDAT\Github\Modul 5> cd "d:\Self\Syncthing\Personal\KULIAH\STRUKDAT\Github\Modul 5\" ; if ($?) { g++ tempCodeRunnerFile.cpp -o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }
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 D:\Self\Syncthing\Personal\KULIAH\STRUKDAT\Github\Modul 5>

```

Deskripsi program

Program di atas adalah implementasi tabel hash untuk menyimpan data nama dan nomor telepon dengan menggunakan metode chaining (linked list) untuk menangani tabrakan (collision). Kelas utama yang digunakan adalah `HashMap`, yang mengelola array vektor dari pointer ke objek `HashNode`. Setiap `HashNode` berisi dua atribut: `name` dan `phone_number`.

Fungsi hash `hashFunc` menghitung nilai hash dari kunci berupa string dengan menjumlahkan nilai ASCII dari setiap karakter dalam string dan

mengambil modulus dengan ukuran tabel hash (`TABLE_SIZE`), yang bernilai 11.

Metode `insert` menambahkan pasangan nama dan nomor telepon ke dalam tabel hash. Jika nama sudah ada, nomor telepon yang terkait akan diperbarui. Jika tidak, node baru akan dibuat dan ditambahkan ke vektor yang sesuai di dalam tabel hash.

Metode `remove` menghapus node berdasarkan nama dari tabel hash. Jika nama ditemukan, node yang terkait akan dihapus dari vektor di bucket yang sesuai.

Metode `searchByName` mencari dan mengembalikan nomor telepon berdasarkan nama. Jika nama tidak ditemukan, metode ini mengembalikan string kosong.

Metode `print` mengiterasi seluruh tabel hash dan mencetak semua pasangan nama dan nomor telepon yang tersimpan dalam tabel.

Pada fungsi `main`, beberapa operasi dasar dilakukan pada tabel hash:

- Menyisipkan beberapa pasangan nama dan nomor telepon ke dalam tabel hash.
- Mencari dan mencetak nomor telepon berdasarkan nama.
- Menghapus data berdasarkan nama.
- Mencetak isi tabel hash setelah beberapa operasi.

LATIHAN KELAS - UNGUIDED

Unguided

Source code

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

const int TABLE_SIZE = 12;

struct HashNode {
    string nim;
    double grade;
};

int hashFunc(const string &nim) {
    int hash_result = 0;
    for (char it : nim) {
        hash_result += it;
    }
    return hash_result % TABLE_SIZE;
}

void addMahasiswa(vector<HashNode> &list, const string &nim,
double grade) {
    HashNode add;
    add.nim = nim;
    add.grade = grade;
    list.push_back(add);
}

string searchByNIM(const vector<HashNode> &list, const string
&nim) {
    for (const auto &node : list) {
        if (node.nim == nim) {
            return "NIM: " + node.nim + "\nGrade: " +
to_string(node.grade);
        }
    }
    return "NIM: " + nim + " not found";
}

void searchByGrade(const vector<HashNode> mahasiswa[], double
grade) {
    bool found = false;
    cout << left << setw(20) << "NIM" << "NILAI" << endl;
```



```

        for (int i = 0; i < TABLE_SIZE; i++) {
            for (const auto &node : mahasiswa[i]) {
                if (node.grade == grade) {
                    cout << left << setw(20) << node.nim << fixed <<
setprecision(2) << node.grade << endl;
                    found = true;
                }
            }
        }
        if (!found) {
            cout << "KOSONG" << endl;
        }
    }

void deleteMahasiswa(vector<HashNode> &list, const string &nim)
{
    for (auto it = list.begin(); it != list.end(); ++it) {
        if (it->nim == nim) {
            list.erase(it);
            return;
        }
    }
}

string inputNIM() {
    string NIM;
    cout << "Masukkan NIM : ";
    cin >> NIM;
    return NIM;
}

double inputGrade() {
    double grade;
    cout << "Masukkan Nilai : ";
    cin >> grade;
    return grade;
}

int main() {
    int choice;
    string nim;
    double grade;

    vector<HashNode> mahasiswa[TABLE_SIZE];

    addMahasiswa(mahasiswa[hashFunc("2311102033")],
"2311102033", 99.99);
    addMahasiswa(mahasiswa[hashFunc("2311102034")],
"2311102034", 70.90);

    do {

```

```

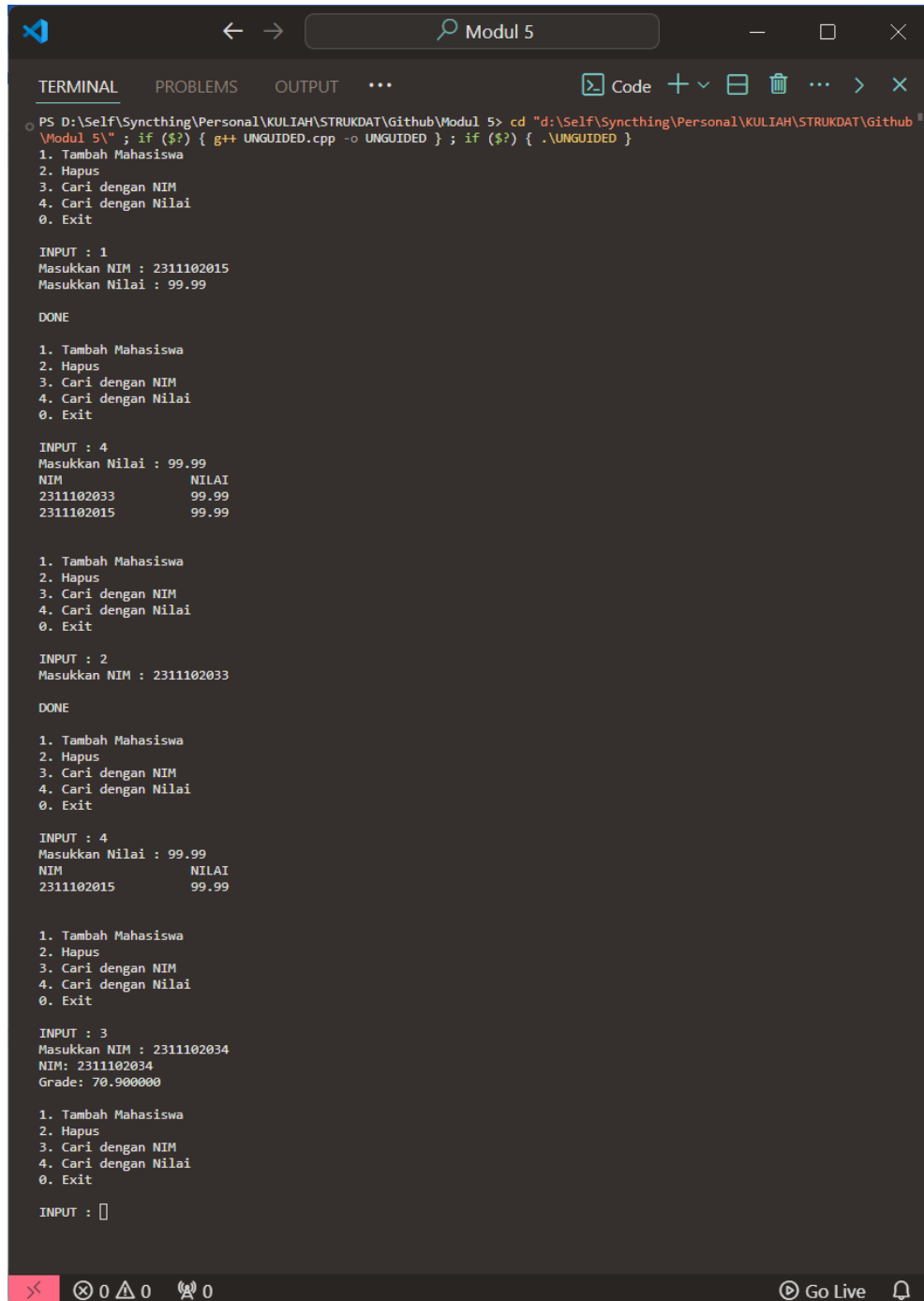
        cout << "1. Tambah Mahasiswa\n2. Hapus\n3. Cari dengan
NIM\n4. Cari dengan Nilai\n0. Exit\n\nINPUT : ";
        cin >> choice;

        switch (choice) {
            case 1:
                nim = inputNIM();
                grade = inputGrade();
                addMahasiswa(mahasiswa[hashFunc(nim)],      nim,
grade);

                cout << "\nDONE\n\n";
                break;
            case 2:
                nim = inputNIM();
                deleteMahasiswa(mahasiswa[hashFunc(nim)], nim);
                cout << "\nDONE\n\n";
                break;
            case 3:
                nim = inputNIM();
                cout <<  searchByNIM(mahasiswa[hashFunc(nim)],
nim) << endl << endl;
                break;
            case 4:
                grade = inputGrade();
                searchByGrade(mahasiswa, grade);
                cout << "\n\n";
                break;
            default:
                break;
        }
    } while (choice > 0 && choice < 5);
    return 0;
}

```

Screenshoot program



```
PS D:\Self\Syncthing\Personal\KULIAH\STRUKDAT\Github\Modul 5> cd "d:\Self\Syncthing\Personal\KULIAH\STRUKDAT\Github\Modul 5\" ; if ($?) { g++ UNGUIDED.cpp -o UNGUIDED } ; if ($?) { .\UNGUIDED }  
1. Tambah Mahasiswa  
2. Hapus  
3. Cari dengan NIM  
4. Cari dengan Nilai  
0. Exit  
  
INPUT : 1  
Masukkan NIM : 2311102015  
Masukkan Nilai : 99.99  
  
DONE  
  
1. Tambah Mahasiswa  
2. Hapus  
3. Cari dengan NIM  
4. Cari dengan Nilai  
0. Exit  
  
INPUT : 4  
Masukkan Nilai : 99.99  
NIM          NILAI  
2311102033    99.99  
2311102015    99.99  
  
1. Tambah Mahasiswa  
2. Hapus  
3. Cari dengan NIM  
4. Cari dengan Nilai  
0. Exit  
  
INPUT : 2  
Masukkan NIM : 2311102033  
  
DONE  
  
1. Tambah Mahasiswa  
2. Hapus  
3. Cari dengan NIM  
4. Cari dengan Nilai  
0. Exit  
  
INPUT : 4  
Masukkan Nilai : 99.99  
NIM          NILAI  
2311102015    99.99  
  
1. Tambah Mahasiswa  
2. Hapus  
3. Cari dengan NIM  
4. Cari dengan Nilai  
0. Exit  
  
INPUT : 3  
Masukkan NIM : 2311102034  
NIM: 2311102034  
Grade: 70.900000  
  
1. Tambah Mahasiswa  
2. Hapus  
3. Cari dengan NIM  
4. Cari dengan Nilai  
0. Exit  
  
INPUT : 
```

Deskripsi program

Program di atas adalah implementasi sederhana dari sebuah tabel hash yang digunakan untuk menyimpan data mahasiswa dengan atribut NIM (Nomor Induk Mahasiswa) dan nilai (grade). Program ini menyediakan beberapa operasi dasar seperti menambahkan, menghapus, mencari mahasiswa berdasarkan NIM, dan mencari mahasiswa berdasarkan nilai.

Struktur utama dalam program ini adalah ``HashNode``, yang terdiri dari dua atribut: ``nim`` (string) dan ``grade`` (double). Fungsi hash (``hashFunc``) digunakan untuk menghitung indeks hash dari NIM mahasiswa dengan menjumlahkan nilai ASCII dari setiap karakter dalam NIM dan mengambil modulus dengan ukuran tabel hash (``TABLE_SIZE``), yang ditetapkan menjadi 12.

Fungsi ``addMahasiswa`` digunakan untuk menambahkan data mahasiswa ke dalam tabel hash. Mahasiswa ditambahkan ke vektor yang sesuai berdasarkan hasil fungsi hash dari NIM. Fungsi ``searchByNIM`` mencari data mahasiswa berdasarkan NIM dan mengembalikan string berisi NIM dan nilai mahasiswa jika ditemukan, atau pesan bahwa NIM tidak ditemukan jika tidak ada dalam tabel hash.

Fungsi ``searchByGrade`` mencari dan mencetak semua mahasiswa dengan nilai yang sesuai dari seluruh tabel hash. Jika tidak ada mahasiswa dengan nilai tersebut, fungsi ini mencetak "KOSONG". Fungsi ``deleteMahasiswa`` menghapus data mahasiswa berdasarkan NIM dari tabel hash.

Fungsi ``inputNIM`` dan ``inputGrade`` digunakan untuk membaca input NIM dan nilai dari pengguna. Fungsi ``main`` adalah fungsi utama yang mengelola interaksi dengan pengguna melalui menu sederhana yang menyediakan pilihan untuk menambah mahasiswa, menghapus mahasiswa, mencari berdasarkan NIM, dan mencari berdasarkan nilai.

Pada awal program, dua data mahasiswa dengan NIM "2311102033" dengan nilai 99.99, dan "2311102034" dengan nilai 70.90, ditambahkan ke dalam tabel hash. Kemudian, dalam loop ``do-while``, program menampilkan menu dan memproses pilihan pengguna sesuai input yang diberikan, terus berjalan hingga pengguna memilih untuk keluar dari program dengan memasukkan nilai 0 sebagai pilihan.

Berikut adalah contoh alur penggunaan program:

1. Menambahkan mahasiswa dengan memasukkan NIM dan nilai.
2. Menghapus mahasiswa berdasarkan NIM.
3. Mencari dan menampilkan informasi mahasiswa berdasarkan NIM.
4. Mencari dan menampilkan informasi mahasiswa berdasarkan nilai.

Program ini memungkinkan pengguna untuk mengelola data mahasiswa dengan cara yang terorganisir menggunakan tabel hash untuk efisiensi pencarian dan manipulasi data.

BAB IV

KESIMPULAN

Dari uraian tersebut, dapat disimpulkan beberapa poin terkait Hash Table

1. Hash table adalah struktur data yang mengorganisir data ke dalam pasangan kunci-nilai menggunakan fungsi hash untuk mengubah kunci menjadi indeks array, memungkinkan pencarian data yang cepat ($O(1)$ dalam kasus terbaik).
2. Fungsi hash menghasilkan nilai unik dari setiap kunci, yang digunakan sebagai indeks untuk menyimpan data dalam array. Hasil dari fungsi hash disebut sebagai nilai hash.
3. Operasi dasar hash table meliputi penyisipan (insertion), penghapusan (deletion), pencarian (searching), pembaruan (updating), dan traversal data.
4. Collision terjadi ketika dua kunci menghasilkan nilai hash yang sama. Teknik penyelesaiannya termasuk chaining (menyimpan item dalam linked list di bucket yang sama) dan closed hashing (linear probing, quadratic probing, double hashing) untuk menemukan bucket kosong.
5. Hash table memungkinkan pencarian dan manipulasi data yang cepat. Namun, keterbatasan seperti collision memerlukan strategi penyelesaian yang efektif, seperti chaining atau probing, untuk mempertahankan performa optimal.

Dengan demikian, Hash Table dan fungsi-fungsi hash merupakan struktur data yang sangat berguna dalam menyimpan dan mengelola data dengan secara aman, dengan masing-masing jenis memiliki karakteristik dan kegunaan yang berbeda

DAFTAR PUSTAKA

- [1] Learning Management System ,MODUL 5 HASH TABLE.