LAPORAN STRUKTUR DATA ALGORITMA
OBJECT COMPOSITION (HASH TABLE, AFFORDANCE)

Disusun Oleh :

1. Ludvie Pradya Paramitha S (G1A024001)
2. Achmad Apanza (G1A024015)
3. Naifah Amanda (G1A024017)
4. Arya Pratama (G1A024061)
5. Abid Abdussalam (G1A024073)
6. Lantera Meunasah (G1A024105)

Dosen Pengampu :

1. Arie Vatresia, S.T., M.TI, Ph.D.
2. Kurnia Anggraini, S.T., M.T., Ph.D.

PROGRAM STUDI INFORMATIKA

FAKULTAS TEKNIK

UNIVERSITAS BENGKULU

2025

KATA PENGANTAR

Puji syukur kami panjatkan kepada Tuhan Yang Maha Esa atas berkat dan rahmat-Nya, sehingga makalah yang berjudul "Object Composition (Hash Table, Affordance)" ini dapat terselesaikan dengan baik. Makalah ini disusun sebagai salah satu tugas dalam mata kuliah Struktur Data dan Algoritma, dengan tujuan untuk mendalami konsep-konsep dasar dalam pemrograman, khususnya yang berkaitan dengan struktur data seperti hash table dan affordance dalam pengembangan perangkat lunak.

Kami mengucapkan terima kasih yang sebesar-besarnya kepada Ibu Arie Vatresia, S.T., M.TI, Ph.D., selaku dosen pembimbing, yang telah memberikan bimbingan dan arahan selama proses penyusunan makalah ini. Bantuan dan ilmu yang diberikan sangat berarti bagi penulis dalam memahami dan mengaplikasikan konsep-konsep yang dibahas dalam makalah ini. Semoga makalah ini dapat memberikan wawasan dan pemahaman yang bermanfaat bagi pembaca, serta menjadi referensi yang berguna dalam mengembangkan pengetahuan di bidang struktur data dan algoritma.

Penulis menyadari bahwa makalah ini masih jauh dari sempurna, oleh karena itu, kritik dan saran yang membangun sangat diharapkan untuk perbaikan di masa yang akan datang.

Bengkulu, 27 April 2025

Penulis

DAFTAR ISI

BAB I
PENDAHULUAN

1.1 Latar Belakang

Perkembangan pemrograman berorientasi objek telah memunculkan berbagai teknik untuk membangun sistem yang efisien dan mudah dikelola. Object composition menjadi teknik penting yang memungkinkan pembangunan objek kompleks dengan menggabungkan objek-objek sederhana, menawarkan alternatif yang lebih fleksibel dibandingkan pewarisan.

Dalam konteks struktur data, hash table merepresentasikan implementasi pemetaan key-value yang efisien dengan kompleksitas waktu pencarian mendekati O(1). Sementara itu, konsep affordance yang diadopsi dari bidang desain interaksi membantu menciptakan antarmuka yang intuitif dengan mengisyaratkan cara penggunaan objek. Makalah ini akan membahas pengertian, fungsi, dan penggunaan object composition terutama dalam konteks hash table dan affordance, serta bagaimana integrasi konsep-konsep ini dapat meningkatkan kualitas perangkat lunak.

1.2 Rumusan Masalah

1. Apa yang dimaksud dengan object composition dan bagaimana penerapannya dalam pengembangan perangkat lunak?
2. Bagaimana fungsi dan implementasi hash table dalam pengelolaan data?
3. Apa yang dimaksud dengan affordance dan bagaimana konsep ini diterapkan dalam desain antarmuka pengguna (user interface)?

BAB II
PEMBAHASAN

2.1 Object Composition

Sebuah objek adalah elemen dalam program yang berisi data sekaligus kode. Alasan utama penggunaan objek adalah karena data menjadi lebih berguna jika terdapat operasi yang dapat digunakan untuk mengubah atau memproses data tersebut. Sebagai contoh, kita menyimpan angka di dalam sistem komputer karena kita dapat melakukan berbagai operasi terhadap angka-angka tersebut. Dalam object composition, sebuah objek mengandung instance dari objek lain sebagai bagian dari strukturnya, bukan mewarisi sifat-sifatnya. Ini memungkinkan pengembang untuk membangun fungsionalitas yang lebih fleksibel dan modular, karena perilaku objek dapat diubah dengan mengganti komponen-komponennya tanpa perlu memodifikasi kode yang sudah ada.

2.1.1 Pengertian Object Composition

Object composition adalah konsep dalam pemrograman berorientasi objek (OOP) di mana sebuah objek besar dibentuk dengan menyusun beberapa objek kecil di dalamnya. Dengan kata lain, suatu objek dapat memiliki objek lain sebagai bagian dari dirinya. Hubungan ini sering disebut sebagai hubungan "HAS-A" (memiliki). Dalam object composition, objek bagian akan dibuat, dikelola, dan dihancurkan bersamaan dengan objek utamanya.

2.1.2 Fungsi Object Composition

1. Membuat Kode Lebih Modular

Object composition memungkinkan program dibangun dari bagian-bagian kecil (modul) yang masing-masing memiliki tugas spesifik. Dengan pendekatan ini, logika sistem tidak ditulis dalam satu tempat secara keseluruhan, melainkan dipisahkan ke dalam kelas-kelas terpisah seperti Mesin, Ban, dan Rem. Manfaat dari pendekatan ini adalah kode menjadi lebih mudah dibaca, dipahami, serta mempercepat proses identifikasi dan perbaikan jika terjadi kesalahan.

2. Meningkatkan Reusability (Dapat Digunakan Ulang)

Karena setiap komponen berdiri sendiri dan memiliki tanggung jawab spesifik, objek-objek tersebut dapat digunakan kembali di berbagai konteks tanpa perlu penulisan ulang. Sebagai contoh, objek Ban yang awalnya digunakan dalam kelas Mobil juga dapat digunakan pada Truk atau Sepeda Motor dengan sedikit atau tanpa modifikasi.

3. Lebih Fleksibel Dibandingkan Pewarisan (Inheritance)

Dalam pewarisan, satu kelas anak hanya dapat mewarisi dari satu kelas induk secara langsung, terutama pada bahasa pemrograman seperti C++ atau Java. Hal ini dapat membatasi fleksibilitas desain. Dengan menggunakan composition, sebuah objek dapat menggabungkan berbagai objek lain sekaligus, seperti Mesin, Ban, Rem, AC, dan Kursi. Hal ini memungkinkan pembuatan sistem yang lebih dinamis, tanpa ketergantungan struktur hierarkis yang kaku seperti pada inheritance.

4. Lebih Mudah dalam Pemeliharaan dan Pengembangan

Penggunaan object composition membuat sistem lebih mudah dipelihara dan dikembangkan. Jika ingin menambahkan fitur baru atau mengganti salah satu komponen, cukup dengan memodifikasi objek terkait tanpa harus mengubah keseluruhan sistem. Sebagai contoh, untuk mengganti mesin mobil dari mesin bensin ke mesin listrik, cukup membuat kelas baru seperti MesinListrik dan mengintegrasikannya ke dalam objek Mobil tanpa perlu mengubah seluruh struktur kode yang sudah ada.

5. Mendukung Prinsip SOLID dalam OOP

Object composition mendukung beberapa prinsip desain perangkat lunak yang baik, khususnya:

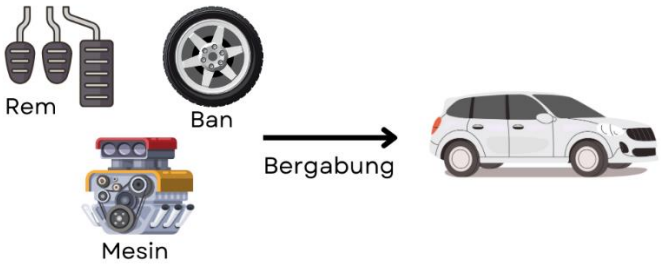
1. Single Responsibility Principle: Setiap kelas hanya memiliki satu tanggung jawab atau fokus tugas.
2. Dependency Inversion Principle: Sistem dibangun sedemikian rupa sehingga komponen tingkat tinggi tidak bergantung langsung pada komponen tingkat rendah, melainkan bergantung pada abstraksi.

Dengan menerapkan prinsip ini, struktur kode menjadi lebih kokoh, lebih fleksibel terhadap perubahan, dan lebih mudah dikembangkan untuk kebutuhan di masa depan.

2.1.3 Analogi dan Implementasi Object Composition

1. Analogi

Kita analogikan ke mobil. Mobil itu bukan cuma satu komponen, melainkan mobil itu terdiri dari beberapa komponen seperti ban, rem, mesin, dll. Masing-masing komponen memiliki fungsi dan karakter sendiri, lalu digabung menjadi mobil.



Gambar 1. Ilustrasi Object Composition

Sederhananya mobil gabungan dari mesin, ban dan rem. Tetapi bukan berarti mobil itu mesin, ban dan rem.

2. Implementasi dalam coding

Print Screen Source Code :

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

class Mesin
{
public:
    void nyalakan()
    {
        cout << "Mesin dinyalakan." << endl;
    }
    void matikan()
    {
        cout << "Mesin dimatikan." << endl;
    }
};

class Ban
{
public:
    void berputar()
    {
        cout << "Ban berputar." << endl;
    }
};

class Rem
{
public:
    void gunakan()
    {
        cout << "Rem digunakan." << endl;
    }
};
```

Gambar 2. Input Code

```cpp
class Mobil
{
private:
    Mesin mesin;
    Ban ban;
    Rem rem;

public:
    void jalan()
    {
        mesin.nyalakan();
        ban.berputar();
        cout << "Mobil berjalan." << endl;
    }

    void berhenti()
    {
        rem.gunakan();
        mesin.matikan();
        cout << "Mobil berhenti." << endl;
    }
};

int main()
{
    Mobil mobilAbid;
    mobilAbid.jalan();
    mobilAbid.berhenti();

    return 0;
}
```

Gambar 3. Input code

Penjelasan Source Code :

Kode ini membuat simulasi sederhana tentang bagaimana sebuah mobil bekerja dengan menggunakan konsep Object Composition dalam C++. Tiga kelas utama dibuat terlebih dahulu, yaitu Mesin, Ban, dan Rem, masing-masing berisi fungsi sederhana: Mesin memiliki fungsi nyalakan dan matikan, Ban memiliki fungsi berputar, dan Rem memiliki fungsi gunakan. Kemudian, kelas Mobil didefinisikan dengan memiliki tiga atribut berupa objek Mesin, Ban, dan Rem. Di dalam Mobil, terdapat dua metode utama, yaitu jalan dan berhenti. Saat fungsi jalan dipanggil, mobil akan menyalakan mesin, membuat ban berputar, lalu menampilkan pesan bahwa mobil berjalan. Sebaliknya, saat fungsi berhenti dipanggil, mobil akan menggunakan rem, mematikan mesin, dan menampilkan pesan bahwa mobil berhenti. Pada fungsi main(), dibuat sebuah objek mobilAbid dari kelas Mobil, lalu fungsi jalan dan berhenti dipanggil untuk mensimulasikan mobil tersebut berjalan dan berhenti.

Output :

```
Mesin dinyalakan.
Ban berputar.
Mobil berjalan.
Rem digunakan.
Mesin dimatikan.
Mobil berhenti.
```

Gambar 4. Output Code

Penjelasan :

Program ini membuat simulasi sederhana tentang bagaimana sebuah mobil bekerja menggunakan konsep Object Composition dalam C++. Tiga kelas utama, yaitu Mesin, Ban, dan Rem, masing-masing memiliki fungsi sederhana untuk menyalakan mesin, memutar ban, dan menggunakan rem. Kelas Mobil kemudian menggabungkan ketiga objek tersebut sebagai atribut, dan memiliki fungsi jalan untuk menyalakan mesin, memutar ban, serta memberi tahu bahwa mobil berjalan, serta fungsi berhenti untuk menggunakan rem, mematikan mesin, dan memberi tahu bahwa mobil berhenti. Pada fungsi main(), objek mobilAbid dibuat, lalu dipanggil fungsi jalan dan berhenti. Saat dijalankan, program akan mencetak urutan output: "Mesin dinyalakan.", "Ban berputar.", "Mobil berjalan.", "Rem digunakan.", "Mesin dimatikan.", dan "Mobil berhenti.", yang menunjukkan simulasi mobil menyala, berjalan, menggunakan rem, mematikan mesin, dan akhirnya berhenti.

2.2 Hash Table

Hashing adalah teknik yang memetakan sekumpulan data yang besar ke dalam sekumpulan data yang lebih kecil menggunakan sebuah fungsi hash. Proses ini bersifat irreversibel, artinya kita tidak dapat menemukan nilai asli dari sebuah key hanya dari nilai hash-nya. Hal ini terjadi karena kita mencoba memetakan data dari ruang yang besar ke ruang yang lebih kecil, sehingga memungkinkan terjadinya *collision* atau benturan, yaitu dua nilai berbeda yang menghasilkan hash yang sama

2.2.1 Pengertian Hash Table

Hash Table adalah struktur data yang sangat efisien untuk menyimpan dan mengambil data berdasarkan sebuah kunci atau key. Konsep dasarnya adalah mengasosiasikan setiap nilai atau value dengan sebuah kunci unik, lalu menggunakan fungsi hash untuk mengubah kunci tersebut menjadi sebuah indeks numerik. Indeks ini digunakan untuk menentukan posisi penyimpanan data dalam sebuah array.

2.2.2 Fungsi Hash Table

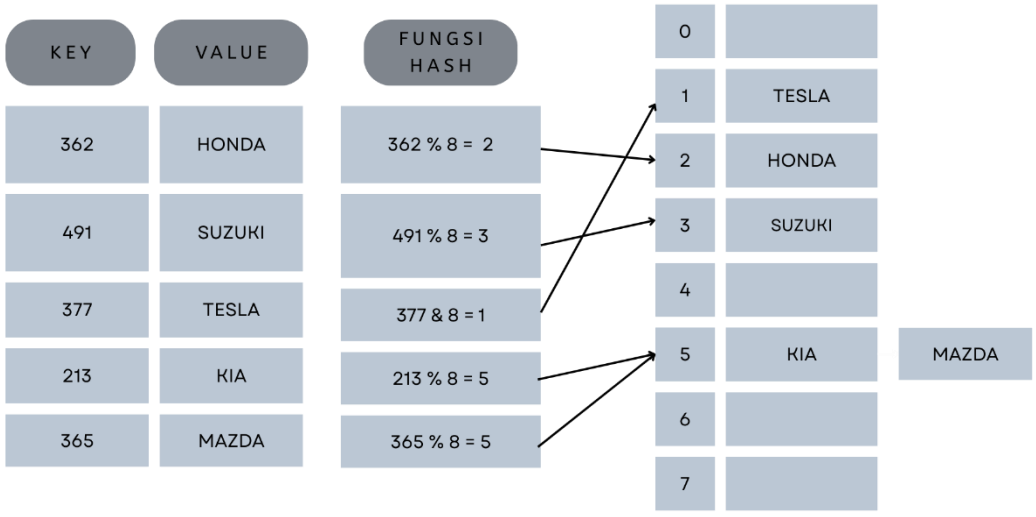
Fungsi utama menggunakan hash table adalah untuk mempercepat proses penyimpanan dan pencarian data berdasarkan kunci unik secara efisien dan dalam waktu konstan (O(1)) pada rata-rata kasus, tanpa bergantung pada ukuran data.

Hash table menggunakan fungsi hash untuk mengonversi kunci menjadi indeks dalam array, sehingga data dapat langsung diakses tanpa perlu pencarian berurutan. Dengan demikian, hash table sangat berguna dalam aplikasi yang membutuhkan akses data cepat seperti basis data, kamus, cache, dan struktur data lain seperti set atau map. Selain itu, hash table juga membantu menghemat memori dengan memetakan ruang kunci yang besar ke indeks array yang lebih kecil, sehingga proses pencarian, penyisipan, dan penghapusan data menjadi lebih efisien

2.2.3 Analogi dan Implementasi Hash Table

1. Analogi

Bayangkan kamu punya sebuah garasi besar dengan 8 rak bernomor 0 sampai 7. Setiap kali ada mobil baru datang, kamu menghitung nilai dari setiap huruf di nama mobil itu pakai kode ASCII, lalu jumlahnya di modulus 8 untuk menentukan rak mana yang akan ditempati mobil tersebut. Jadi, nama mobil otomatis menentukan raknya, bukan kamu yang pilih manual. Kalau kamu mau mencari atau menghapus mobil, kamu cukup menghitung lagi nama mobilnya dengan cara yang sama, lalu pergi ke rak yang tepat dan mencari mobilnya di sana satu per satu. Rak bisa berisi lebih dari satu mobil, dan semua mobil di rak diatur berurutan seperti antrean.



Gambar. 5 Ilustrasi Hash Table

2. Implementasi dalam coding

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

const int TABLE_SIZE = 8;

struct Mobil
{
    string nama;
    Mobil *next;
};

class HashTable
{
private:
    Mobil *table[TABLE_SIZE];

    int hashFunction(const string &key)
    {
        int hash = 0;
        for (char c : key)
        {
            hash += c;
        }
        return hash % TABLE_SIZE;
    }

public:
    HashTable()
    {
        for (int i = 0; i < TABLE_SIZE; ++i)
        {
            table[i] = nullptr;
        }
    }

    void tambah(const string &nama)
    {
        int index = hashFunction(nama);
        Mobil *newMobil = new Mobil{nama, nullptr};

        if (table[index] == nullptr)
        {
            table[index] = newMobil;
        }
        else
```

Gambar 6. Input Code

```cpp
        {
            newMobil->next = table[index];
            table[index] = newMobil;
        }
    }

    void tampilkan()
    {
        cout << "Isi Hash Table Mobil:" << endl;
        for (int i = 0; i < TABLE_SIZE; ++i)
        {
            if (table[i] != nullptr)
            {
                cout << "Rak " << i << ":" << endl;
                Mobil *current = table[i];
                while (current != nullptr)
                {
                    cout << " - " << current->nama << endl;
                    current = current->next;
                }
            }
        }
    }

    void cari(const string &nama)
    {
        int index = hashFunction(nama);
        Mobil *current = table[index];
        while (current != nullptr)
        {
            if (current->nama == nama)
            {
                cout << "Mobil " << nama << " ada." << endl;
                return;
            }
            current = current->next;
        }
        cout << "Mobil " << nama << " tidak ada." << endl;
    }
};
```

Gambar 7. Input Code

```
int main()
{
    HashTable mobil;

    mobil.tambah("HONDA");
    mobil.tambah("SUZUKI");
    mobil.tambah("TESLA");
    mobil.tambah("KIA");
    mobil.tambah("MAZDA");

    mobil.tampilkan();

    cout << endl;

    mobil.cari("TESLA");
    mobil.cari("KIA");
    mobil.cari("HONDA");
    mobil.cari("BMW");
}
```

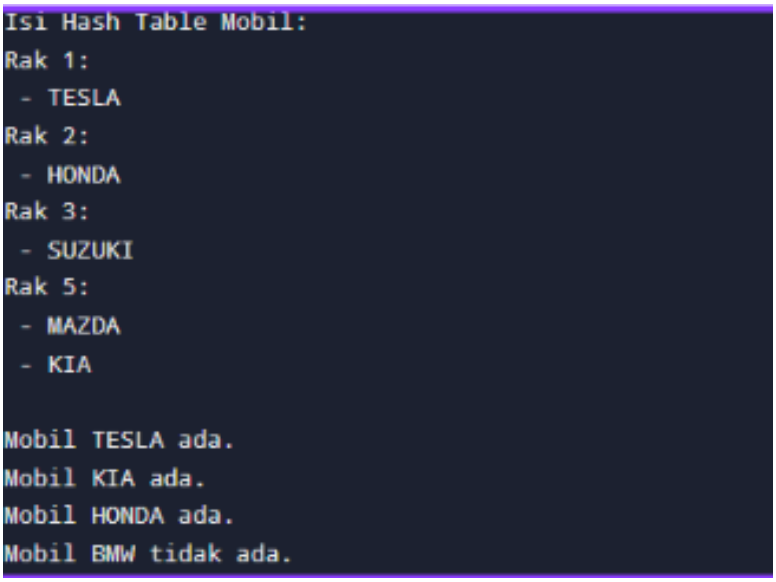
Gambar 8. Input Code

Penjelasan :

Kode C++ ini mengimplementasikan hash table sederhana untuk menyimpan data mobil. Struktur data Mobil memiliki atribut nama (nama mobil) dan pointer next untuk menangani collision menggunakan linked list. Kelas HashTable memiliki array table yang menyimpan pointer ke objek Mobil, fungsi hashFunction untuk menghasilkan indeks hash dari nama mobil, konstruktor untuk menginisialisasi semua elemen table dengan nullptr, fungsi tambah untuk menambahkan mobil ke hash table, fungsi tampilkan untuk menampilkan isi hash table, dan fungsi cari untuk mencari mobil berdasarkan nama.

Fungsi hashFunction menghitung indeks hash berdasarkan jumlah kode ASCII dari setiap karakter dalam nama mobil, lalu mengambil modulus dengan ukuran tabel. Fungsi tambah membuat objek Mobil baru, menghitung indeks hash dari namanya, dan menambahkannya ke table pada indeks tersebut, menangani collision dengan menambahkan objek baru ke awal linked list. Fungsi tampilkan mengiterasi setiap elemen dalam table dan mencetak semua nama mobil yang ada dalam linked list pada setiap indeks. Fungsi cari mencari nama mobil dalam hash table dan menampilkan pesan apakah mobil tersebut ditemukan atau tidak.

Output :



Gambar 9. Output Code

Penjelasan :

Bagian pertama menampilkan isi hash table. Setiap "Rak" (indeks) menunjukkan mobil apa saja yang ada di indeks tersebut. Misalnya:

Rak 1: TESLA

Rak 2: HONDA

Rak 3: SUZUKI

Rak 5: MAZDA dan KIA

Ini berarti mobil "TESLA" berada di indeks 1, "HONDA" di indeks 2, "SUZUKI" di indeks 3, dan "MAZDA" serta "KIA" di indeks 5 (kemungkinan terjadi collision di indeks ini).

Bagian kedua menunjukkan hasil pencarian beberapa mobil:

Mobil TESLA ada.

Mobil KIA ada.

Mobil HONDA ada.

Mobil BMW tidak ada.

Ini menunjukkan bahwa mobil "TESLA", "KIA", dan "HONDA" berhasil ditemukan dalam hash table, sedangkan mobil "BMW" tidak ditemukan.

2.3 Affordance Analysis

Human Computer Interaction (HCI), oleh Donald Norman dalam bukunya yang inovatif The Psychology of Everyday Things (1988). Norman mendefinisikan affordance sebagai sifat yang dirasakan atau aktual dari benda itu, terutama sifat-sifat mendasar yang menentukan bagaimana benda itu bisa digunakan.

2.3.1 Pengertian Affordance Analysis

Affordance Analysis adalah suatu proses yang bekerja untuk menganalisis apa saja tindakan yang dapat dilakukan terhadap sebuah objek atau sebuah sistem, berdasarkan petunjuk atau bentuknya. dalam desain, affordance adalah isyarat alami yang menunjukkan seperti bagaimana sesuatu seharusnya digunakan tetapi tanpa harus dijelaskan. affordance analysis memastikan bahwa pengguna langsung tahu bagaimana cara menggunakan objek itu tanpa membuat bingung.

2.3.2 Fungsi Affordance Analysis

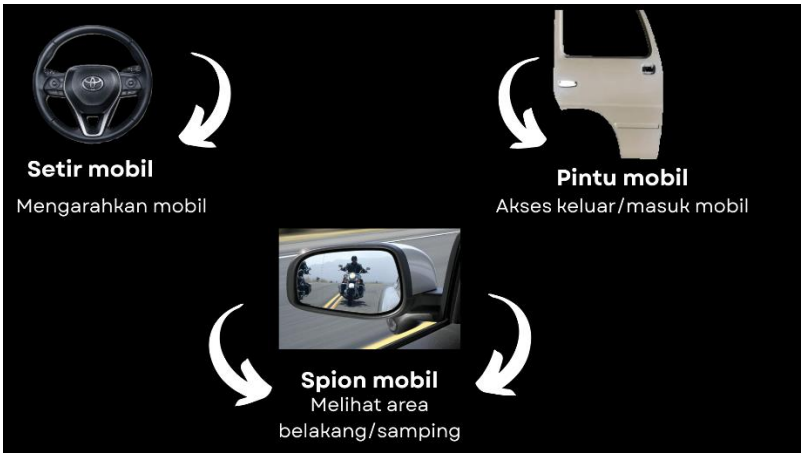
1. Meningkatkan kegunaan (usability). Membantu memastikan bahwa pengguna bisa langsung paham cara menggunakan produk atau sistem tanpa perlu instruksi rumit.
2. Mengidentifikasi potensi masalah interaksi. Menemukan bagian dari desain yang mungkin membingungkan, bikin frustrasi, atau bahkan berbahaya.
3. Membantu merancang produk yang intuitif. Agar bentuk, tampilan, atau perilaku suatu objek langsung "memberi tahu" pengguna apa yang harus dilakukan.
4. Mengoptimalkan pengalaman pengguna (user experience). Dengan memahami affordance, produk bisa dibuat lebih natural dan menyenangkan untuk digunakan.
5. Mendukung inovasi desain. Dengan tahu bagaimana affordance bekerja, desainer bisa menciptakan solusi kreatif baru yang lebih efektif.

2.3.3 Analogi dan Implementasi Affordance

1. Analogi

Bayangkan kamu sedang membuat katalog bagian-bagian mobil. Dalam kode ini, setiap Part adalah benda di mobil (seperti roda, rem, atau setir), yang disimpan rapi dalam sebuah daftar (vector). Kemudian, layaknya seorang pemandu yang memperkenalkan satu per satu benda kepada pengunjung, program ini

membaca semua bagian dan menjelaskan apa kegunaan (affordance) masing-masing, sehingga orang yang melihat langsung paham apa fungsi setiap komponen mobil tersebut.



Gambar 10. Ilustrasi Affordance

2. Implementasi dalam coding

Print Screen Source Code :

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

struct Part
{
    string name;
    string affordance;
};

int main()
{
    vector<Part> Bagianmobil = {
        {"Roda", "Bergerak maju/mundur/belok"},
        {"Rem", "Menghentikan mobil"},
        {"Setir", "Mengarahkan mobil"},
        {"Mesin", "Menggerakkan mobil"},
        {"Lampu", "Menerangi jalan / Memberi sinyal"},
        {"Spion", "Melihat area belakang/samping"},
        {"Klakson", "Memberi peringatan suara"},
        {"Pintu", "Akses keluar/masuk mobil"},
        {"Kaca", "Memberi pandangan keluar"}};

    cout << "Affordance Analysis Bagian Mobil:" << endl;
    for (const auto &part : Bagianmobil)
    {
        cout << "- " << part.name << ": " << part.affordance << endl;
    }

    return 0;
}
```

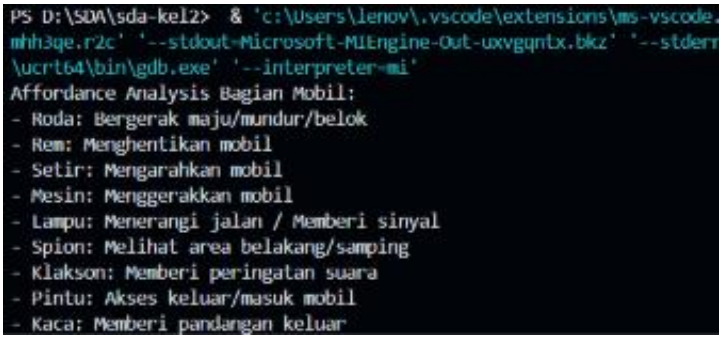
Gambar 11. Input Code

Penjelasan :

Input kode ini melakukan analisis affordance sederhana terhadap bagian-bagian mobil, struktur part mendefinisikan setiap bagian mobil dengan atribut name (nama bagian) dan affordance (fungsi atau kegunaan bagian tersebut). Di dalam fungsi main(), sebuah vector bernama Bagianmobil diinisialisasi dengan

beberapa objek Part, yang masing-masing mewakili bagian mobil seperti roda, rem, setir, mesin, lampu, spion, klakson, pintu, dan kaca, beserta affordance atau fungsi masing-masing. Kemudian, kode mencetak judul "Affordance Analysis Bagian Mobil:" dan melakukan iterasi melalui vector Bagianmobil. Untuk setiap bagian mobil, kode mencetak nama bagian dan affordance-nya, dipisahkan oleh tanda ":" dan diawali dengan tanda "-". Hasilnya adalah daftar bagian mobil dan fungsi-fungsi yang ditawarkannya (affordance), yang memberikan gambaran tentang bagaimana setiap bagian berkontribusi terhadap pengoperasian dan pengalaman berkendara mobil.

Output :



Gambar. 12 Output

Penjelasan :

Output tersebut menunjukkan hasil analisis affordance dari berbagai bagian mobil. Setiap baris dimulai dengan nama bagian mobil, diikuti dengan deskripsi affordance atau fungsi dari bagian tersebut. Contohnya, "Roda: Bergerak maju/mundur/belok" berarti roda memiliki fungsi untuk memungkinkan mobil bergerak maju, mundur, dan berbelok. Demikian pula, "Rem: Menghentikan mobil" menunjukkan bahwa rem berfungsi untuk menghentikan mobil. Daftar ini mencakup bagian-bagian seperti setir, mesin, lampu, spion, klakson, pintu, dan kaca, beserta fungsi atau kegunaan masing-masing dalam konteks pengoperasian mobil.

2.4 Gabungan antara Object Composition, Hash Table, dan Affordance

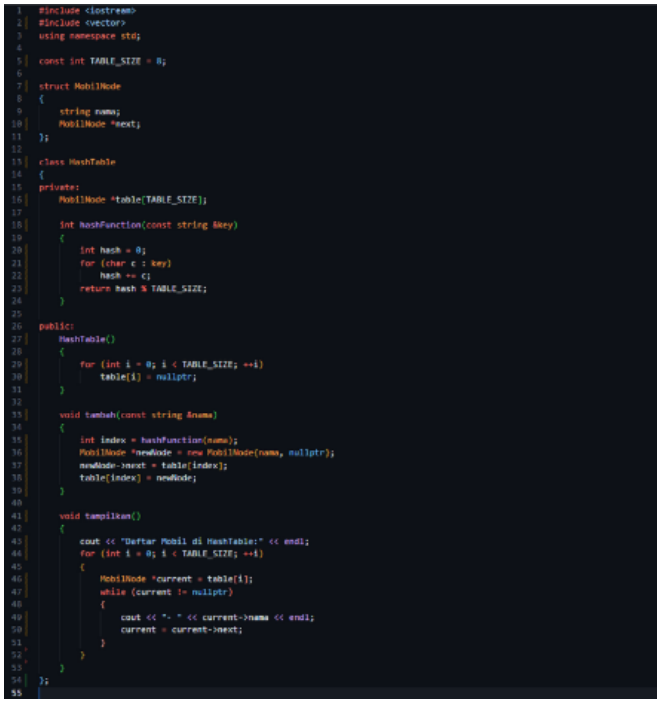
Object Composition adalah teknik membangun objek dari beberapa objek kecil tanpa pewarisan (*inheritance*). Hash Table adalah struktur data untuk menyimpan *key-value pair* dengan akses cepat. Affordance dalam konteks ini

berarti membuat antarmuka objek mudah dipahami dan digunakan. Ketiga konsep ini bisa digabung untuk membangun sistem yang modular, cepat, dan intuitif. Misalnya, sebuah objek User bisa dibentuk dari objek PersonalInfo, Settings, dan Preferences (composition), lalu semua User disimpan dalam Hash Table berdasarkan ID. Fungsi seperti findUser(id) menciptakan affordance karena memberi petunjuk alami bagaimana objek digunakan.

2.4.1 Implementasi Code dan penjelasan

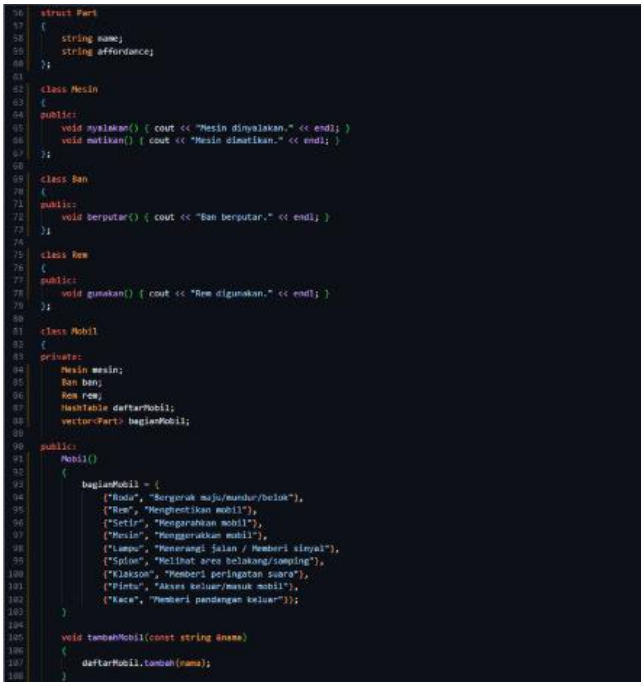
Program ini membuat sebuah objek Mobil yang di dalamnya terdiri dari beberapa komponen seperti Mesin, Ban, dan Rem (object composition), serta dilengkapi dengan Hash Table untuk menyimpan daftar nama mobil dan Affordance Analysis untuk menganalisis fungsi bagian-bagian mobil. Hash Table digunakan untuk memasukkan dan menampilkan nama mobil berdasarkan perhitungan sederhana dari karakter nama. Sementara affordance analysis berupa daftar bagian mobil (seperti Roda, Rem, Setir) beserta fungsinya (seperti bergerak, menghentikan, mengarahkan). Program ini berjalan dengan cara menambahkan mobil ke tabel, membuat mobil berjalan dan berhenti, lalu menampilkan fungsi dari tiap bagian mobil secara terstruktur dan sederhana.

Print Screen Source Code :



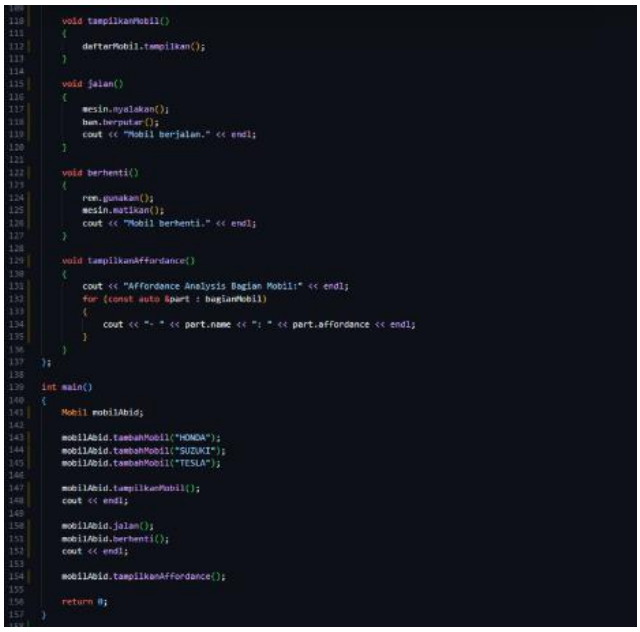
```cpp
#include <iostream>
#include <vector>
using namespace std;

const int TABLE_SIZE = 8;

struct MobilNode
{
    string nama;
    MobilNode *next;
};

class HashTable
{
private:
    MobilNode *table[TABLE_SIZE];

    int hashFunction(const string &key)
    {
        int hash = 0;
        for (char c : key)
            hash += c;
        return hash % TABLE_SIZE;
    }

public:
    HashTable()
    {
        for (int i = 0; i < TABLE_SIZE; ++i)
            table[i] = nullptr;
    }

    void tambah(const string &nama)
    {
        int index = hashFunction(nama);
        MobilNode *newNode = new MobilNode(nama, nullptr);
        newNode->next = table[index];
        table[index] = newNode;
    }

    void tampilkan()
    {
        cout << "Daftar Mobil di HashTable:" << endl;
        for (int i = 0; i < TABLE_SIZE; ++i)
        {
            MobilNode *current = table[i];
            while (current != nullptr)
            {
                cout << "- " << current->nama << endl;
                current = current->next;
            }
        }
    }
};
```

Gambar 13. Input Code



```
struct Part
{
    string name;
    string affordance;
};

class Mesin
{
public:
    void nyalakan() { cout << "Mesin dinyalakan." << endl; }
    void matikan() { cout << "Mesin dimatikan." << endl; }
};

class Ban
{
public:
    void berputar() { cout << "Ban berputar." << endl; }
};

class Rem
{
public:
    void gunakan() { cout << "Rem digunakan." << endl; }
};

class Mobil
{
private:
    Mesin mesin;
    Ban ban;
    Rem rem;
    HashTable daftarMobil;
    vector<Part> bagianMobil;

public:
    Mobil()
    {
        bagianMobil = {
            {"Roda", "Bergerak maju/mundur/belok"},
            {"Rem", "Menghentikan mobil"},
            {"Setir", "Mengarahkan mobil"},
            {"Mesin", "Menggerakkan mobil"},
            {"Lampu", "Menerangi jalan / Memberi sinyal"},
            {"Spion", "Melihat area belakang/samping"},
            {"Klakson", "Memberi peringatan suara"},
            {"Pintu", "Akses keluar/masuk mobil"},
            {"Kaca", "Memberi pandangan keluar"}};
    }

    void tambahMobil(const string &nama)
    {
        daftarMobil.tambah(nama);
    }
```

Gambar 14. Input Code



```
    void tampilkanMobil()
    {
        daftarMobil.tampilkan();
    }

    void jalan()
    {
        mesin.nyalakan();
        ban.berputar();
        cout << "Mobil berjalan." << endl;
    }

    void berhenti()
    {
        rem.gunakan();
        mesin.matikan();
        cout << "Mobil berhenti." << endl;
    }

    void tampilkanAffordance()
    {
        cout << "Affordance Analysis Bagian Mobil:" << endl;
        for (const auto &part : bagianMobil)
        {
            cout << "- " << part.name << ": " << part.affordance << endl;
        }
    }
};

int main()
{
    Mobil mobilAbid;

    mobilAbid.tambahMobil("HONDA");
    mobilAbid.tambahMobil("SUZUKI");
    mobilAbid.tambahMobil("TESLA");

    mobilAbid.tampilkanMobil();
    cout << endl;

    mobilAbid.jalan();
    mobilAbid.berhenti();
    cout << endl;

    mobilAbid.tampilkanAffordance();

    return 0;
}
```

Gambar 15. Input Code

Penjelasan Source Code :

Kode ini terdiri dari dua bagian utama yang saling melengkapi, yaitu struktur hash table untuk menyimpan data mobil dan simulasi bagian-bagian mobil seperti mesin, ban, dan rem. Pada bagian kiri gambar (file pertama), terdapat implementasi hash table menggunakan array statik dan linked list untuk menangani kemungkinan collision. Struktur MobilNode menyimpan nama mobil dan pointer

ke node berikutnya. Kelas HashTable memiliki fungsi untuk meng-hash nama mobil dan menyimpannya dalam array. Fungsi tambah() digunakan untuk menambahkan nama mobil ke dalam hash table, sedangkan tampilkan() mencetak semua mobil yang telah disimpan di dalam hash table.

Pada bagian kanan gambar (file kedua), terdapat implementasi simulasi sistem bagian-bagian mobil. Terdapat struktur Part untuk menyimpan nama dan affordance dari tiap bagian mobil. Kelas Mesin, Ban, dan Rem mewakili komponen-komponen mobil yang memiliki fungsi tertentu seperti nyalakan(), matikan(), berputar(), dan gunakan(). Kelas Mobil menggabungkan semua komponen ini serta menyimpan daftar mobil dalam bentuk hash table dan vektor bagian-bagian mobil. Fungsi tambahMobil() menambahkan nama mobil ke dalam hash table. Fungsi jalan() dan berhenti() menjalankan fungsi-fungsi pada komponen seperti mesin, ban, dan rem. Terakhir, fungsi tampilkanAffordance() mencetak affordance dari semua bagian mobil yang telah ditentukan.

Dalam fungsi main(), beberapa mobil ditambahkan ke dalam daftar, lalu daftar mobil ditampilkan, mobil dijalankan, dihentikan, dan affordance dari tiap bagian mobil dicetak. Kode ini merupakan contoh implementasi konsep OOP (Object-Oriented Programming) di C++, penggunaan struktur data hash table, serta bagaimana menggabungkan simulasi logika mekanikal dengan manajemen data.

Output :

```
Daftar Mobil di HashTable:
- TESLA
- HONDA
- SUZUKI

Mesin dinyalakan.
Ban berputar.
Mobil berjalan.
Rem digunakan.
Mesin dimatikan.
Mobil berhenti.

Affordance Analysis Bagian Mobil:
- Roda: Bergerak maju/mundur/belok
- Rem: Menghentikan mobil
- Setir: Mengarahkan mobil
- Mesin: Menggerakkan mobil
- Lampu: Menerangi jalan / Memberi sinyal
- Spion: Melihat area belakang/samping
- Klakson: Memberi peringatan suara
- Pintu: Akses keluar/masuk mobil
- Kaca: Memberi pandangan keluar
```

Gambar 16. Output Kode Gabungan

Penjelasan :

Output ini menunjukkan hasil eksekusi dari program C++ yang telah dijelaskan sebelumnya. Pertama, program mencetak daftar mobil yang berhasil disimpan di dalam struktur hash table, yaitu "TESLA", "HONDA", dan "SUZUKI". Selanjutnya, program mensimulasikan proses mobil berjalan, dimulai dari mesin yang dinyalakan, ban yang berputar, hingga mobil bergerak. Setelah itu, rem digunakan untuk menghentikan mobil, mesin dimatikan, dan akhirnya mobil berhenti. Di bagian akhir, program menampilkan analisis affordance dari berbagai bagian mobil, menjelaskan fungsi dari komponen-komponen seperti roda, rem, setir, mesin, lampu, spion, klakson, pintu, dan kaca. Analisis ini menggambarkan kemampuan atau peran setiap bagian dalam mendukung fungsionalitas mobil secara keseluruhan.

BAB III
KESIMPULAN

3.1 Kesimpulan

Berdasarkan pembahasan yang telah disampaikan, dapat disimpulkan bahwa *object composition*, *hash table*, dan *affordance* merupakan tiga konsep penting dalam pemrograman berorientasi objek yang dapat saling melengkapi dalam membangun sistem perangkat lunak yang modular, efisien, dan intuitif. Melalui pendekatan *object composition*, objek kompleks seperti mobil dapat dibentuk dari gabungan objek-objek sederhana seperti mesin, ban, dan rem, yang masing-masing memiliki fungsi spesifik dan independen. Penggunaan *hash table* memungkinkan pengelolaan data mobil secara cepat dan efisien, terutama dalam proses pencarian dan penyimpanan data berdasarkan kunci unik. Sementara itu, *affordance analysis* membantu dalam memahami dan merancang bagian-bagian sistem yang mudah digunakan dan dipahami pengguna, dengan menekankan fungsi-fungsi alami dari setiap komponen.

Implementasi ketiga konsep ini dalam kode program C++ yang telah dibuat menunjukkan bagaimana sistem yang fleksibel, dapat dikembangkan, serta user-friendly dapat dibangun dengan menggabungkan prinsip-prinsip desain perangkat lunak yang baik. Dengan simulasi mobil berjalan dan berhenti, serta analisis affordance dari tiap komponennya, program ini tidak hanya merepresentasikan teori, tetapi juga memperlihatkan aplikasinya secara praktis. Hal ini membuktikan bahwa pendekatan komposisi objek yang dipadukan dengan struktur data efisien dan desain interaktif mampu menghasilkan sistem yang optimal dalam pengembangan perangkat lunak modern

DAFTAR PUSTAKA

Bower, M. (2008). Affordance analysis: Matching learning tasks with learning technologies. *Educational Media International*, *45*(1), 3–15. https://doi.org/10.1080/09523980701847115

D'Ambra, J., Wilson, C. S., & Akter, S. (2017). Affordance theory and e-books: Evaluating the e-reading experience using netnography. *Personal and Ubiquitous Computing*, 21(1), 39–53.

Mailund, T. (2019). *The joys of hashing: Hash table programming with C* (1st ed.). Apress. https://doi.org/10.1007/978-1-4842-4066-3

Rustiraning, A. K. (2022). Indigenous people and social media use: Social media affordances actualization of BaduyCraft and @SaungGunung.id *Jurnal Komunikasi*, *14*(1), 120–139. https://doi.org/10.24912/jk.v14i1.12220

Sanchez, J., & Canton, M. P. (2002). *Java programming for engineers*. CRC Press.