LAPORAN PRAKTIKUM
Modul 3
Abstract Data Type (ADT)

Disusun Oleh:
Jauhar Fajar Zuhair
2311104072
S1SE-07-2

Dosen :
Wahyu Andri Saputra, S.Pd., M.Eng.

PROGRAM STUDI S1 SOFTWARE ENGINEERING
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY
PURWOKERTO

2024

1. Tujuan

- Memperoleh wawasan mendalam tentang prinsip-prinsip ADT dan signifikansinya dalam desain program.
- Mengembangkan kemampuan untuk merancang dan mengimplementasikan ADT dalam bahasa pemrograman yang relevan.
- Mempelajari cara mengorganisir kode dengan lebih terstruktur menggunakan pendekatan ADT.
- Meningkatkan keterampilan dalam menciptakan program yang modular dan mudah dipelihara.
- Memahami keuntungan penggunaan ADT dalam pengembangan perangkat lunak skala besar.

2. Landasan Teori

Konsep Abstract Data Type (ADT) dalam Pemrograman

Abstract Data Type (ADT) merupakan konsep penting dalam pemrograman yang menggabungkan tipe data dengan operasi-operasi dasarnya. ADT tidak hanya mendefinisikan struktur data, tetapi juga menyertakan aturan dan aksioma yang berlaku pada tipe tersebut. Ini menciptakan definisi statis yang dapat digunakan sebagai dasar untuk pengembangan program yang lebih kompleks.

Komponen Utama ADT:

1. Tipe Data: Mendefinisikan struktur data yang akan digunakan.
2. Primitif: Sekumpulan operasi dasar yang dapat dilakukan pada tipe data tersebut.

ADT dapat bersifat hierarkis, di mana satu ADT bisa terdiri dari ADT lainnya. Misalnya, ADT waktu bisa terdiri dari ADT jam dan tanggal, atau ADT garis yang terdiri dari dua ADT titik.

Implementasi ADT dalam Bahasa Pemrograman:

Ketika mengimplementasikan ADT dalam bahasa pemrograman seperti C++, tipe data biasanya diterjemahkan menjadi struct, sedangkan primitif diimplementasikan sebagai fungsi atau prosedur.

Kategori Primitif dalam ADT:

1. Konstruktor: Membentuk nilai tipe data baru.

2. Selektor: Mengakses komponen dari tipe data.
3. Mutator: Mengubah nilai komponen.
4. Validator: Memeriksa kevalidan nilai tipe data.
5. Destruktor: Menghapus objek dan membebaskan memori.
6. I/O Handler: Menangani input dan output.
7. Operator Relasional: Membandingkan nilai antar objek.
8. Operator Aritmatika: Melakukan operasi matematika pada objek.
9. Konverter: Mengubah tipe data ke bentuk lain dan sebaliknya.

Struktur Implementasi ADT:

ADT biasanya diimplementasikan dalam tiga modul:

1. Header File (.h): Berisi definisi tipe data dan deklarasi fungsi-fungsi primitif.
2. Source File (.cpp): Berisi implementasi dari fungsi-fungsi yang dideklarasikan di header.
3. Driver Program: Program utama yang menggunakan ADT tersebut.

Keuntungan Penggunaan ADT:

1. Modularitas: Memisahkan implementasi dari penggunaan, meningkatkan maintainability.
2. Abstraksi: Menyembunyikan detail implementasi, fokus pada fungsionalitas.
3. Reusabilitas: Memungkinkan penggunaan kembali kode di berbagai proyek.
4. Keamanan: Membatasi akses langsung ke data, mengurangi risiko kesalahan.

3. **Guided**

Struktur Data(ADT)

```
struct mahasiswa {
    char nim[10];
    int nilai1, nilai2;
};
```

Ini mendefinisikan ADT 'mahasiswa' yang memiliki tiga anggota:

- nim: array karakter untuk menyimpan NIM mahasiswa (maksimal 10 karakter)
- nilai1 dan nilai2: dua nilai integer untuk menyimpan nilai mahasiswa

Fungsi Input(Primitf):

```
void inputMhs(mahasiswa &m) {
    cout << "input nim";
    cin >> (m).nim;
    cout << "input nilai: ";
    cin >> (m).nilai1;
    cout << "input nilai2: ";
    cin >> (m).nilai2;
}
```

Fungsi ini mengambil input dari pengguna untuk mengisi data mahasiswa. Perhatikan penggunaan referensi (&m) yang memungkinkan fungsi untuk mengubah nilai struct asli.

Fungsi Perhitungan Rata-Rata(Primitif):

```
float rata2(mahasiswa m) {
    return(m.nilai1+m.nilai2)/2;
}
```

Fungsi ini menghitung rata-rata dari dua nilai mahasiswa.

Fungsi Main:

```
int main(){
    mahasiswa mhs;
    inputMhs(mhs);
    cout << rata2(mhs);
    return 0;
}
```

Fungsi main mendemonstrasikan penggunaan ADT:

- Membuat objek mhs dari struct mahasiswa
- Memanggil inputMhs() untuk mengisi data
- Memanggil rata2() untuk menghitung dan mencetak rata-rata nilai

Penjelasan:

1. Implementasi ADT: Kode ini menunjukkan implementasi dasar ADT, di mana struktur data (struct mahasiswa) dikombinasikan dengan operasi-operasi yang bekerja pada struktur tersebut (inputMhs dan rata2).
2. Enkapsulasi: Meskipun ini adalah implementasi sederhana, kode sudah menunjukkan prinsip enkapsulasi di mana data (nim, nilai1, nilai2) dan operasi (inputMhs, rata2) dikelompokkan bersama.
3. Abstraksi: Pengguna (dalam hal ini fungsi main) tidak perlu tahu detail internal tentang bagaimana data disimpan atau bagaimana rata-rata dihitung. Mereka hanya perlu memanggil fungsi yang disediakan.
4. Modularitas: Kode dibagi menjadi fungsi-fungsi terpisah (inputMhs, rata2) yang masing-masing memiliki tugas spesifik, meningkatkan keterbacaan dan pemeliharaan kode.
5. Penggunaan Referensi: Fungsi inputMhs menggunakan parameter referensi (&m) untuk mengubah nilai struct secara langsung, mendemonstrasikan cara efisien untuk memodifikasi ADT.
6. Perhitungan Sederhana: Fungsi rata2 menunjukkan bagaimana operasi dapat dilakukan pada data dalam ADT.

4. **Unguided**
   1.

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

struct Mahasiswa {
    string nama;
    string nim;
    float uts;
    float uas;
    float tugas;
    float nilai_akhir;
};
float hitungNilaiAkhir(float uts, float uas, float tugas) {
    return 0.3 * uts + 0.4 * uas + 0.3 * tugas;
}
void inputMahasiswa(Mahasiswa &mhs) {
    cout << "Masukkan Nama: ";
    getline(cin >> ws, mhs.nama);
    cout << "Masukkan NIM: ";
    cin >> mhs.nim;
    cout << "Masukkan nilai UTS: ";
    cin >> mhs.uts;
    cout << "Masukkan nilai UAS: ";
    cin >> mhs.uas;
    cout << "Masukkan nilai Tugas: ";
    cin >> mhs.tugas;
    mhs.nilai_akhir = hitungNilaiAkhir(mhs.uts, mhs.uas, mhs.tugas);
```

```cpp
void tampilkanMahasiswa(const Mahasiswa &mhs) {
    cout << setw(20) << left << mhs.nama
         << setw(15) << mhs.nim
         << setw(10) << mhs.uts
         << setw(10) << mhs.uas
         << setw(10) << mhs.tugas
         << setw(10) << fixed << setprecision(2) << mhs.nilai_akhir << endl;
```

```cpp
int main() {
    const int MAX_MAHASISWA = 10;
    Mahasiswa daftarMahasiswa[MAX_MAHASISWA];
    int jumlahMahasiswa = 0;

    while (jumlahMahasiswa < MAX_MAHASISWA) {
        cout << "\nMasukkan data mahasiswa ke-" << jumlahMahasiswa + 1 << ":\n";
        inputMahasiswa(&daftarMahasiswa[jumlahMahasiswa]);
        jumlahMahasiswa++;

        char lanjut;
        cout << "Tambah data mahasiswa lagi? (y/n): ";
        cin >> lanjut;
        if (lanjut != 'y' && lanjut != 'Y') break;
    }

    cout << "\nDaftar Mahasiswa:\n";
    cout << setw(20) << left << "Nama"
         << setw(15) << "NIM"
         << setw(10) << "UTS"
         << setw(10) << "UAS"
         << setw(10) << "Tugas"
         << setw(10) << "Nilai Akhir" << endl;
    cout << string(75, '-') << endl;

    for (int i = 0; i < jumlahMahasiswa; i++) {
        tampilkanMahasiswa(daftarMahasiswa[i]);
    }
    return 0;
```

Penjelasan program:

1. Kita mendefinisikan struct Mahasiswa yang memiliki field sesuai dengan yang diminta: nama, nim, uts, uas, tugas, dan nilai_akhir.
2. Fungsi hitungNilaiAkhir() digunakan untuk menghitung nilai akhir sesuai dengan rumus yang diberikan.
3. Fungsi inputMahasiswa() digunakan untuk memasukkan data mahasiswa.
4. Fungsi tampilkanMahasiswa() digunakan untuk menampilkan data seorang mahasiswa dengan format yang rapi.
5. Dalam main():
    - Kita membuat array daftarMahasiswa dengan ukuran maksimal 10.
    - Program akan terus meminta input data mahasiswa sampai pengguna

memilih untuk berhenti atau jumlah maksimum mahasiswa tercapai.
- Setelah input selesai, program menampilkan daftar semua mahasiswa yang telah dimasukkan.

6. Program menggunakan manipulator seperti setw(), left, fixed, dan setprecision() untuk memformat output agar lebih rapi.

2. pelajaran.h

```
#ifndef PELAJARAN_H
#define PELAJARAN_H

#include <string>

struct pelajaran {
    std::string namaMapel;
    std::string kodeMapel;
};

pelajaran create_pelajaran(std::string namapel, std::string kodepel);
void tampil_pelajaran(pelajaran pel);

#endif // PELAJARAN_H
```

Pelajaran.cpp

```
#include "pelajaran.h"
#include <iostream>

pelajaran create_pelajaran(std::string namapel, std::string kodepel) {
    pelajaran pel;
    pel.namaMapel = namapel;
    pel.kodeMapel = kodepel;
    return pel;
}

void tampil_pelajaran(pelajaran pel) {
    std::cout << "Nama pelajaran : " << pel.namaMapel << std::endl;
    std::cout << "Kode           : " << pel.kodeMapel << std::endl;
}
```

Main.cpp

```cpp
#include "pelajaran.h"
#include <string>

using namespace std;

int main() {
    string namapel = "Struktur Data";
    string kodepel = "STD";
    pelajaran pel = create_pelajaran( namapel,  kodepel);
    tampil_pelajaran( pel);

    return 0;
}
```

1. "pelajaran.h":
    - Mendefinisikan struktur pelajaran dengan dua anggota: namaMapel dan kodeMapel.
    - Mendeklarasikan fungsi create_pelajaran dan prosedur tampil_pelajaran.
2. "pelajaran.cpp":
    - Mengimplementasikan fungsi create_pelajaran yang membuat dan mengembalikan objek pelajaran.
    - Mengimplementasikan prosedur tampil_pelajaran yang mencetak informasi pelajaran.
3. "main.cpp":
    - Menggunakan ADT pelajaran yang telah dibuat.
    - Membuat objek pelajaran dengan nama "Struktur Data" dan kode "STD".
    - Menampilkan informasi pelajaran menggunakan prosedur tampil_pelajaran.

3.

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

void tampilkanArray(int arr[3][3]) {
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            cout << arr[i][j] << " ";
        }
        cout << endl;
    }
    cout << endl;
}

void tukarElemen(int arr1[3][3], int arr2[3][3], int baris, int kolom) {
    int temp = arr1[baris][kolom];
    arr1[baris][kolom] = arr2[baris][kolom];
    arr2[baris][kolom] = temp;
}

void tukarPointer(int* ptr1, int* ptr2) {
    int temp = *ptr1;
    *ptr1 = *ptr2;
    *ptr2 = temp;
}
```

```
int main() {
    int array1[3][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
    int array2[3][3] = {{10, 11, 12}, {13, 14, 15}, {16, 17, 18}};

    int a = 100, b = 200;
    int *ptr1 = &a, *ptr2 = &b;

    cout << "Array 1 sebelum pertukaran:" << endl;
    tampilkanArray(array1);

    cout << "Array 2 sebelum pertukaran:" << endl;
    tampilkanArray(array2);

    tukarElemen(arr1:array1, arr2:array2, baris:1, kolom:1);

    cout << "Array 1 setelah pertukaran elemen:" << endl;
    tampilkanArray(array1);

    cout << "Array 2 setelah pertukaran elemen:" << endl;
    tampilkanArray(array2);

    cout << "Nilai pointer sebelum pertukaran:" << endl;
    cout << "ptr1: " << *ptr1 << ", ptr2: " << *ptr2 << endl;

    tukarPointer(ptr1, ptr2);

    cout << "Nilai pointer setelah pertukaran:" << endl;
    cout << "ptr1: " << *ptr1 << ", ptr2: " << *ptr2 << endl;
    return 0;
```

Penjelasan program:

1. Program ini memenuhi semua ketentuan yang diminta:
    - Terdapat 2 buah array 2D integer berukuran 3x3 (array1 dan array2).
    - Terdapat 2 buah pointer integer (ptr1 dan ptr2).
2. Fungsi tampilkanArray() digunakan untuk menampilkan isi sebuah array integer 2D.

3. Fungsi tukarElemen() digunakan untuk menukarkan isi dari 2 array integer 2D pada posisi tertentu.
4. Fungsi tukarPointer() digunakan untuk menukarkan isi dari variabel yang ditunjuk oleh 2 buah pointer.
5. Dalam main():
    - Kita menginisialisasi array dan pointer.
    - Menampilkan array sebelum pertukaran.
    - Melakukan pertukaran elemen array pada posisi (1,1).
    - Menampilkan array setelah pertukaran.
    - Menampilkan nilai yang ditunjuk pointer sebelum pertukaran.
    - Melakukan pertukaran nilai yang ditunjuk oleh pointer.
    - Menampilkan nilai yang ditunjuk pointer setelah pertukaran.

4. Kesimpulan

- Materi ini fokus pada konsep dan implementasi Abstract Data Type (ADT) dalam pemrograman C++, menekankan pemisahan antara definisi dan implementasi, serta memberikan latihan praktis untuk penerapan konsep tersebut.