LAPORAN PRAKTIKUM

Modul 3

“ABSTRACT DATA TYPE (ADT)”

Disusun Oleh:

Alya Rabani - 2311104076

S1SE-07-B

Dosen :

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

PROGRAM STUDI S1 SOFTWARE ENGINEERING

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY

PURWOKERTO

2024

1. Tujuan

- Memahami konsep Abstract Data Type (ADT) dan penggunaannya dalam pemrograman.

2. Landasan Teori

Abstrak Data Tipe (ADT) adalah suatu konsep dalam pemrograman yang mendefinisikan sebuah tipe data baru beserta sekumpulan operasi yang dapat dilakukan pada tipe data tersebut. ADT juga mencakup aturan-aturan yang harus dipenuhi oleh data tersebut (invarian) dan hubungan antara operasi-operasi (aksioma).

ADT dapat terdiri dari ADT lain, sehingga memungkinkan kita untuk membangun struktur data yang kompleks. Setiap ADT memiliki operasi-operasi dasar seperti pembuatan objek (konstruktor), pengaksesan komponen (selector), pengubahan nilai komponen, validasi, penghancuran objek, input/output, perbandingan, operasi aritmatika, dan konversi tipe.

Implementasi ADT biasanya dibagi menjadi dua bagian utama yaitu, definisi tipe dan operasi (header file) serta implementasi operasi (file sumber). Definisi tipe menjelaskan struktur data, sedangkan implementasi operasi memberikan detail tentang bagaimana operasi-operasi tersebut bekerja.
Dengan menggunakan ADT, kita dapat membuat program yang lebih modular, mudah dipahami, dan dapat digunakan kembali.

ADT memiliki berbagai konsep kunci seperti:

- Abstraksi: ADT memfokuskan pada *apa* yang dapat dilakukan dengan data, bukan pada bagaimana data tersebut diimplementasikan secara internal. Hal ini memungkinkan kita untuk menggunakan ADT tanpa perlu mengetahui detail implementasinya.
- Enkapsulasi: Data dan operasi yang terkait dengan ADT dibungkus dalam satu unit, sehingga melindungi data dari akses yang tidak sah dan menjaga integritas data.
- Spesifikasi: ADT didefinisikan melalui sekumpulan operasi yang dapat dilakukan pada data, beserta dengan pra-kondisi dan pasca-kondisi dari setiap operasi.

Contoh ADT

- Stack: ADT yang mengikuti prinsip LIFO (Last In, First Out). Operasi yang umum pada stack adalah push (menambahkan elemen), pop (menghapus elemen teratas), dan peek (melihat elemen teratas).
- queve ADT yang mengikuti prinsip FIFO (First In, First Out). Operasi yang umum pada queue adalah enqueue (menambahkan elemen) dan dequeue (menghapus elemen terdepan).
- List: ADT yang menyimpan koleksi elemen yang urutannya dapat berubah. Operasi yang umum pada list adalah insert (menambahkan elemen), delete (menghapus elemen), dan search (mencari elemen).
- Tree: ADT yang merepresentasikan struktur hierarkis. Operasi yang umum pada tree adalah insert, delete, search, dan traversal.

3. Guided

Program ini adalah menghitung rata-rata dua nilai untuk seorang mahasiswa. Program ini menggunakan struct untuk merepresentasikan mahasiswa, yang memiliki tiga anggota: nim (nomor induk mahasiswa), nilai1 (nilai pertama), dan nilai2 (nilai kedua). Program ini meminta pengguna untuk memasukkan nomor induk mahasiswa dan dua nilai, menghitung rata-rata, dan kemudian menampilkan hasilnya.

Pertama menginclude file header iostream dan membawa namespace std ke dalam scope. Lalu, definisikan struct yang disebut mahasiswa, yang merepresentasikan mahasiswa. Pada struct ini memiliki tiga anggota yaitu nim, nilai1 dan nilai2. Kemudian dibuat fungsi yang mengambil struct mahasiswa oleh referensi dan memasukkan data untuk mahasiswa.

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

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

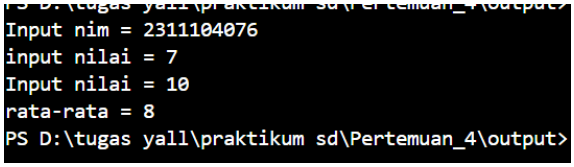
Setelah itu, dibuat fungsi utama untuk mendeklarasikan struct mahasiswa yang disebut mhs. Lalu, memanggil fungsi inputMhs untuk memasukkan data untuk mahasiswa. Panggil juga fungsi rata2 untuk menghitung rata-rata dua nilai. Gunakan fungsi cout untuk menampilkan hasil dan kembalikan 0 untuk menunjukkan eksekusi yang berhasil.

```
void inputMhs(mahasiswa &m);
float rata2(mahasiswa m);

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

Selanjutnya, mengambil struct mahasiswa oleh referensi untuk meminta pengguna untuk memasukkan nomor induk mahasiswa, nilai pertama, dan nilai kedua menggunakan cout. Lalu, gunakan cin untuk membaca input dari pengguna dan menyimpannya dalam anggota yang sesuai dari struct mahasiswa. Kemudian, dibuat fungsi rata-rata untukmengambil struct mahasiswa oleh nilai.

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

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

Output yang dihasilkan seperti berikut:

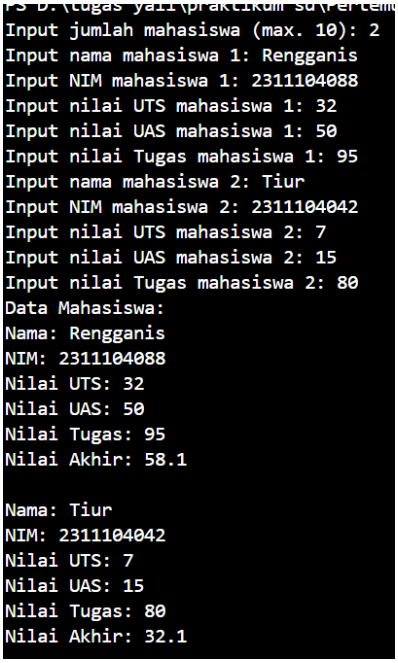


4. Unguided

1. Program ini dirancang untuk menghitung dan menampilkan nilai akhir mahasiswa. Dimulai dengan mendefinisikan struktur data Mahasiswa dan fungsi hitungNilaiAkhir untuk melakukan perhitungan nilai akhir. Pada fungsi main, program meminta pengguna untuk memasukkan jumlah mahasiswa yang ingin diproses, dengan batasan maksimal 10 mahasiswa. Setelah itu, program akan menginput data mahasiswa secara individu, termasuk nilai-nilai yang dibutuhkan untuk menghitung nilai akhir. Proses perhitungan nilai akhir setiap mahasiswa dilakukan menggunakan fungsi hitungNilaiAkhir. Terakhir, program akan menampilkan data lengkap setiap mahasiswa, termasuk nilai akhir yang telah dihitung, ke layar. Jika pengguna memasukkan jumlah mahasiswa melebihi batas maksimal, program akan memberikan pesan kesalahan dan berhenti.

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

struct Mahasiswa {
    string nama;
    string nim;
    float uts, uas, tugas, nilaiAkhir;
};

float hitungNilaiAkhir(float uts, float uas, float tugas) {
    return 0.3 * uts + 0.4 * uas + 0.3 * tugas;
}

void inputMahasiswa(Mahasiswa mhs[], int n) {
    for (int i = 0; i < n; i++) {
        cout << "Input nama mahasiswa " << i + 1 << ": ";
        cin >> mhs[i].nama;
        cout << "Input NIM mahasiswa " << i + 1 << ": ";
        cin >> mhs[i].nim;
        cout << "Input nilai UTS mahasiswa " << i + 1 << ": ";
        cin >> mhs[i].uts;
        cout << "Input nilai UAS mahasiswa " << i + 1 << ": ";
        cin >> mhs[i].uas;
        cout << "Input nilai Tugas mahasiswa " << i + 1 << ": ";
        cin >> mhs[i].tugas;
        mhs[i].nilaiAkhir = hitungNilaiAkhir(mhs[i].uts, mhs[i].uas, mhs[i].tugas);
    }
}

void displayMahasiswa(Mahasiswa mhs[], int n) {
    cout << "Data Mahasiswa:" << endl;
    for (int i = 0; i < n; i++) {
        cout << "Nama: " << mhs[i].nama << endl;
        cout << "NIM: " << mhs[i].nim << endl;
        cout << "Nilai UTS: " << mhs[i].uts << endl;
        cout << "Nilai UAS: " << mhs[i].uas << endl;
        cout << "Nilai Tugas: " << mhs[i].tugas << endl;
        cout << "Nilai Akhir: " << mhs[i].nilaiAkhir << endl;
        cout << endl;
    }
}

int main() {
    const int MAX_MAHASISWA = 10;
    Mahasiswa mhs[MAX_MAHASISWA];
    int n;
    cout << "Input jumlah mahasiswa (max. 10): ";
    cin >> n;
    if (n > MAX_MAHASISWA) {
        cout << "Jumlah mahasiswa melebihi batas!" << endl;
        return 1;
    }
    inputMahasiswa(mhs, n);
    displayMahasiswa(mhs, n);
    return 0;
}

```

Dari program tersebt dihasilkan output seperti berikut:



2. Program ini menunjukkan cara membuat objek pelajaran baru menggunakan fungsi create_pelajaran dan menampilkan detail pelajaran menggunakan prosedur tampil_pelajaran. Struct pelajaran digunakan untuk merepresentasikan pelajaran, yang berisi atribut seperti nama pelajaran dan kode pelajaran. Fungsi create_pelajaran membuat objek pelajaran baru dengan nama pelajaran dan kode pelajaran yang diberikan sebagai parameter. Prosedur tampil_pelajaran menampilkan detail pelajaran, yaitu nama pelajaran dan kode pelajaran.

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

// Definisi ADT Pelajaran di file "pelajaran.h"
struct pelajaran {
    string namamapel;
    string kodepel;
};

// Fungsi untuk membuat objek Pelajaran baru
pelajaran create_pelajaran(string namapel, string kodepel) {
    pelajaran pel;
    pel.namamapel = namapel;
    pel.kodepel = kodepel;
    return pel;
}

// Prosedur untuk menampilkan detail Pelajaran
void tampil_pelajaran(pelajaran pel) {
    cout << "nama pelajaran: " << pel.namamapel << endl;
    cout << "nilai: " << pel.kodepel << endl;
}

int main() {
    // Membuat objek Pelajaran
    pelajaran pel = create_pelajaran("Struktur Data", "STD");

    // Menampilkan detail Pelajaran
    tampil_pelajaran(pel);

    return 0;
}
```

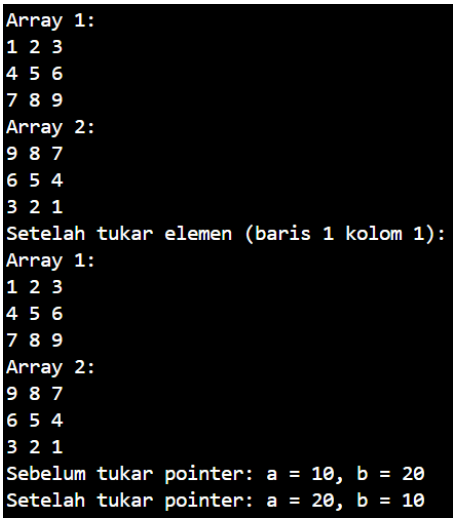
Program tersebut akan menghasilkan output seperti berikut:

```
PS D:\tugas yall\praktikum sd\
nama pelajaran: Struktur Data
nilai: STD
PS D:\tugas yall\praktikum sd\
```

3. Program ini dirancang untuk mengilustrasikan konsep array dua dimensi dan pointer. Program ini bekerja dengan memanipulasi dua buah array dua dimensi dan dua pointer integer. Terdapat tiga fungsi utama yang menjalankan tugas-tugas spesifik: fungsi displayArray menampilkan isi dari sebuah array dua dimensi,
   fungsi swapArrayElements menukar elemen pada posisi tertentu di antara dua array
   fungsi swapPointers menukar nilai yang ditunjuk oleh dua pointer.
   Dalam fungsi main, program menginisialisasi kedua array dengan nilai, menampilkan isi awal kedua array, kemudian melakukan penukaran elemen pada posisi yang ditentukan oleh pengguna. Selain itu, program juga mendemonstrasikan cara menukar nilai dua variabel integer dengan menggunakan pointer.

```cpp
#include <iostream>

using namespace std;

// Fungsi untuk menampilkan isi array 2D
void displayArray(int arr[3][3]) {
    for (int i = 0; i < 3; ++i) {
        for (int j = 0; j < 3; ++j) {
            cout << arr[i][j] << " ";
        }
        cout << endl;
    }
}

// Fungsi untuk menukarkan isi dari dua array 2D pada posisi tertentu
void swapArrayElements(int arr1[3][3], int arr2[3][3], int row, int col) {
    if (row >= 0 && row < 3 && col >= 0 && col < 3) {
        int temp = arr1[row][col];
        arr1[row][col] = arr2[row][col];
        arr2[row][col] = temp;
    } else {
        cout << "Indeks di luar batas!" << endl;
    }
}

// Fungsi untuk menukarkan isi dari variabel yang ditunjuk oleh dua pointer
void swapPointers(int* ptr1, int* ptr2) {
    int temp = *ptr1;
    *ptr1 = *ptr2;
    *ptr2 = temp;
}

int main() {
    // Inisialisasi dua buah array 2D
    int array1[3][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
    int array2[3][3] = {{9, 8, 7}, {6, 5, 4}, {3, 2, 1}};

    // Menampilkan isi kedua array
    cout << "Array 1:" << endl;
    displayArray(array1);

    cout << "Array 2:" << endl;
    displayArray(array2);

    // Menukarkan elemen pada posisi tertentu (contoh: baris 1 kolom 1)
    swapArrayElements(array1, array2, 1, 1);

    cout << "Setelah tukar elemen (baris 1 kolom 1):" << endl;

    cout << "Array 1:" << endl;
    displayArray(array1);

    cout << "Array 2:" << endl;
    displayArray(array2);

    // Inisialisasi dua pointer integer
    int a = 10;
    int b = 20;

    int* ptrA = &a;
    int* ptrB = &b;

    // Menampilkan nilai sebelum tukar
    cout << "Sebelum tukar pointer: a = " << *ptrA << ", b = " << *ptrB << endl;

    // Menukarkan nilai yang ditunjuk oleh kedua pointer
    swapPointers(ptrA, ptrB);

    // Menampilkan nilai setelah tukar
    cout << "Setelah tukar pointer: a = " << *ptrA << ", b = " << *ptrB << endl;

    return 0;
}
```

Dari program tersebut dihasilkan output seperti berikut:



5. Kesimpulan

Dari laporan tersebut dapat disimpulkan bahwa konsep Abstract Data Type (ADT) sangat penting dalam pemrograman, karena memungkinkan pembuatan struktur data yang lebih modular, mudah dipahami, dan dapat digunakan kembali. ADT menyediakan abstraksi yang memisahkan operasi yang dapat dilakukan pada data dari cara data diimplementasikan, menjaga enkapsulasi dan integritas data. Melalui praktikum ini, implementasi ADT dalam berbagai program, seperti pengolahan data mahasiswa, menunjukkan penerapan konsep ini untuk menyederhanakan pemrosesan dan manipulasi data dalam konteks nyata.