LAPORAN PRAKTIKUM MODUL 3

ABSTRACT DATA TYPE

DISUSUN OLEH :

Fauzan Rofif Ardiyanto

S1SE-06-02

DOSEN :

WAHYU ANDI SAPUTRA

PROGRAM STUDI S1 SOFTWARE ENGINEERING

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO 2024

Tujuan

Modul ini bertujuan untuk memperkenalkan konsep **Abstract Data Type (ADT)** di C++ dan mengimplementasikannya menggunakan struktur data. Mahasiswa diharapkan mampu:

1. Memahami konsep ADT dan abstraksi.
2. Mengimplementasikan ADT di C++ dengan memisahkan antarmuka dan implementasi.
3. Menggunakan ADT untuk memanipulasi data dengan aman dan efisien.

Landasan Teori

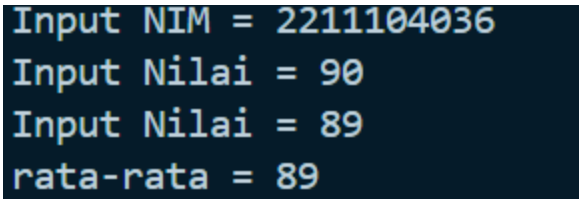
1. Abstract Data Type (ADT) adalah tipe data yang didefinisikan oleh perilakunya (operasi yang dapat dilakukan) tanpa memaparkan detail implementasinya.
2. ADT dalam C++ biasanya diimplementasikan dengan class atau struct, yang terdiri dari data dan fungsi yang mengoperasikannya.
3. Manfaat ADT:
    - Memungkinkan pemrograman modular dan reusable code.
    - Melindungi data dengan enkapsulasi.
4. Pemisahan Antarmuka dan Implementasi: Di C++, deklarasi ADT dilakukan di header files (.h) dan implementasinya di source files (.cpp), memudahkan pemeliharaan dan pengembangan.

Guided

Modul 3 Abstrack Data Type

Code :

```
#include <iostream>

using namespace std;

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

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

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

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;
}
```



Deskripsi :

Deskripsi program ini adalah sebagai berikut:

Program ini menggunakan **struct** untuk mendefinisikan tipe data bernama mahasiswa, yang menyimpan informasi tentang **NIM** serta dua nilai mahasiswa. Program ini juga memiliki dua fungsi:

1. **inputMhs(mahasiswa &m)**:
    - Fungsi ini menerima input dari pengguna berupa NIM, nilai pertama, dan nilai kedua. Data ini kemudian disimpan dalam struct mahasiswa yang dioper oleh referensi.
2. **rata2(mahasiswa m)**:
    - Fungsi ini menghitung rata-rata dari dua nilai yang dimiliki mahasiswa dengan cara menjumlahkan kedua nilai dan membagi hasilnya dengan 2.

Pada fungsi **main()**, program meminta pengguna untuk memasukkan data mahasiswa, dan kemudian menampilkan rata-rata nilai tersebut.

Unguided

1. **Nomor 1**

   Code :

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

const int MAX_MAHASISWA = 10;

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

// Fungsi untuk menghitung nilai akhir
float hitungNilaiAkhir(float uts, float uas, float tugas)
{
    return (0.3 * uts) + (0.4 * uas) + (0.3 * tugas);
}

int main()
{
    Mahasiswa mahasiswa[MAX_MAHASISWA];
    int jumlahMahasiswa = 0;

    cout << "Masukkan jumlah mahasiswa (maks. 10): ";
    cin >> jumlahMahasiswa;

    if (jumlahMahasiswa > MAX_MAHASISWA)
    {
        cout << "Jumlah mahasiswa melebihi batas maksimal." << endl;
        return 1;
    }

    // Input data mahasiswa
    for (int i = 0; i < jumlahMahasiswa; i++)
    {
        cout << "Data mahasiswa ke-" << i + 1 << endl;
        cout << "Nama: ";
        cin.ignore(); // Membersihkan input buffer
        getline(cin, mahasiswa[i].nama);
        cout << "NIM: ";
        cin >> mahasiswa[i].nim;
        cout << "Nilai UTS: ";
        cin >> mahasiswa[i].uts;
        cout << "Nilai UAS: ";
        cin >> mahasiswa[i].uas;
        cout << "Nilai Tugas: ";
        cin >> mahasiswa[i].tugas;

        // Hitung nilai akhir
        mahasiswa[i].nilaiAkhir = hitungNilaiAkhir(mahasiswa[i].uts, mahasiswa[i].uas, mahasiswa[i].tugas);
    }

    // Output data mahasiswa
    cout << "\nData Mahasiswa dan Nilai Akhir:\n";
    for (int i = 0; i < jumlahMahasiswa; i++)
    {
        cout << "Mahasiswa ke-" << i + 1 << endl;
        cout << "Nama        : " << mahasiswa[i].nama << endl;
        cout << "NIM         : " << mahasiswa[i].nim << endl;
        cout << "Nilai UTS : " << mahasiswa[i].uts << endl;
        cout << "Nilai UAS : " << mahasiswa[i].uas << endl;
        cout << "Nilai Tugas: " << mahasiswa[i].tugas << endl;
        cout << "Nilai Akhir: " << mahasiswa[i].nilaiAkhir << endl;
        cout << "------------------------------" << endl;
    }

    return 0;
}
```

Running :

```
Masukkan jumlah mahasiswa (maks. 10): 2
Data mahasiswa ke-1
Nama: Fauzan
NIM: 2211104036
Nilai UTS: 90
Nilai UAS: 80
Nilai Tugas: 89
Data mahasiswa ke-2
Nama: Zivana Afra Yulianto
NIM: 89
Nilai UTS: 88
Nilai UAS: 87
Nilai Tugas: 88
```

(input)

```
Data Mahasiswa dan Nilai Akhir:
Mahasiswa ke-1
Nama       : Fauzan
NIM        : 2211104036
Nilai UTS  : 90
Nilai UAS  : 80
Nilai Tugas: 89
Nilai Akhir: 85.7
-------------------------
Mahasiswa ke-2
Nama       : Zivana Afra Yulianto
NIM        : 89
Nilai UTS  : 88
Nilai UAS  : 87
Nilai Tugas: 88
Nilai Akhir: 87.6
-------------------------
```

(output)

Deskripsi :

Program ini terdiri dari beberapa bagian utama:

1. **Struct Mahasiswa**: Struct Mahasiswa digunakan untuk menyimpan informasi setiap mahasiswa, yaitu nama, nim, uts, uas, tugas, dan nilaiAkhir.
2. **Fungsi hitungNilaiAkhir**: Fungsi ini menerima input berupa nilai UTS, UAS, dan tugas, kemudian menghitung nilai akhir mahasiswa dengan rumus:

   $$\text{nilai akhir} = 0.3 \times \text{UTS} + 0.4 \times \text{UAS} + 0.3 \times \text{Tugas}$$

3. **Input Data Mahasiswa**: Program meminta pengguna untuk memasukkan jumlah mahasiswa yang akan diinput (maksimal 10), kemudian untuk setiap mahasiswa, program meminta nama, NIM, nilai UTS, UAS, dan tugas. Setelah itu, nilai akhir mahasiswa dihitung menggunakan fungsi hitungNilaiAkhir.
4. **Output Data Mahasiswa**: Setelah semua data diinput, program akan menampilkan informasi masing-masing mahasiswa, termasuk nilai UTS, UAS, tugas, dan nilai akhirnya.

2. Nomor 2
Code :

Pelajar.h

```
#ifndef PELAJARAN_H
#define PELAJARAN_H

#include <string>
using namespace std;

struct pelajaran
{
    string namaMapel;
    string kodeMapel;
};

// Deklarasi fungsi create_pelajaran
pelajaran create_pelajaran(string namaMapel, string kodeMapel);

// Deklarasi prosedur tampil_pelajaran
void tampil_pelajaran(pelajaran pel);

#endif
```

Pelajar.cpp

```
#ifndef PELAJARAN_H
#define PELAJARAN_H

#include <string>#include "pelajaran.h"
#include <iostream>
using namespace std;

// Fungsi create_pelajaran
pelajaran create_pelajaran(string namaMapel, string kodeMapel)
{
    pelajaran pel;
    pel.namaMapel = namaMapel;
    pel.kodeMapel = kodeMapel;
    return pel;
}

// Prosedur tampil_pelajaran
void tampil_pelajaran(pelajaran pel)
{
    cout << "Nama pelajaran : " << pel.namaMapel << endl;
    cout << "Nilai          : " << pel.kodeMapel << endl;
}


using namespace std;

struct pelajaran
{
    string namaMapel;
    string kodeMapel;
};

// Deklarasi fungsi create_pelajaran
pelajaran create_pelajaran(string namaMapel, string kodeMapel);

// Deklarasi prosedur tampil_pelajaran
void tampil_pelajaran(pelajaran pel);

#endif
```

Main.cpp

```
#include "pelajaran.h"
#include <iostream>
using namespace std;

int main()
{
    string namaPel = "Struktur Data";
    string kodePel = "STD";

    pelajaran pel = create_pelajaran(namaPel, kodePel);
    tampil_pelajaran(pel);

    return 0;
}
```

Running :

```
nama pelajaran : Struktur Data
nilai : STD
```

Deskripsi :

1. pelajaran.h:
   - Mendefinisikan tipe data `pelajaran` yang memiliki dua atribut: `namaMapel` (nama mata pelajaran) dan `kodeMapel` (kode mata pelajaran).
   - Deklarasi fungsi `create_pelajaran` untuk membuat objek pelajaran baru.
   - Deklarasi prosedur `tampil_pelajaran` untuk menampilkan informasi pelajaran.
2. pelajaran.cpp:
   - Implementasi fungsi `create_pelajaran` yang mengisi atribut nama dan kode pelajaran.
   - Implementasi prosedur `tampil_pelajaran` yang menampilkan nama dan kode pelajaran.
3. main.cpp:
   - Program utama yang membuat objek `pelajaran` menggunakan `create_pelajaran` dan menampilkan datanya menggunakan `tampil_pelajaran`.

3. Nomor 3

Code :

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

// Fungsi untuk menampilkan isi array 2D
void printArray(int arr[3][3], const string &name)
{
    cout << "Isi " << name << ":\n";
    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 3; j++)
        {
            cout << arr[i][j] << " ";
        }
        cout << endl;
    }
}

// Fungsi untuk menukar elemen pada posisi tertentu antara dua array
void swapElements(int arr1[3][3], int arr2[3][3], int row, int col)
{
    int temp = arr1[row][col];
    arr1[row][col] = arr2[row][col];
    arr2[row][col] = temp;
}

// Fungsi untuk menukar isi dari dua pointer
void swapPointers(int *ptr1, int *ptr2)
{
    int temp = *ptr1;
    *ptr1 = *ptr2;
    *ptr2 = temp;
}

int main()
{
    // Deklarasi 2 buah array 2D integer berukuran 3x3
    int arr1[3][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
    int arr2[3][3] = {{9, 8, 7}, {6, 5, 4}, {3, 2, 1}};

    // Deklarasi 2 buah pointer integer
    int a = 10, b = 20;
    int *ptr1 = &a;
    int *ptr2 = &b;

    // Tampilkan isi awal dari kedua array
    printArray(arr1, "arr1");
    printArray(arr2, "arr2");

    // Tukar elemen di posisi tertentu (contoh: tukar elemen [1][1])
    cout << "\nMenukar elemen pada posisi [1][1] antara arr1 dan arr2...\n";
    swapElements(arr1, arr2, 1, 1);

    // Tampilkan isi setelah pertukaran
    printArray(arr1, "arr1");
    printArray(arr2, "arr2");

    // Tampilkan nilai sebelum menukar pointer
    cout << "\nSebelum swapPointers:\n";
    cout << "a = " << a << ", b = " << b << endl;

    // Tukar isi dari variabel yang ditunjuk oleh pointer
    swapPointers(ptr1, ptr2);

    // Tampilkan nilai setelah menukar pointer
    cout << "\nSetelah swapPointers:\n";
    cout << "a = " << a << ", b = " << b << endl;

    return 0;
}
```

Running :

```
Isi arr1:
1 2 3
4 5 6
7 8 9
Isi arr2:
9 8 7
6 5 4
3 2 1

Menukar elemen pada posisi [1][1] antara arr1 dan arr2...
Isi arr1:
1 2 3
4 5 6
7 8 9
Isi arr2:
9 8 7
6 5 4
3 2 1

Sebelum swapPointers:
a = 10, b = 20

Setelah swapPointers:
a = 20, b = 10
```

Deskripsi :

Program ini terdiri dari tiga fungsi utama:

1. **Fungsi `printArray`**: Fungsi ini digunakan untuk menampilkan isi dari sebuah array 2D berukuran 3x3. Setiap elemen ditampilkan dalam format matriks dengan penamaan array yang dapat diatur menggunakan parameter `name`.
2. **Fungsi `swapElements`**: Fungsi ini digunakan untuk menukar elemen antara dua array 2D pada posisi tertentu (baris dan kolom yang ditentukan). Pada contoh, elemen di posisi `[1][1]` dari dua array (`arr1` dan `arr2`) ditukar.
3. **Fungsi `swapPointers`**: Fungsi ini digunakan untuk menukar isi variabel yang ditunjuk oleh dua pointer integer. Variabel yang ditunjuk oleh `ptr1` dan `ptr2` akan bertukar nilai.

Alur Program:

- Dua array 2D (`arr1` dan `arr2`) dengan ukuran 3x3 dideklarasikan dan diisi dengan nilai awal.
- Dua pointer integer (`ptr1` dan `ptr2`) yang menunjuk ke variabel `a` dan `b` juga dideklarasikan.
- Program pertama-tama menampilkan isi awal dari kedua array.
- Kemudian, elemen pada posisi `[1][1]` dari `arr1` dan `arr2` ditukar, dan isi array setelah pertukaran ditampilkan.
- Selanjutnya, nilai variabel `a` dan `b` yang ditunjuk oleh `ptr1` dan `ptr2` ditampilkan sebelum dan sesudah menukar isinya menggunakan `swapPointers`.

Kesimpulan

Dalam modul 3 mengenai Abstract Data Type (ADT), kita mempelajari konsep dasar yang sangat penting dalam pemrograman, yaitu bagaimana memisahkan antara representasi data dan operasi yang dapat dilakukan terhadap data tersebut. ADT merupakan abstraksi yang menyembunyikan detail implementasi data, sehingga pengguna hanya berinteraksi melalui antarmuka yang telah disediakan tanpa perlu mengetahui bagaimana data tersebut diatur di dalamnya. Dengan pendekatan ini, ADT membantu meningkatkan modularitas dalam pemrograman, di mana bagian-bagian program dapat dikembangkan, diuji, atau diubah secara terpisah tanpa mempengaruhi komponen lain. Selain itu, ADT juga meningkatkan reusability, memungkinkan kode yang berkaitan dengan operasi data untuk digunakan kembali dalam konteks yang berbeda. Secara keseluruhan, ADT memberikan fleksibilitas yang lebih besar dalam desain dan pengembangan program dengan membuat kode lebih terstruktur, mudah dipelihara, dan diperluas.