LAPORAN PRAKTIKUM
Modul 3.
ABSTRACT DATA TYPE

Disusun Oleh:
Zhafir Zaidan Avail
S1-SE-07-2

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

PROGRAM STUDI S1 SOFTWARE ENGINEERING
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY
PURWOKERTO
2024

1. Tujuan

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

2. Landasan Teori

- **Abstract Data Type (ADT)**
  ADT adalah TYPE dan sekumpulan PRIMITIF (operasi dasar) terhadap TYPE tersebut. Selain itu, dalam sebuah ADT yang lengkap, disertakan pula definisi invarian dari TYPE dan aksioma yang berlaku. ADT merupakan definisi STATIK. Definisi type dari sebuah ADT dapat mengandung sebuah definisi ADT lain. Misalnya :
  1. ADT waktu yang terdiri dari ADT JAM dan ADT DATE
  2. Garis terdiri dari duah buah ADT POINT
  SEGI4 yang terdiri dari pasangan dua buah POINT (Top,Left) dan (Bottom,Right)

TYPE diterjemahkan menjadi type terdefinisi dalam bahasa yang bersangkutan. Jika dalam bahasa C menggunakan struct PRIMITIF, dalam konteks prosedural, diterjemahkan menjadi fungsi atau prosedur. PRIMITIF dikelompokan menjadi:

1. Konstruktor/Kreator, pemebentuk nilai type.Semua objek (variabel) bertype tersebut harus melalui konstruktor. Biasanya namanya diawali Make.
2. Selector, untuk mengakses tipe komponen(biasanya namanya diawali Get).
3. Prosedur pengubah nilai komponen (biasanya namanya diawali Get).
4. Tipe validator komponen, yang dipakai untuk mentest apakah dapat membentuk tipe sesuai dengan batasan.
5. Destruktor/Dealokator yaitu untuk "menghancurkan" nilai objek/variabel (sekaligus memori penyimpanannya).
6. Baca/Tulis, untuk interface dengan input/output device.
7. Operator relasional, terhadap tipe tersebut untuk mendefinisikan lebih besar, lebih kecil, sama dengan dan sebagainya.
8. Aritmatika terhadap tipe tersebut, karena biasanya aritmatika dalam bahasa C hanya terdefinisi untuk bilangan numerik.
9. Konversi dari tipe tersebut ke tipe dasar dan sebaliknya.

ADT biasanya diimplementasikan menjadi dua buah modul utama dan 1 modul interface program utama (driver). Dua modul tersebut adalah sebagai berikut:

1. Definisi/Spesifikasi Type dan Primitif/Header fungsi (.h)
   - Spesifikasi type sesuai dengan kaidah bahasa yang dipakai
   - Spesifikasi dari primitif sesuai dengan kaidah dalam konteks prosedural,yaitu:
   - Fungsi : nama, domain, range, dan prekondisi jika ada
   - Prosedur : Initial state, Final state, dan proses yang dilakukan
2. Body/realisasi dari primitif (.c)
   Berupa kode program dalam bahasa yang bersangkutan (dalam praktikum ini berarti dengan bahasa C++). Realisasi fungsi dan prosedur harus sedapat mungkin memanfaatkan selector dan konstruktor. Untuk memahami lebih jelas mengenai konsep ADT, perhatikan ilustrasi berikut.

ALGORITMA	C++

`Program coba_ADT`
`Type`
`  mahasiswa <`
`  nim : char[10]`
`  nilai1,nilai2 :`
`integer`
`Kamus`
`  mhs : mahasiswa`
`  procedure`
`inputMhs(input/output`
`  m : mahasiswa )`
`  function rata2(input:`
`  m : mahasiswa) : real`
 	
`Algoritma`
`  inputMhs(mhs)`
`  output( rata2(mhs) )`
 	
`procedure inputMhs(input/output`
`  m : mahasiswa )`
`kamus`
`algoritma`
	`#include <iostream>`
	`#include <conio.h>`
	`#include <stdlib.h>`
	
	`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) << endl;`
	`  return 0;`
	`}`

```
       input(m.nim, m.nilai1,
m.nilai2)

function rata2(input:
       m : mahasiswa) : real
kamus
algoritma
→ (m.nilai1 + m.nilai1) / 2
```

```
void inputMhs(Mahasiswa &m) {
    cout << "Input NIM = ";
    cin >> m.nim;
    cout << "Input Nilai 1 = ";
    cin >> m.nilai1;
    cout << "Input Nilai 2 = ";
    cin >> m.nilai2;
}

float rata2(Mahasiswa m) {
    return (m.nilai1 + m.nilai2) /
2.0;
}
```

Untuk menerapkan konsep ADT, kita harus memisah deklarasi tipe, variabel, dan fungsi dari program ke dalam sebuah file.h dan memisah definisi fungsi dari program ke sebuah file.cpp. Sehingga jika kita menerapkan konsep ADT berdasarkan contoh program di atas, bentuk code program akan dipisah menjadi seperti berikut.

ALGORITMA	C++

Program coba_ADT
 	
Type
mahasiswa <
nim : char[10]
nilai1, nilai2 : integer
 	
Kamus
mhs : mahasiswa
 	
procedure inputMhs(i/o
m : mahasiswa)
 	
function rata2(input:
m : mahasiswa) : real
 	
Algoritma
 	
inputMhs(mhs)
output(rata2(mhs))
 	
procedure inputMhs(input/output
m : mahasiswa)
Kamus
Algoritma
input(m.nim, m.nilai1,
m.nilai2)
 	
function rata2(input: m:
mahasiswa): real
kamus algoritma →(m.nilail + m.
nilail) / 2
	Mahaiswa.h
	
	#ifndef MAHASISWA_H_INCLUDED
	#define MAHASISWA_H_INCLUDED
	
	struct mahasiswa {
	char nim[10];
	int nilai1, nilai2;
	};
	
	void inputMhs(mahasiswa &m);
	float rata2(mahasiswa m);
	
	#endif // MAHASISWA_H_INCLUDED
	
	Mahasiswa.cpp
	
	void inputMhs(mahasiswa &m) {
	cout << "input nama = ";
	cin >> m.nim;
	cout << "input nilai = ";
	cin >> m.nilai1;
	cout << "input nilai2 = ";
	cin >> m.nilai2;
	}
	float rata2(mahasiswa m){
	return (m.nilai1+m.nilai2)/2;
	}
	
	main.cpp
	
	#include <iostream>
	#include <conio.h>
	#include <stdlib.h>
	#include "mahasiswa.cpp"
	
	using namespace std;
	
	int main() {
	mahasiswa mhs;
	
	inputMhs(mhs);
	
	cout << "Rata-rata nilai = " <<
	rata2(mhs) << endl;

	return 0; }

3. Unguided

1. Program Penyimpanan data Mahasiswa

Output:

```
Masukkan jumlah mahasiswa (maksimal 10): 1

Data Mahasiswa ke-1
Nama: Zhafir Zaidan Avail
NIM: 2311104059
Nilai UTS: 80
Nilai UAS: 85
Nilai Tugas: 89

Data Mahasiswa:
Nama: Zhafir Zaidan Avail
NIM: 2311104059
Nilai Akhir: 84.7


Process returned 0 (0x0)   execution time : 24.837 s
Press any key to continue.
```

2. Pembuatan ADT dan Implementasi ADT

ALGORITMA	C++

// Struktur Pelajaran
struct Pelajaran {
  string namaMapel;
  string kodeMapel;
  int sks;
  string mahasiswa[]; // Array of strings untuk menyimpan NIM mahasiswa
}
	pelajaran.h
	#ifndef PELAJARAN_H
	#define PELAJARAN_H
	
	#include <string>
	
	struct Pelajaran {
	  std::string namaMapel;
	  std::string kodeMapel;
	};
	
	Pelajaran create_pelajaran(std::string nama, std::string kode) {
	  Pelajaran pelajaran;
	  pelajaran.namaMapel = nama;
	  pelajaran.kodeMapel = kode;
	  return pelajaran;
	}

```
// Fungsi untuk membuat objek
Pelajaran
Pelajaran create_pelajaran(string
nama, string kode, int jumlahSKS) {
  Pelajaran pelajaran;
  pelajaran.namaMapel = nama;
  pelajaran.kodeMapel = kode;
  pelajaran.sks = jumlahSKS;
  // Inisialisasi array mahasiswa
dengan ukuran yang sesuai
  return pelajaran;
}
// Fungsi untuk menampilkan
informasi Pelajaran
void tampil_pelajaran(Pelajaran
pelajaran) {
  // Tampilkan namaMapel,
kodeMapel, sks, dan daftar
mahasiswa
}
// Fungsi untuk menambahkan
mahasiswa
void tambah_mahasiswa(Pelajaran
&pelajaran, string nim) {
  // Tambahkan nim ke dalam array
mahasiswa
}

// Fungsi untuk menghapus mahasiswa
void hapus_mahasiswa(Pelajaran
&pelajaran, string nim) {
  // Hapus nim dari array mahasiswa
}
```

```
void tampil_pelajaran(Pelajaran
pelajaran) {
    std::cout << "nama pelajaran: "
<< pelajaran.namaMapel << std::endl;
    std::cout << "nilai: " <<
pelajaran.kodeMapel << std::endl;
}

#endif
```

pelajaran.cpp

```
#include "pelajaran.h"
```

main.cpp

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

using namespace std;

int main() {
    string namaMapel = "Struktur
Data";
    string kodeMapel = "STD";

    Pelajaran pel =
create_pelajaran(namaMapel,
kodeMapel);
    tampil_pelajaran(pel);

    return 0;
}
```

Output:

```
nama pelajaran: Struktur Data
nilai: STD

Process returned 0 (0x0)   execution time : 0.048 s
Press any key to continue.
```

3. Array 2D, pointer, dan fungsi dalam C++ untuk memanipulasi data.

```
#include <iostream>

using namespace std;

void cetakArray(int arr[][3], int baris, int kolom) {
    for (int i = 0; i < baris; i++) {
        for (int j = 0; j < kolom; j++) {
            cout << arr[i][j] << " ";
        }
```

```
        cout << endl;
    }
}

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

int main() {
    int arr1[3][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
    int arr2[3][3] = {{10, 11, 12}, {13, 14, 15}, {16, 17, 18}};

    cout << "Array 1:\n";
    cetakArray(arr1, 3, 3);

    cout << "\nArray 2:\n";
    cetakArray(arr2, 3, 3);

    // Tukar elemen pada posisi (1,1) dari kedua array
    int *ptr1 = &arr1[1][1]; // Menunjuk ke elemen pada baris 1, kolom 1
di arr1
    int *ptr2 = &arr2[1][1]; // Menunjuk ke elemen pada baris 1, kolom 1
di arr2
    tukarElemen(ptr1, ptr2);

    cout << "\nSetelah ditukar:\n";
    cout << "Array 1:\n";
    cetakArray(arr1, 3, 3);

    cout << "\nArray 2:\n";
    cetakArray(arr2, 3, 3);

    return 0;
}
```

Output:

```
Array 1:
1 2 3
4 5 6
7 8 9

Array 2:
10 11 12
13 14 15
16 17 18

Setelah ditukar:
Array 1:
1 2 3
4 14 6
7 8 9

Array 2:
10 11 12
13 5 15
16 17 18

Process returned 0 (0x0)   execution time : 0.048 s
Press any key to continue.
```

4. Kesimpulan

Abstract Data Type (ADT) adalah sebuah konsep yang mendefinisikan tipe data dan operasi dasar (primitif) yang dapat dilakukan terhadap tipe tersebut. ADT fokus pada apa yang bisa dilakukan oleh data tanpa terikat pada cara implementasinya, sehingga memisahkan antara logika program dan implementasi. ADT juga digunakan untuk membuat program lebih modular dan mudah dikelola. Beberapa operasi dasar dalam ADT meliputi konstruktor (pembuat objek), selector (pengakses komponen), mutator (pengubah nilai), validator (pemeriksa validitas), destruktor (pembebas memori), serta operasi baca/tulis untuk berinteraksi dengan perangkat input dan output. Selain itu, ADT juga dapat menyediakan operator relasional dan aritmatika, serta konversi tipe agar lebih fleksibel dalam penggunaannya.

Implementasi ADT dalam C++ dilakukan dengan memisahkan deklarasi dan definisi fungsi serta tipe data ke dalam beberapa file terpisah. Biasanya, kode dibagi menjadi tiga bagian: header file (.h) yang berisi deklarasi struct dan fungsi, file implementasi (.cpp) yang memuat realisasi fungsi, dan file driver (main.cpp) yang berisi program utama untuk menguji fungsi dan operasi ADT. Misalnya, dalam contoh ADT mahasiswa, deklarasi struct mahasiswa dan fungsi seperti inputMhs dan rata2 diletakkan dalam header Mahasiswa.h. Definisi fungsi-fungsi tersebut dimasukkan dalam Mahasiswa.cpp. Program utama yang menggunakan dan menguji fungsi ADT ini ditulis dalam main.cpp.

Modularitas dan pemisahan kode berdasarkan ADT memiliki banyak manfaat. Dengan memisahkan kode menjadi beberapa modul, program menjadi lebih mudah dikembangkan dan dipelihara. Proses pemisahan ini juga mendukung konsep encapsulation, di mana implementasi dapat disembunyikan dari pengguna sehingga perubahan internal tidak memengaruhi program utama. Selain itu, modul yang telah dibuat bisa digunakan kembali di program lain tanpa perlu menulis ulang, meningkatkan reusability dan membuat program lebih scalable.

Penerapan ADT pada contoh program mahasiswa menunjukkan bagaimana tipe data dan operasi dasar bekerja bersama. Struct mahasiswa digunakan sebagai tipe data abstrak yang menyimpan NIM dan dua nilai. Prosedur inputMhs digunakan untuk mengisi data mahasiswa, dan fungsi rata2 menghitung rata-rata dua nilai tersebut. Dengan pembagian kode yang jelas ke dalam beberapa file, seperti Mahasiswa.h, Mahasiswa.cpp, dan main.cpp, program menjadi lebih terstruktur, mudah dibaca, dan siap untuk pengembangan lebih lanjut. Secara keseluruhan, konsep ADT membantu dalam pengembangan program yang lebih terorganisir, memudahkan pemeliharaan, serta memungkinkan modularitas dan pengembangan skala besar.