Modul 3. Abstract Data Type & Struct

Jump to bottom

Ahmad Faisal edited this page on May 2 · 5 revisions

Abstract Data Type & Struct

Abstact Data Type

Abstract Data Type, atau biasa disingkat ADT secara sederhana adalah tipe/objek dimana perilaku dari ADT tersebut didefinisikan oleh kumpulan nilai (value) dan kumpulan operasi. Abstract Data Type merupakan interface “abstrak” yang implementasinya disembunyikan dari user. Implementasi dari sebuah ADT bisa bermacam-macam, namun sebuah ADT selalu mempunyai maksud yang sama.

Contohnya adalah Integer. Integer merupakan ADT, yang mempunyai nilai-nilai $\ldots,-2,-1,0,1,2,\ldots$ dan operasi-operasi seperti penjumlahan, pengurangan, perkalian, pembagian dsb. Disini, user tidak perlu tahu bagaimana Integer diimplementasikan pada komputer. User hanya perlu tahu bahwa “there exist Integer”.

ADT vs Struktur Data

ADT merupakan ranah logikal (menyediakan interface) dan dapat diimplementasikan dengan berbagai cara. Struktur Data adalah implementasi konkret dari ADT.

Contoh Analogi

Bayangkan “Kendaraan” sebagai sebuah ADT. “Kendaraan” dapat diimplementasikan dengan berbagai bentuk, bisa mobil, kereta, bus,dll.

Contoh-contoh ADT yang sering dijumpai pada bahasa pemrograman.

Abstract Data Type	Implementasi Struktur Data
List	Dynamic Array/Linked List
Stack	Linked List/Dynamic Array
Queue	Linked List/Dynamic Array

Abstract Data Type	Implementasi Struktur Data
Bitset	Dynamic/Static Array
Priority Queue	Linked List
Set dan Map	Balanced Binary Search
(Unordered) Set dan Map	Balanced Binary Search Tree (AVL Tree)
Graph	Directed/Undirected Graph

Pengertian Struct (Structure)

Sebelumnya, kita telah mempelajari bahwa array memungkinkan kita untuk menyimpan beberapa nilai yang memiliki tipe data yang serupa di satu variable, sedangkan 'struct' adalah kumpulan dari beberapa variabel dengan beragam tipe data yang dibungkus dalam satu variabel. Dengan kata lain, `struct` merepresentasiskan sebuah record.

Mengapa dan kapan kita membutuhkan sebuah struct? Bayangkan ketika kita membuat list buku yang ada di perpustakaan, attribute yang digunakan adalah id_buku, judul dan tahun. Apabila tanpa menggunakan struct maka kita akan membuat seperti ini:

```
int id_buku1 = 1;
char judul1[] = "Laskar Pelangi";
int tahun1 = 2012;

int id_buku2 = 2;
char judul2[] = "Dilan";
int tahun2 = 2018;

int id_buku3 = 3;
char judul3[] = "5 CM";
int tahun3 = 2010;
```

Sangat boros variable bukan? Namun, dengan menggunakan `struct` kita dapat lebih mudah dalam membuat “template” buku dan digunakan untuk buku yang berbeda.

Membuat struct

Sebelum kita dapat membuat variabel struktur, kita perlu menentukan tipe datanya. Untuk mendefinisikan struct, kita harus menggunakan kata kunci struct diawal kemudian diikuti dengan nama struct dan isinya. Berikut ini adalah format struct:

```
struct namaStruktur {
  dataType member1;
  dataType member2;
```

```
  ...
};
```

Contoh penggunaannya,

```
struct Buku {
  int id_buku;
  char *judul;
  int tahun;
};
```

Cobalah kode dibawah ini lalu jalankan program tersebut dan lihat apa yang terjadi.

```
struct Buku {
   int id_buku;
   char  *judul;
   int tahun;
};

int main() {

   struct Buku buku1, buku2; //menggunakan tipe struct

   /* memasukkan nilai ke struct buku 1 */
   buku1.id_buku = 1;
   buku1.judul = "Laskar Pelangi";
   buku1.tahun = 2012;

      /* memasukkan nilai ke struct buku 2 */
   buku2.id_buku = 2;
   buku2.judul = "Dilan Dilan";
   buku2.tahun = 2018;


   /* print buku1 info */
   printf( "Buku 1 id : %d\n",buku1.id_buku);
   printf( "Buku 1 judul : %s\n", buku1.judul);
   printf( "Buku 1 tahun : %d\n", buku1.tahun);

      /* print buku2 info */
   printf( "Buku 2 id : %d\n",buku2.id_buku);
   printf( "Buku 2 judul : %s\n", buku2.judul);
   printf( "Buku 2 tahun : %d\n", buku2.tahun);

   return 0;
}
```

Ketika tipe struct dideklarasikan, tidak ada penyimpanan atau memori yang dialokasikan. Untuk mengakses struct, kita harus membuat variabelnya. Gunakan kata kunci struct di dalam metode `main()`, diikuti dengan nama struktur dan kemudian nama variabel struktur. Dari kode diatas, kita menggunakan pointer `*` untuk data string lalu untuk mengakses anggota struct menggunakan sintaks titik `(.)` .

Typedef

Kata kunci `typedef` digunakan untuk membuat nama alias untuk tipe data. Biasanya digunakan dengan struct untuk menyederhanakan sintaks dalam mendeklarasikan variabel. Kata kunci `typedef` berada di depan struct.

```
typedef struct Buku {
   int id_buku;
   char  *judul;
   int tahun;
} buku;
```

Cobalah kode dibawah ini lalu jalankan program tersebut dan temukan perbedaannya.

```
typedef struct Buku {
   int id_buku;
   char  *judul;
   int tahun;
} buku;

int main() {

   buku buku1, buku2; //menggunakan tipe struct

   /* memasukkan nilai ke struct buku 1 */
   buku1.id_buku = 1;
   buku1.judul = "Laskar Pelangi";
   buku1.tahun = 2012;

      /* memasukkan nilai ke struct buku 2 */
   buku2.id_buku = 2;
   buku2.judul = "Dilan Dilan";
   buku2.tahun = 2018;


   /* print buku1 info */
   printf( "Buku 1 id : %i\n",buku1.id_buku);
   printf( "Buku 1 judul : %s\n", buku1.judul);
   printf( "Buku 1 tahun : %i\n", buku1.tahun);

      /* print buku2 info */
   printf( "Buku 2 id : %d\n",buku2.id_buku);
```

```
   printf( "Buku 2 judul : %s\n", buku2.judul);
   printf( "Buku 2 tahun : %d\n", buku2.tahun);

   return 0;
}
```

Struct Bersarang

Bersarang artinya ada struct di dalam struct. Cobalah kode dibawah ini lalu jalankan program tersebut dan lihat apa yang terjadi.

```
struct Buku {
   int id;
   char  *judul;
   int tahun;
};

struct Perpustakaan {
    char *nama;
    char *alamat;

    struct Buku data_buku; //struct didalam struct
} data_perpustakaan;

int main( ) {

   struct Perpustakaan data_perpustakaan = {"Perpustakaan ITTS", "Ketintang No. 156", 001,
   printf( "Id Buku : %d\n",data_perpustakaan.data_buku.id);
   printf( "Judul Buku : %s\n",data_perpustakaan.data_buku.judul);
   printf( "Tahun Buku : %d\n",data_perpustakaan.data_buku.tahun);
   printf( "Nama Perpustakaan : %s\n",data_perpustakaan.nama);
   printf( "Alamat Perpustakaan : %s\n",data_perpustakaan.alamat);

   return 0;
}
```

Dari kode diatas, kita mengakses struct bersarang menggunakan dot `(.)` .

Array dalam Struct

Cobalah kode dibawah ini lalu jalankan program tersebut dan lihat apa yang terjadi.

```
#include <stdio.h>
#include <string.h>

struct Buku {
```

```
    char pengarang[50];
    char judul[30];
    int tahun;
};

int main( ) {

    struct Buku buku[3];
    for (int i=0;i<3;i++){
        printf("Pengisian Buku Ke-%i \n", i+1);
        printf("Masukkan nama pengarang : ");
        fflush(stdin);gets(buku[i].pengarang);
        printf("Masukkan judul          : ");
        fflush(stdin);gets(buku[i].judul);
        printf("Masukkan tahun : ");
        scanf("%i",&buku[i].tahun);
    }

    printf("\nData yang telah dimasukkan adalah : \n");
    printf("————————————————— \n");
    printf("| Pengarang | Judul | Tahun \n");
    printf("————————————————— \n");
    for (int i=0;i<3;i++){
        printf("| %s | %s | %i | \n", buku[i].pengarang, buku[i].judul, buku[i].tahun);
    }
    printf("————————————————— \n");

   return 0;
}
```

Struct dalam Fungsi

Cobalah kode dibawah ini lalu jalankan program tersebut dan lihat apa yang terjadi.

```
#include <stdio.h>
#include <string.h>

struct Buku {
   char pengarang[50];
   char judul[30];
   int tahun;
};

int main( ) {

    struct Buku buku;
    printf("Isi Buku Ini    : \n");
    printf("Masukkan nama pengarang : ");
    fflush(stdin);gets(buku.pengarang);
    printf("Masukkan judul          : ");
    fflush(stdin);gets(buku.judul);
```

```
    printf("Masukkan tahun : ");
    scanf("%i",&buku.tahun);
    tampil (buku);

   return 0;
}

//membuat fungsi dengan struct sebagai parameter
void tampil (struct Buku b){
    printf("\n\nData yang telah dimasukkan adalah pengarang buku %s, judul buku %s, tahun %

}
```

Pointer dalam Struct

Kita dapat mendefinisikan pointer ke struct dengan cara yang sama seperti saat kita mendefinisikan pointer ke variabel lainnya. Untuk menemukan alamat variabel struct, tempatkan `&` operator sebelum nama struktur. Untuk mengakses anggota struktur menggunakan penunjuk ke struktur tersebut, menggunakan operator → Cobalah kode dibawah ini lalu jalankan program tersebut dan lihat apa yang terjadi.

```
#include <stdio.h>
#include <string.h>

struct Buah{
   int berat;
   char namaBuah[10];
};

void cetak (struct Buah *b);

int main(){

    struct Buah buah;

    printf("Input: \n");
    printf("Masukkan Berat buah : ");
    scanf("%d", &buah.berat);
    printf("Masukkan Nama buah  : ");
    scanf("%[^\n]%*c", buah.namaBuah);

    cetak (&buah);
    return 0;
}

void cetak (struct Buah *b){
    printf("\nOutput: \n");
    printf("Berat buah : %dkg\n", b->berat);
```

```
    printf("Nama buah   : %s\n", b->namaBuah);
}
```

Self Referential Structures

Self Referential Structures adalah struktur yang memiliki setidaknya satu anggota pointer yang menunjuk ke struktur jenisnya sendiri.Jenis struct ini banyak digunakan pada struktur data dinamis seperti tree, linked list, dll. Cobalah kode dibawah ini lalu jalankan program tersebut dan lihat apa yang terjadi.

```
#include <stdio.h>

struct LinkedList {
        int data1;
        char data2;
        struct LinkedList* link;
};

int main()
{
        struct LinkedList l1, l2

        l1.link = NULL;
        l1.data1 = 10;
    l1.data2 = 20;

        l2.link = NULL;
        l2.data1 = 30;
        l2.data2 = 40;

        // Linking L1 dan L2
        l1.link = &l2;

        // Mengkakses data member l2 menggunakan l1
        printf("%d", l1.link->data1);
        printf("\n%d", l1.link->data2);
        return 0;
}
```

Pada kode diatas, penggunaan Self referential structure terlihat dari anggota struct LinkedList memiliki pointer ke jenisnya sendiri. Untuk mempelajari materi ini, akan dibahas lebih dalam pada pertemuan `Linked-List` .

Link Latihan Soal

🟣 Asisten Praktikum Algoritma Struktur Data Informatika 2023 🟣

▸ **Pages** 8

Clone this wiki locally

```
https://github.com/fzl-22/Algoritma-dan-Struktur-Data_Informatika.wiki.git
```