MODUL 3: STRUCTURE DAN POINTER

1.1. Deskripsi Singkat

Pada praktikum sebelumnya kita telah mehami dan mempraktikkan penggunaan tipe data dasar dan array dalam Bahasa Pemrograman C. Pada array, kita dapat menampung kumpulan data yang sejenis. Sebagai contoh, jika sebuah array kita deklarasikan bertipe integer, maka semua data yang ada di dalam array tersebut semuanya harus bertipe integer. Untuk menampung sejumlah data yang memiliki tipe data yang berbeda, kita dapat menggunakan structure. Pada praktikum kali ini, kita akan mencoba mehami dan mempraktikkan penggunaan structure dalam Bahasa Pemrograman C. Selain kita juga akan mencoba memahami dan mempraktikkan penggunaan variabel pointer.

1.2. Tujuan Praktikum

1) Mahasiswa mampu mendeklarasikan dan menggunakan struct dalam bahasa C.
2) Mahasiswa mampu mendeklarasikan dan menggunakan pointer dalam bahasa C.

1.3. Material Praktikum

Kegiatan pada modul ini memerlukan material berupa software editor dan compiler (atau IDE) untuk bahasa pemrograman C.

1.4. Kegiatan Praktikum

A. Structure

Structure (struktur) adalah kumpulan elemen-elemen data yang digabungkan menjadi satu kesatuan. Di dalam beberapa bahasa pemrograman, structure dikenal dengan nama **record**. Masing-masing elemen data tersebut dikenal juga dengan sebutan **field**. Elemen data tersebut dapat memiliki tipe data yang sama ataupun berbeda.

Mengapa kita membutuhkan structure? Misalnya kita ingin menyimpan data mahasiswa. Kita bisa saja melakukannya seperti ini:

```
char name[] = "Dian";
char address[] = "Mataram";
int age = 22;
```

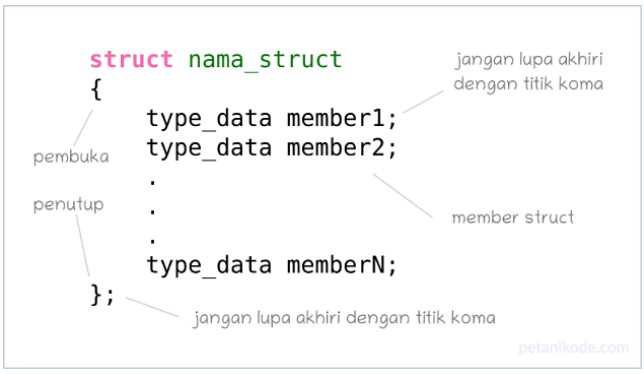
Lalu bagaimana kalau ada lebih dari satu mahasiswa? Mungkin bisa saja kita buat seperti ini:

```
char name[] = "Dian";
char address[] = "Mataram";
int age = 22;

char name2[] = "Bambang";
char address2[] = "Surabaya";
```

```
int age2 = 23;

char name3[] = "Bimo";
char address3[] = "Jakarta";
int age3 = 23;
```

Untuk tujuan kemudahan dalam operasinya, elemen-elemen tersebut akan lebih baik bila digabungkan menjadi satu, yaitu dengan menggunakan structure. Dengan kata lain, structure merupakan bentuk struktur data yang dapat menyimpan variabel dengan satu nama.

1. Pendeklarasian structure selalu diawali kata baku **struct**, diikuti nama structure dan deklarasi field-field yang membangun structure di antara pasangan tanda kurung kurawal " { } ". Berikut adalah bentuk umum dan contohnya:




```
struct mahasiswa
{
    char nim[11];
    char nama[30];
    char lamat[50];
    float ipk;
};
```

Agar structure dapat digunakan, kita harus membuatkan variabel untuknya. Variabel structure dapat dideklarasikan bersamaan dengan deklarasi structure atau sebagai deklarasi terpisah seperti mendeklarasikan variabel dengan tipe data dasar. Cobalah untuk mendeklarasikan structure seperti contoh di bawah ini, lalu simpan dengan nama **praktikum3_1a.c.** Pastikan tidak ada error saat program dijalankan.

```
struct data_tanggal
{
 int tahun;
 int bulan;
 int tanggal;
} today;
```

Potongan program yang baru saja dijalankan adalah contoh deklarasi structure **(data_tanggal)** bersamaan dengan deklarasi variabel structure **(today)**. Variabel structure yang dideklarasikan secara terpisah dapat dilakukan dengan cara berikut.

```
struct data_tanggal
{
 int tahun;
 int bulan;
```

```
 int tanggal;
};
struct data_tanggal today;
```

Ketik ulang potongan program di atas lalu simpan dengan nama **praktikum3_1b.c**. Pastikan tidak ada error yang muncul saat program dijalankan.

Kita juga dapat mendeklarasikan structure di luar fungsi **main()**, dan variabel structurenya dideklarasikan di dalam fungsi **main()**, seperti yang ditunjukkan pada potongan program di bawah ini. Ketik ulang potongan program tersebut, lalu simpan dengan nama **praktikum3_1c.c.** Pastikan tidak ada error yang muncul saat program dijalankan.

```
struct data_tanggal
{
 int tahun;
 int bulan;
 int tanggal;
};

int main()
{
   struct data_tanggal today;
   return 0;
}
```

2. Kita juga dapat menggunakan **typedef** pada structure. Kata kunci **typedef** adalah kata kunci untuk mendefinisikan tipe data baru. Kita bisa menggunakan kata kunci ini di depan **struct** untuk menyatakannya sebagai tipe data baru. Sebagai contoh, pertama-tama ketik ulang potongan program di bawah ini lalu simpan dengan nama **praktikum3_2.c.**

```
// membuat struct
struct Distance{
    int feet;
    float inch;
};

void main() {
    // menggunakan struct
```

```
    struct Distance d1, d2;
}
```

Tanpa **typedef**, kita akan menggunakan struct dengan cara seperti itu. Jika menggunakan **typedef**, maka penggunaan struct akan menjadi seperti ini:

```
// membuat struct dengan typedef
typedef struct Distance{
    int feet;
    float inch;
} distances;

void main() {
    // menggunakan struct
    distances dist1, dist2, sum;
}
```

Perhatikan bedanya, lalu modifikasi potongan program yang telah Anda simpan pada file **praktikum3_2.c.** Simpan modifikasi yang Anda buat, lalu jalankan **praktikum3_2.c.** Pastikan tidak ada error yang muncul.

3. Setelah berhasil mendeklarasikan structure, kita dapat menginisialiasi nilai-nilai elemen yang terdapat di dalamya. Anggota struktur dapat diinisialisasi dengan menggunakan kurung kurawal '**{ }**'. Kita akan mempraktikkan beberapa cara dalam menginisialisasi elemen structure, menggunakan file-file praktikum pada poin 1.
   Cara yang pertama untuk variabel structure yang dideklarasikan bersamaan dengan deklarasi structure. Pada **praktikum3_1a.c**, baris terakhir program dapat dimodifikasi menjadi seperti berikut ini:

```
} today = {1998,7,24};
```

Simpan **praktikum3_1a.c** yang telah ditambahkan potongan program di atas, lalu coba jalankan.

Cara yang kedua, bila variabel structure dideklarasikan secara terpisah dari deklarasi structure, maka inisialisasi nilai-nilai elemennya dapat dilakukan seperti ini:

```
struct data_tanggal today = {1998,7,28};
```

Modifikasi potongan program **praktikum3_1b.c** dan **praktikum3_1c.c** mengikuti cara inisialisasi yang kedua ini. Pastikan tidak ada error saat program dijalankan kembali.

4. Walaupun elemen-elemen di dalam structure berada dalam satu kesatuan, masing-masing elemen tersebut tetap dapat diakses secara individual. Untuk mengakses elemen-elemen pada structure, gunakan operator titik **(.)**. Berikut ini adalah contoh dalam

mengakses elemen-elemen pada suatu struct. Ketik ulang potongan program di bawah ini, modifikasi bagian-bagian yang ditandai dengan komentar.

```
#include<stdio.h>

struct data_tanggal
{
 int tahun;
 int bulan;
 int tanggal;
}ultah;

int main()
{
   //menginisialisasi elemen-elemen struct ultah
   ultah.tanggal = 28; //ganti dengan tanggal lahir Anda
   ultah.bulan = 7;   //ganti dengan bulan lahir Anda
   ultah.tahun = 1998; //ganti dengan tahun lahir Anda

  //mengakses elemen-elemen struct ultah
   printf ("tanggal = %d, bulan = %d, tahun = %d"
           ,ultah.tanggal, ultah.bulan, ultah.tahun);
   return 0;
}
```

Simpan program dengan nama **praktikum3_4.c**, lalu jalankan. Bila tidak ada error dan program berhasil dijalankan, artinya kita sudah bisa menginisialisasi sekaligus mengakses elemen-elemen yang terdapat pada structure. Anda juga dapat menambahkan elemen-elemen lain pada structure yang terdapat pada **praktikum3_4.c.**

Berikut ini adalah contoh-contoh program menggunakan structure. Silakan Anda ketik ulang pada IDE Anda, lalu cobalah untuk memodifikasi elemen-elemen yang terdapat pada structure pada contoh-contoh program tersebut.

5. Pertama, contoh program yang menerapkan konsep struktur untuk buku yang terdiri dari elemen: judul, pengarang dan id, simpan dengan nama **praktikum3_5.c.**

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

```
struct Buku {
   char  judul[50];
   char  pengarang[50];
   int   id;
};

int main( ) {
   struct Buku Buku1;
   struct Buku Buku2;

   /* Spesifikasi Buku 1 */
   strcpy( Buku1.judul, "C Programming");
   strcpy( Buku1.pengarang, "Nuha Ali");
   Buku1.id = 6495407;

   /* Spesifikasi Buku 2 */
   strcpy( Buku2.judul, "Telecom Billing");
   strcpy( Buku2.pengarang, "Zara Ali");
   Buku2.id = 6495700;

   /* Cetak informasi Buku 1 */
   printf( "Judul Buku 1  : %s\n", Buku1.judul);
   printf( "Pengarang Buku 1  : %s\n", Buku1.pengarang);
   printf( "Id Buku 1 : %d\n\n", Buku1.id);

    /* Cetak informasi Buku 2 */
   printf( "Judul Buku 2  : %s\n", Buku2.judul);
   printf( "Pengarang Buku 2  : %s\n", Buku2.pengarang);
   printf( "Id Buku 2 : %d\n", Buku2.id);

   return 0;
}
```

Jalankan program tersebut. Apa output yang didapatkan?

6. Contoh berikutnya adalah Struct Bersarang. Struct dapat dibuat bersarang (*nested*), yang artinya ada struct di dalam struct.

```
struct complex
{
 int imag;
 float real;
};

struct number
{
   struct complex comp;
   int integers;
} num1, num2;
```

Cara menggunanya adalah seperti ini:

```
num1.integers = 12;
num1.comp.real = 44.12;
num2.comp.imag = 11;
```

Satukan kedua potongan program di atas dan simpan dengan nama **praktikum3_6.c.** Modifikasi **praktikum3_6.c** dengan menambahkan potongan program untuk menampilkan nilai setiap elemen ke layar. Lalu coba jalankan.

7. Structure juga dapat kita buat sebagai parameter untuk fungsi. Contoh:

```
#include <stdio.h>
struct student
{
    char name[50];
    int age;
};

void main() {
    struct student s1;
    printf("Enter name: ");
    gets(s1.name);
    printf("Enter age: ");
    scanf("%d", &s1.age);
    display(s1);   // passing structure as an argument
```

```
}

// membuat fungsi dengan struct sebagai parameter
void display(struct student s) {
  printf("\nDisplaying information\n");
  printf("Name: %s", s.name);
  printf("\nRoll: %d", s.age);
}
```

Ketik ulang potongan program tersebut, lalu simpan dengan nama **praktikum3_7.c.** Jalankan program tersebut. Apa output yang didapatkan?

B. Pointer

Setiap variabel yang kita buat pada program akan memiliki alamat memori. Alamat memori berfungsi untuk menentukan lokasi penyimpanan data pada memori (RAM). Kadang alamat memori ini disebut *reference* atau referensi. Untuk melihat alamat memori yang digunakan pada variabel, kita bisa pakai simbol **&** (referensi). Tak perlu jauh-jauh, kita sudah tahu tentang fungsi **`scanf()`**, yaitu fungsi yang meminta input dari pengguna. Contoh:

```
scanf("%d", &a);
```

Pada contoh tersebut, terdapat **'&a'** yang artinya program akan menyimpan nilai input ke dalam alamat memori **a**. Alamat memori memiliki format heksadesimal (hex), yaitu sistem bilangan yang memiliki 16 simbol dalam penomoran. Contoh untuk mencetak alamat dari memori **a**:

```
#include <stdio.h>
int main(){
    int a;
    printf("%p\n", &a);
}
```

Hasil keluaran (Hasil bisa berbeda setiap saat dan berbeda antara perangkat satu dengan lainnya):

```
0x7ffef6dec9ec
```

Lalu apa hubungannya alamat memori dengan pointer? Pointer adalah sebuah **variabel yang berisi alamat memori** dari variabel yang lain. Setelah mengetahui cara mengambil alamat sebuah variabel, sekarang kita harus menggunakan pointer untuk menyimpannya. Pointer juga bisa mengakses data yang ada di suatu alamat memori.

8. Meskipun berbeda dari variabel umumnya, namun cara mendeklarasikan pointer hampir sama dengan yang lain. Deklarasi pointer:

```
tipe_data *nama_variabel;
```

Karena pointer menampung alamat memori, maka dalam melakukan inisialisasi juga harus memberikan alamat memori. Untuk memahami deklarasi dan inisialisasi pointer, ketik ulang potongan program di bawah ini.

```
#include <stdio.h>
int main(){
    int a = 5;
    int *p;
    p = &a;
    printf("%p\n", p);
}
```

Simpan program dengan nama **praktikum3_8.c.** Setelah program dijalankan, apakah output yang didapatkan? Pada contoh tersebut, saat memberikan nilai pada variabel pointer **p**, tak perlu menambahkan *****. Kemudian, untuk mencetaknya tak perlu menggunakan &, karena **p** merupakan variabel pointer.

Operator dereferensi (*****) merupakan kebalikan dari operator referensi (**&**) karena operator ini akan menunjuk nilai yang berada di alamat memori. Modifikasi baris sebelum kurung penutup pada **praktikum3_8.c**, menjadi seperti berikut:

```
printf("%d\n", *p);
```

Simpan ulang program, kemudian jalankan. Maka output yang didapatkan akan berbeda dengan sebelumnya. **p** merupakan pointer yang menyimpan alamat dari **a**, sehingga ***p** berisi nilai dari **a**.

Agar lebih mengerti mengenai penggunaan pointer, cobalah untuk menjalankan dan memodifikasi contoh-contoh program yang menggunakan pointer berikut ini.

9. Buatlah file baru dengan nama **praktikum3_9.c**, kemudian isi dengan kode berikut:

```
#include <stdio.h>

void main(){
    // membuat variabel
    int umur = 19;
    float tinggi = 175.6;
```

```
    // membuat pointer
    int *pointer_umur = &umur;
    int *pointer2 = &umur;
    float *p_tinggi = &tinggi;


    // mecetak alamat memori variabel
    printf("alamat memori variabel 'umur' = %d\n", &umur);
    printf("alamat memori variabel 'tinggi' = %d\n",
&tinggi);
    // mencetak referensi alamat memori pointer
    printf("referensi alamat memori *pointer_umur = %d\n",
pointer_umur);
    printf("referensi alamat memori *pointer2 = %d\n",
pointer2);
    printf("referensi alamat memori *p_tinggi = %d\n",
p_tinggi);
}
```

Setelah itu, jalankan program. Perhatikan hasilnya. Alamat memori yang digunakan sebagai referensi pada pointer akan sama dengan alamat memori dari variabel yang kita gunakan sebagai referensi.

10. Pointer juga memiliki alamat memorinya sendiri. Sama seperti variabel biasa, jika ingin melihat alamat memori dari pointer, maka kita harus menggunakan simbol **&**. Sebagai contoh, kita masih menggunakan program yang tersimpan pada **praktikum3_9.c** Untuk melihat alamat pointer-pointer pada program tersebut, tambahkan potongan program di bawah ini:

```
// mencetak alamat memori pointer
    printf("alamat memori *pointer_umur = %d\n",
&pointer_umur);
    printf("alamat memori *pointer2 = %d\n", &pointer2);
    printf("alamat memori *p_tinggi = %d\n", &p_tinggi);
```

Jalankan kembali **praktikum3_9.c**. Maka akan kita dapatkan alamat memori dari pointer-pointer yang ada pada program tersebut.

11. Berikutnya kita akan mencoba menggunakan pointer untuk melakukan passing argumen berdasaran referensinya (*pass by reference*). Pertama-tama ketik ulang program di bawah ini, lalu simpan dengan nama **praktikum3_11.c.**

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

```
void add_score(int score){
    score = score + 5;
}

void main(){
    int score = 0;

    printf("score sebelum diubah: %d\n", score);
    add_score(score);
    printf("score setelah diubah: %d\n", score);
}
```

Pada program ini, kita membuat fungsi dengan nama **add_score()** untuk menambahkan nilai score sebanyak 5. Tapi ketika dijalankan, nilai variabel score tidak berubah, ia tetap bernilai 0. Hal ini karena variabel **score** dibuat di dalam fungsi **main()**, lalu ketika fungsi **add_score()** mencoba mengubah nilainya, maka perubahan hanya terjadi secara lokal di dalam fungsi **add_score()** saja. Hal ini dapat dibuktikan dengan menambahkan potongan program berikut ke dalam fungsi **add_score()**:

```
printf("Score dibuah ke %d\n", score);
```

Jalankan ulang **praktikum3_11.c** Dapat dilihat bahwa nilai **score** pada fungsi **add_score()** sudah berubah menjadi 5, namun variabel **score** pada fungsi **main()** akan tetap bernilai 0. Permasalah ini salah satunya dapat diselesaikan dengan menggunakan pointer untuk melakukan *pass-by-reference*.

Sekarang, coba ubah kode programnya menjadi seperti ini:

```
#include <stdio.h>

void add_score(int *score){
    *score = *score + 5;
    printf("score dibuah ke: %d\n", *score);
}

void main(){
    int score = 0;

    printf("score sebelum diubah: %d\n", score);
    add_score(&score);
    printf("score setelah diubah: %d\n", score);
```

```
}
```

Karena argumen fungsi **add_score()** kita ubah menjadi pointer, maka kita harus memberikan alamat memori saat memanggilnya. Sekarang, setiap fungsi **add_score()** dipanggil atau dieksekusi, maka nilai variabel score akan bertambah 5.

12. Pointer juga sering digunakan untuk mengakses elemen array, seperti yang ditunjukkan pada program di bawah ini. Ketik ulang program tersebut, lalu simpan dengan nama **praktikum3_12.c**.

```
#include <stdio.h>

void main(){
    printf("## Program Antrian CS ##\n");

    char no_antrian[5] = {'A', 'B', 'C', 'D', 'E'};

    // menggunakan pointer
    char *ptr_current = &no_antrian;

    for(int i = 0; i < 5; i++){
        printf("📢 Pelanggan dengan no antrian %c silahkan
ke loket!\n", *ptr_current);
        printf("Saat ini CS sedang melayani: %c\n",
*ptr_current);
        printf("-------- Tekan Enter untuk Next --------");
        getchar();
        ptr_current++;
    }

    printf("✅ Selesai");
}
```

Pada program ini, kita menggunakan **ptr_current** untuk mengakses elemen array. Saat pertama kali dibuat, pointer **ptr_current** akan mereferensi pada elemen pertama array. Lalu pada perulangan, dilakukan increment **ptr_current++**, maka pointer ini akan mereferensi ke elemen array sealnjutnya. Untuk memahami hal ini, jalankan program **praktikum3_12.c**, dan pelajari hasil yang dikeluarkan oleh program.

1.5. Penugasan

1. Buatlah contoh program sederhana yang memuat elemen array di dalam structure DAN sekaligus memuat structure di dalam array.

2. Buatlah contoh program sederhana yang memuat sebuah structure dengan elemen pointer di dalamnya.

3. Buatlah contoh program sederhana yang memuat sebuah pointer yang menunjuk sebuah structure.