MODUL 2: TIPE DATA C, STRUCTURE

1.1. Deskripsi Singkat

Secara umum, struktur data dikelompokkan menjadi dua kategori, yaitu struktur data primitif (sederhana) dan struktur data non primitif (majemuk). Struktur data primitif adalah tipe data dasar *(primitive data types)* yang didukung oleh bahasa pemrograman (*built-in*), misalnya Integer, floating point, characters, pointer, dan-lain-lain. Struktur data non primitif misalnya arrays, structure, stack, queues, linked lists, dan lain-lain. Struktur data non-primitif sendiri diturunkan dari struktur data primitif. Oleh karena itu, pemahaman mengenai tipe data sangat penting untuk memahami dan mengimplementasikan struktur data. Dalam praktikum kita ini akan memahami tipe data dasar dalam Bahasa Pemrograman C serta mempraktekkan penggunaannya. Selain itu, kita akan mencoba memahami dan mempraktikkan *structure* atau struktur. Structure digunakan untuk menampung sejumlah data yang memiliki tipe data yang berbeda.

1.2. Tujuan Praktikum

Setelah praktikum pada modul 2 ini diharapkan mahasiswa mampu:

1) Memahami struktur data primitif (tipe data dasar) dalam Bahasa Pemrograman C
2) Mendeklarasikan deklarasi data, serta mengoperasikan dan menampilkannya dalam format yang sesuai
3) Mahasiswa mampu mendeklarasikan dan menggunakan structure dalam bahasa C.

1.3. Alokasi Waktu

- Tatap muka di kelas (wajib): 50 menit
- Kerja Mandiri: 120 menit pengerjaan penugasan

1.4. Material Praktikum

Kegiatan pada modul 2 ini memerlukan material berupa software editor dan compiler (atau IDE) untuk bahasa pemrograman C.

1.5. Kegiatan Praktikum

A. Tipe Data

Dalam bahasa C terdapat beberapa jenis tipe data yang bisa digunakan untuk mendeklarasikan sebuah variabel, konstansta, atau fungsi pada program. Jenis variabel menentukan berapa banyak ruang yang ditempati dalam penyimpanan dan bagaimana pola bit yang disimpan ditafsirkan. Variabel biasa digunakan di dalam program dengan tujuan

untuk menampung data. Nilai yang terdapat pada variabel sewaktu-waktu dapat diubah. Jumlah variabel yang dibuat dapat tidak terbatas, namun masing-masing variabel tersebut harus bersifat unik dan tidak boleh ada nama variabel yang sama. Agar dapat digunakan, sebelumnya variabel dideklarasikan dahulu berserta tipe data yang akan disimpan dalam variabel tersebut. variabel dideklarasikan dengan format:

```
tipe_data nama_variabel
```

Adapun aturan-aturan penamaan variabel dalam bahasa C adalah sebagai berikut :

1. Harus diawali dengan huruf (tidak boleh diawali dengan angka), selanjutnya dapat diikuti dengan angka atau underscore (_).
2. Tidak boleh mengandung spasi, simbol, atau karakter khusus lain selain huruf, angka, dan underscore.
3. Bersifat case sensitive.
4. Tidak boleh memakai kata kunci (keywords) yang telah digunakan oleh Bahasa C.

Selain variabel terdapat konstanta yang juga dapat menampung data. Hanya saja dalam kostanta nilai yang ada tidak dapat diubah atau bernilai pasti. Konstanta dideklarasikan dengan menggunakan preprocessor **#define**:

```
#define nama_konstanta
```

atau dengan dengan singkatan **const:**

```
const tipe_data dan nama_konstanta.
```

Dalam pembuatan fungsi, tipe data digunakan untuk mendeklarasikan tipe data kembalian dari fungsi. Berikut format dasar cara penulisan fungsi dalam bahasa C:

```
tipeDataKembalian namaFunction() {
  // Isi function disini...
  // Isi function disini...
  return nilai;
}
```

Jika suatu fungsi tidak mengembalikan nilai, **tipeDataKembalian** ditulis sebagai **void**. Sebuah fungsi yang tidak mengembalikan nilai terkadang disebut juga sebagai **procedure**.

C menyediakan lima macam tipe data dasar, yaitu:

- tipe data integer (nilai numerik bulat, yang dideklarasikan dengan **int**)
- floating-point (nilai numerik pecahan ketetapan tunggal, yang dideklarasikan dengan **float**)
- double-precision (nilai numerik pecahan ketetapan ganda, yang dideklarasikan dengan **double**)

- karakter (dideklarasikan dengan **char**)
- **void** (tidak bertipe, tidak menyimpan apapun, biasanya untuk tipe fungsi yang tidak return value apapun)

1. a. Berikut cara mendeklarasikan variable menggunakan **int**. Ketik ulang kode program berikut ke dalam Code::Blocks atau IDE C lainnya yang Anda gunakan.

```
#include <stdio.h>
int main() {
    int tanggal = 17;
    printf("Tanggal %i", tanggal);
    return 0;
}
```

Simpan potongan program tersebut dengan nama **praktikum2A.c**. Jalankan program tersebut. Output yang didapatkan adalah sebagai berikut:

```
Tanggal 17
```

b. Kita juga bisa mendeklarasikan dua variable sekaligus jika tipe datanya sama. Modifikasi program **praktikum2A.c** menjadi seperti berikut ini.

```
#include <stdio.h>
int main() {
    int tanggal = 17;
    int bulan = 8;
    printf("Tanggal %i, Bulan %d", tanggal, bulan);
    return 0;
}
```

Simpan ulang dan jalankan program. Hasilnya adalah sebagai berikut:

```
Tanggal 17, Bulan 8
```

c. Bisa kita perhatikan, pada variabel tanggal kita menggunakan format specifier **%i** sedangkan pada variabel bulan kita menggunakan format specifier **%d**. Pada **printf()** tidak ada perbedaan antara keduanya. Namun ketika menggunakan **scanf()** dan menginput nilai dengan 0 didepan angka (011, 012, 013), maka akan terlihat perbedaannya. Pada **%d** diasumsikan sebagai basis 10 (desimal) sedangkan **%i** diasumsikan sebagai basis 8 (oktal). Silakan dicoba, dengan mengetikkan ulang potongan program di bawah ini, lalu simpan dengan nama **praktikum2B.c**.

```
#include <stdio.h>

int main()
{
    int a, b;
    printf("Angka pertama: ");
    scanf("%d", &a);
    printf("Angka kedua: ");
    scanf("%i", &b);
    printf("\nNilai desimal dari angka pertama adalah: %i",
a);
    printf("\nNilai oktal dari angka kedua adalah: %i", b);

    return 0;
}
```

Jika kita berikan **011** sebagai input, maka ouput yang didapatkan adalah sebagai berikut:

```
Angka pertama: 011
Angka kedua: 011

Nilai desimal dari angka pertama adalah: 11
Nilai oktal dari angka kedua adalah: 9
```

2. Jika kita menggunakan **float** atau **double** untuk bilangan desimal, double akan dua kali lebih lebih teliti daripada float. Float memiliki ketepatan 7 digit desimal, sedangkan double memiliki ketepatan hingga 15 digit desimal. Contoh kode penggunaan kedua tipe data ini adalah sebagai berikut:

```
#include <stdio.h>

void pifloat()
{
   float pi = 22.0/7.0;
   printf("=== FLOAT ===\n");
   printf("%.15lf\n", pi);
   printf("%lf\n", pi);
}
```

```
void pidouble()
{
   double pi = 22.0/7.0;
   printf("=== DOUBLE ===\n");
   printf("%.15lf\n", pi);
   printf("%lf\n", pi);
}

int main()
{
   pifloat();
   printf("\n");
   pidouble();

   return 0;
}
```

Ketik ulang program tersebut, lalu simpan dengan nama **praktikum2C.c**. Bila dijalankan, outputnya adalah:

```
=== FLOAT ===
3.142857074737549
3.142857

=== DOUBLE ===
3.142857142857143
3.142857
```

Bisa kita lihat, ada perbedaan antara penggunaan float dan double pada desimal ke-tujuh.

3. Keyword **char** digunakan untuk mendeklarasikan tipe data yang memuat nilai berupa karakter, contohnya adalah sebagai berikut:

```
#include <stdio.h>

int main() {
    char huruf = 'Y';
```

```
    printf("%c", huruf);

return 0;
}
```

Ketik ulang program di atas, lalu simpan dengan nama **praktikum2D.c.** Output:

```
Y
```

Dengan menggunakan kode diatas hanya akan mengeluarkan satu huruf saja. Untuk menyimpan kalimat atau **string**, maka harus kita ubah terlebih dahulu kodenya. Simpan potongan program di bawah ini dengan nama **praktikum2E.c**.

```
#include <stdio.h>

int main() {
    char kalimat[10] = "AnbiDev";
    printf("%s", kalimat);

return 0;
}
```

Output yang dihasilkan:

```
AnbiDev
```

Terlihat perbedaan pada kode pada **praktikum2D.c** dan **praktikum2E.c**. Pada **praktikum2D.c** kita menggunakan tanda petik satu untuk nilainya dan menggunakan format specifier **%c**. Pada **praktikumE.c**, kita menggunakan tanda petik dua dan menggunakan format specifier **%s** sebagai ganti **%c**.

B. Typedef

Pada bahasa C, kita dapat memberikan sebuah nama baru pada tipe data. Keyword **typedef** biasanya digunakan untuk membuat nama alias dari tipe data.

Bentuk umum:

```
typedef <bentuk asal> <nama type>;
```

Contoh:

```
typedef int bulat;

typedef struct {
    double r, theta;
}Kompleks;
```

4. Sebagai contoh penggunaan typedef, simpan program di bawah ini dengan nama **praktikum2F.c**, lalu coba jalankan.

```
#include <stdio.h>
typedef unsigned char BYTE;
int main()
{
  BYTE b1, b2;
  b1 = 'c';
  printf("%c ", b1);
  return 0;
}
```

Pada program tersebut, dapat dilihat bahwa **unsigned char** selanjutnya dapat digunakan/dipanggil dengan nama **BYTE**. Jalankan program **praktikum2F.c**, apakah output yang dihasilkan?

C. 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;
```

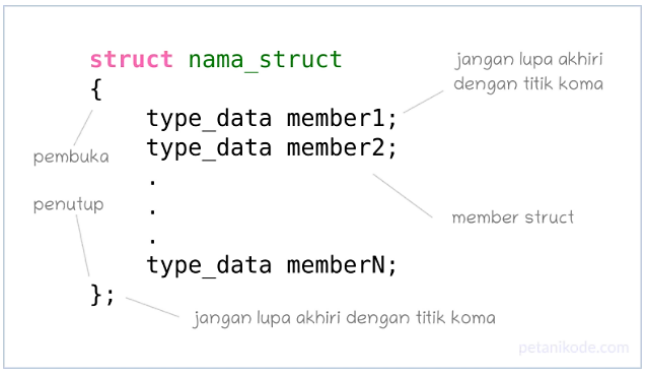
Lalu bagaimana kalau ada lebih dari satu mahasiswa? Mungkin bisa saja kita buat seperti ini:

```
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 **praktikum2G.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 **praktikum2H.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 **praktikum2Ic.** 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 **praktikum2J.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 **praktikum2J.c.** Simpan modifikasi yang Anda buat, lalu jalankan **praktikum2J.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 **praktikum2G.c**, baris terakhir program dapat dimodifikasi menjadi seperti berikut ini:

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

Simpan **praktikum2G.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 **praktikum2H.c** dan **praktikum2I.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 **praktikum2K.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 **praktikum2K.c.**

5. 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 **praktikum2L.c.** Jalankan program tersebut. Apa output yang didapatkan?

1.6. Penugasan

1. **`sizeof()`** adalah sebuah operator untuk mengetahui jumlah memori (byte) yang diperlukan oleh suatu tipe data pada bahasa C. Gunakan **`sizeof()`** untuk mengetahui ukuran memori pada berbagai tipe data pada bahasa pemrograman C, seperti **char, int, float, double.** Catat hasil yang Anda dapatkan. Bandingkan dengan hasil yang didapatkan oleh teman-teman Anda. Diskusikan apakah yang menyebabkan hasil yang didapatkan berbeda-beda.

2. Ketik program berikut pada IDE Anda, lalu simpan.

```
/* Aturan Scope pada Bahasa C */
#include<stdio.h>

int main()
{
  {
    int x = 10, y = 20;
    {
       printf("x = %d, y = %d\n", x, y);
```

```
        {
           int y = 40;
           x++;
           y++;
           printf("x = %d, y = %d\n", x, y);
        }
        printf("x = %d, y = %d\n", x, y);
     }
   }
 return 0;
}
```

Amati output yang dihasilkan. Jelaskan mengapa bisa terjadi perubahan demikian pada nilai `x` maupun `y`.

3. 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 menggunakannya adalah seperti ini:

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

Satukan kedua potongan program di atas dan simpan, kemudian modifikasi dengan menambahkan potongan program untuk menampilkan nilai setiap elemen ke layar. Lalu coba jalankan. Simpan program dengan nama **praktikum2L.c**

4. Modifikasi potongan program yang telah Anda simpan pada file **praktikum2J.c** (program yang menggunakan typedef pada structure). Tambahkan potongan program untuk

menginisialiasi nilai-nilai elemen yang terdapat di dalam structure dan menampikan nilai tersebut ke layar.

Gabungkan file-file **praktikum2J.c** dan **praktikum2L.c** menggunakan zip/rar dan beri nama dengan format **Praktikum2_kelas_noabsen.rar**. Unggah file zip/rar pada Google Classroom sesuai dengan batas waktu yang telah ditetapkan.