`struct`, `union`, `enum`, DAN `typedef`

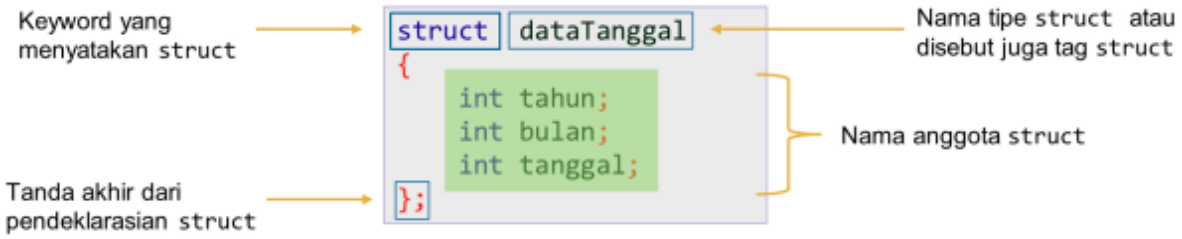
OBJEKTIF :

1. Mahasiswa mampu memahami tentang `struct`.
2. Mahasiswa mampu memahami tentang `union`.
3. Mahasiswa mampu memahami tentang `enum`.
4. Mahasiswa mampu memahami tentang `typedef`.

6.1 `struct`

6.1.1 PENGERTIAN `struct`

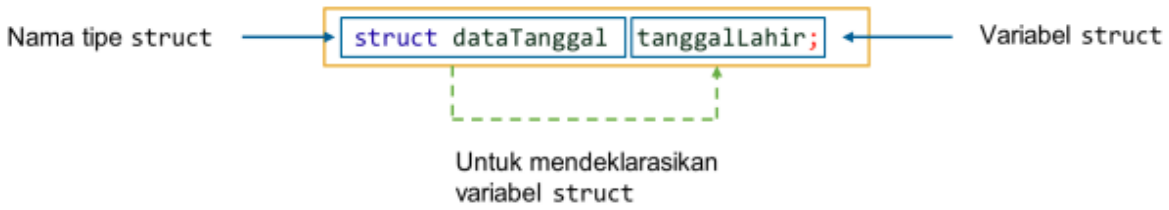
`struct` bermanfaat untuk mengelompokkan sejumlah data dengan tipe yang berlainan. `struct` juga dapat mengelompokkan data dengan tipe sama. Di bawah ini merupakan contoh dari pendeklarasian `struct`:



Dimana `struct` merupakan *keyword* yang menyatakan `struct`, kemudian `dataTanggal` merupakan nama tipe `struct` atau disebut juga tag `struct`. Kemudian pada blok `struct` terdapat tiga nama anggota `struct` dimana kita dapat mendeklarasikan variabel beserta dengan tipe datanya. Lalu diakhiri dengan semicolon sebagai akhir dari pendeklarasian `struct`.

6.1.2 PENDEKLARASIAN `struct`

Apabila suatu `struct` telah diciptakan, `struct` ini dapat digunakan untuk mendeklarasikan suatu variabel. Di bawah ini merupakan contoh dari pendeklarasian variabel dengan menggunakan `struct`:



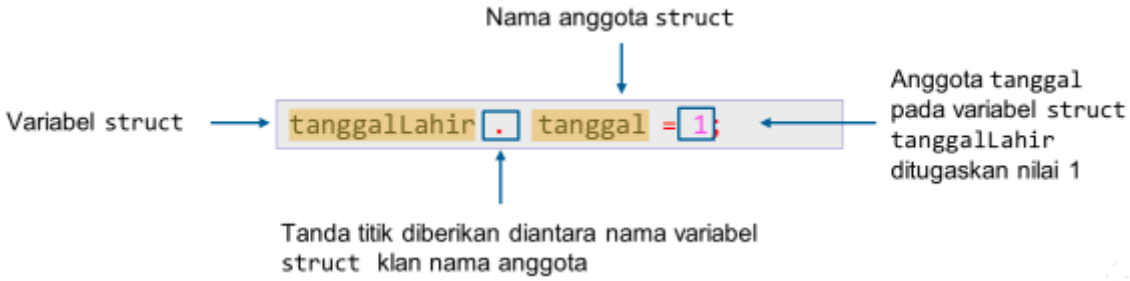
Dimana `struct` `dataTanggal` merupakan nama tipe `struct` dan `tanggalLahir` merupakan variabel `struct`. Sehingga untuk mendeklarasikan variabel `struct`, perlu menuliskan nama tipe `struct` terlebih dahulu dan diikuti dengan variabel `struct`.

6.1.3 PENGAKSESAN `struct`

Anggota `struct` diakses dengan menggunakan bentuk umum seperti berikut:

```
variabel_struct.nama_anggota;
```

Sebagai contoh perhatikan cara pengaksesan dari `struct` berikut:



Di bawah ini merupakan program pengaksesan `struct`:

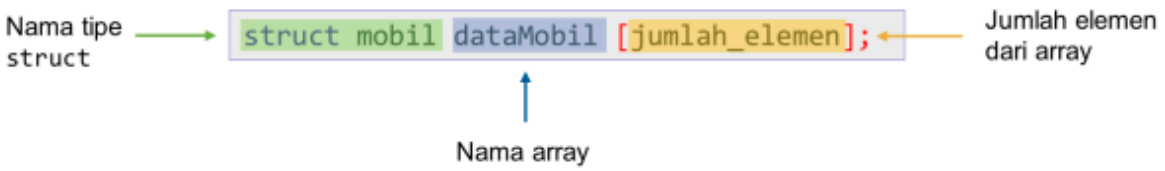
```
#include <stdio.h>

int main()
{
     struct  dataTanggal
    {
       int tahun;
       int bulan;
       int tanggal;
    };

    struct dataTanggal tanggalLahir;

    tanggalLahir.tanggal = 1;
    tanggalLahir.bulan = 9;
    tanggalLahir.tahun = 1964;


    printf("%d/%d/%d\n", tanggalLahir.tanggal, tanggalLahir.bulan,
tanggalLahir.tahun);

    return 0;
}
```

Output program:

```
1/9/1964
```

6.1.4 ARRAY DAN `struct`

`struct` dapat mengelompokkan data dengan tipe data sama atau berbeda. Beda halnya dengan `struct`, array hanya dapat mengelompokkan data dengan tipe data sama. Namun, array dapat dideklarasikan dengan tipe `struct`. Berikut adalah contoh dari pendeklarasian array `struct`:



Dimana `struct` mobil merupakan nama tipe `struct` yang diikuti dengan `dataMobil` yang merupakan nama array. Seperti jika menuliskan array, pada array `struct` juga dapat dituliskan jumlah elemen dari array.
Di bawah ini merupakan program array `struct`:

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

// Deklarasi struct
struct mahasiswa
{
    char nama[10];
    char npm[10];
    float ipk;
};

int main()
{
    int i;
    // Array struct
    struct mahasiswa dataMahasiswa[5];

    printf("Masukkan data dari 5 Mahasiswa");

    for(i = 0; i < 5; i++)
    {

        printf("\nMasukkan nama : ");
        scanf("%s", &dataMahasiswa[i].nama);
        printf("Masukkan npm : ");
        scanf("%s", &dataMahasiswa[i].npm);
        printf("Masukkan ipk : ");
        scanf("%f", &dataMahasiswa[i].ipk);
    }

    printf("\nData Mahasiswa");
    printf("\n============================");

    for(i=0; i < 5; i++)
    {
        printf("\nNama:%s\t NPM:%s\t IPK:%.2f", dataMahasiswa[i].nama,
                                                dataMahasiswa[i].npm,
```

```
                                                dataMahasiswa[i].ipk);
    }

    return 0;
}
```

Output program:

```
Masukkan nama : Budi
Masukkan NPM : 1001
Masukkan IPK : 3.9

Masukkan nama : Lala
Masukkan NPM : 1002
Masukkan IPK : 3.5

Masukkan nama : Dian
Masukkan NPM : 1003
Masukkan IPK : 3.7

Masukkan nama : Lulu
Masukkan NPM : 1004
Masukkan IPK : 4

Masukkan nama : Siti
Masukkan NPM : 1005
Masukkan IPK : 3.6

Data Mahasiswa
==============
Nama:Budi       NPM:1001        IPK:3.90
Nama:Lala       NPM:1002        IPK:3.50
Nama:Dian       NPM:1003        IPK:3.70
Nama:Lulu       NPM:1004        IPK:4.00
Nama:Siti       NPM:1005        IPK:3.60
```

6.2 `union`

6.2.1 PENGERTIAN `union`

`union` adalah suatu metode pengelompokkan nilai yang menyerupai `struct` namun mempunyai fungsi yang berbeda. Di dalam `union`, kita dapat mendeklarasikan lebih dari satu nama variabel. Tipe data dari nama variabel yang di simpan di dalam `union` dapat berbeda. Di bawah ini merupakan bentuk umum deklarasi `union`:

Dimana untuk mendeklarasikan `union` maka keyword yang digunakan adalah `union`. namaUnion merupakan nama tipe `union` dimana bagian tersebut tidak dapat dipanggil secara langsung. Untuk dapat memanggil nama tipe `union` diperlukan variabel `union`. Bagian elemen `union` akan diisi oleh beberapa variabel baik yang memiliki tipe data yang sama maupun yang berbeda. Bagian variabel `union` digunakan sebagai perantara yang digunakan untuk mengakses suatu `union`.

6.2.2 PERBEDAAN `struct` DAN `union`

Lokasi a dan b BEDA

```c
#include <stdio.h>

int main()
{
  struct angkaStruct
  {
     int a;
     int b;
  };

  struct angkaStruct namaStruct;

  printf("Alamat dari variabel a adalah:
          %p\n", namaStruct.a);
  printf("Alamat dari variabel b adalah:
          %p\n", namaStruct.b);

  return 0;
}
```

Output:

```
Alamat dari variabel a adalah: 0032D000
Alamat dari variabel b adalah: 00400080
```

Lokasi a dan b SAMA

```c
#include <stdio.h>

int main()
{
  union angkaUnion
  {
     int a;
     int b;
  };

  union angkaUnion namaUnion;

  printf("Alamat dari variabel a adalah:
          %p\n", namaUnion.a);
  printf("Alamat dari variabel b adalah:
          %p\n", namaUnion.b);

  return 0;
}
```

Output:

```
Alamat dari variabel a adalah: 00400080
Alamat dari variabel b adalah: 00400080
```

Pada output program yang menggunakan `struct`, dapat dilihat bahwa setiap variabel menempati lokasi memori yang berbeda. Sedangkan output program yang menggunakan `union`, dapat dilihat bahwa setiap variabel menempati lokasi memori yang sama. Hal ini dikarenakan konsep `union` adalah berbagi memori antar variabel.
Di bawah ini merupakan program `union`:

```c
#include <stdio.h>

int main()
{
  union bilBulat
  {
     unsigned int bilG;
     unsigned char bilC[2];
  };

  union bilBulat bilX;
  bilX.bilG = 0x1234;

  printf("bilG    : %x\n", bilX.bilG);
  printf("bilC[0] : %x\n", bilX.bilC[0]);
  printf("bilC[1] : %x\n", bilX.bilC[1]);

  return 0;

}
```

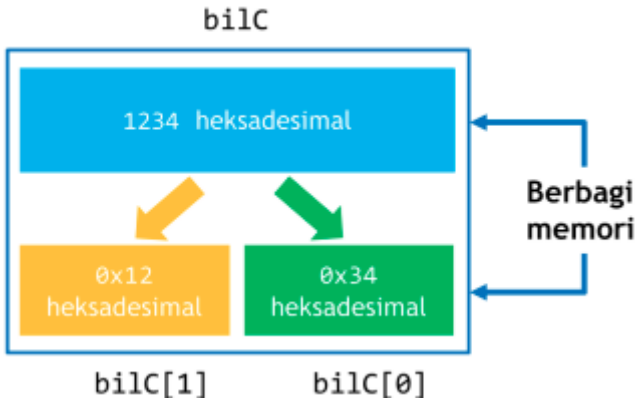
Output program:

```
bilG    : 1234
bilC[0] : 34
bilC[1] : 12
```

Keterangan:

- Output `1234` tercetak disebabkan oleh format `%x`, karena `0x1234` merupakan bilangan dengan format heksadesimal
- Berbeda dengan `array`, pembacaan nilai dari indeks untuk bilangan heksadesimal dimulai dari bilangan yang berada di belakang, maka dari itu nilai dari `bilC[0]` adalah 34

Di bawah ini merupakan gambaran pembagian memori berdasarkan program di atas:



Pertama deklarasikan sebuah variabel `bilC` dimana pada variabel tersebut ditugaskan untuk menampung variabel yang sama dengan variabel `bilG` yaitu 1234. Variabel `bilC` dibagi dua yaitu pada indeks pertama menampung nilai 34 lalu pada indeks kedua menampung nilai 12. Dikarenakan konsep `union` adalah berbagi memori, nilai variabel yang satu akan mempengaruhi nilai variabel lainnya yang ada di dalam `union`.

6.3 `enum`

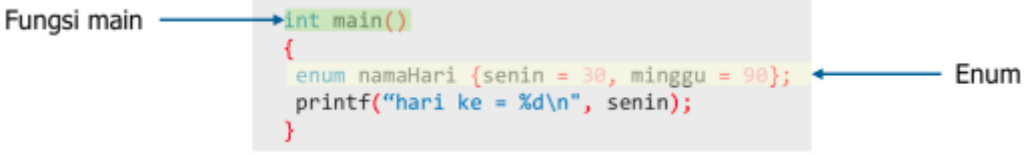
6.3.1 PENGERTIAN `enum`

`enum` merupakan tipe data yang didefinisikan oleh *programmer* yang akan menugaskan data menjadi konstanta `integer`. `enum` hanya dapat digunakan jika kemungkinan nilai suatu data telah diketahui dan jumlahnya sedikit, misal jenis kelamin dan nama hari.
Kelebihan `enum`:

- Tipe data `enum` dapat dideklarasikan secara variabel lokal
- Tipe data `enum` akan melakukan inisialisasi secara otomatis

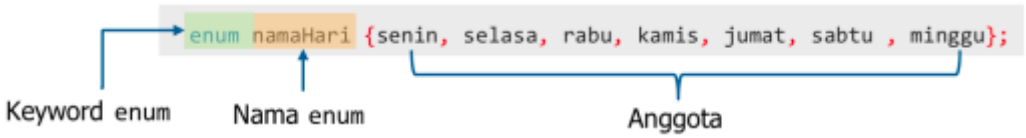
6.3.2 PENDEFINISIAN `enum`

6.3.2.1 Pendefinisian `enum` Pada Variabel Lokal



Saat mendefinisikan `enum` dalam variabel lokal, maka `enum` tersebut hanya dapat dikenali oleh fungsi tempat `enum` tersebut didefinisikan.

6.3.2.2 Pendefinisian `enum` Secara Otomatis



Pada saat mendefinisikan `enum`, jika tidak menugaskan nilai pada anggota `enum`, maka akan ditugaskan secara otomatis mulai dari nol, seperti pada contoh di atas.

6.3.3 ATURAN PADA `enum`

Di bawah ini merupakan aturan - aturan yang perlu diperhatikan pada saat pendefinisian `enum` agar tidak terjadi error:

- Jika pada sebuah fungsi terdapat lebih dari satu `enum`, maka setiap anggota yang terdapat di dalamnya harus unik, seperti:

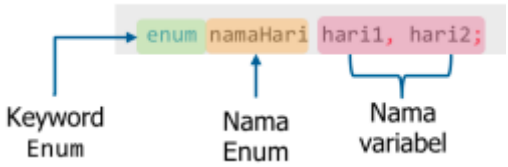
```
int main()
{
 enum hari1 {senin, selasa, rabu};
 enum hari2 {kamis, jumat, sabtu, minggu};
 printf("hari ke = %d\n", minggu);
}
```

- Saat menugaskan nilai pada anggota yang terdapat dalam `enum`, nilai tersebut harus bertipe data integer, seperti:

```
int main()
{
 enum hari1 {senin = 2, selasa = 10};
 printf("hari ke = %d\n", senin);
}
```

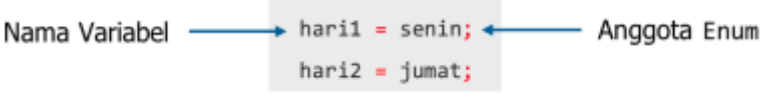
6.3.4 DEKLARASI DAN INISIALISASI `enum`

6.3.4.1 Deklarasi `enum`



Dalam menggunakan `enum` perlu dilakukan deklarasi tipe data `enum` ke variabel. Pertama digunakan *keyword* `enum` yang diikuti dengan nama `enum` yang telah didefinisikan dan nama variabel yang diinginkan.

6.3.4.2 Inisialisasi `enum`



Pertama dituliskan nama variabelnya lalu kemudian terdapat simbol *assignment* yang diikuti dengan anggota `enum`.

Di bawah ini merupakan program `enum`:

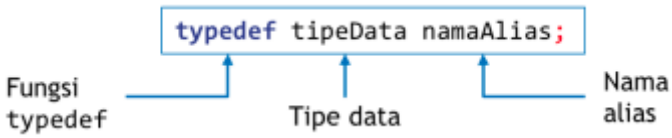
```
#include <stdio.h>
int main()
{
 enum namaHari {senin, selasa, rabu,
        kamis, jumat, sabtu, minggu};

    enum namaHari hari1, hari2;

    hari1 = senin;
    hari2 = sabtu;

    int selisih;
    selisih = hari2 - hari1;

    printf("selisih hari = %d\n", selisih);

    return 0;

}
```

Output program:

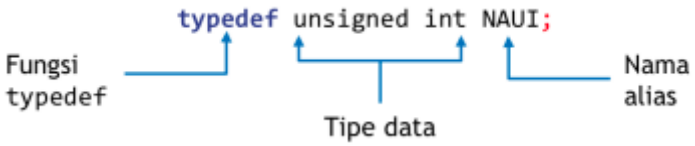
```
selisih hari = 5
```

6.4 `typedef`

`typedef` dapat digunakan untuk memberi nama alias pada suatu tipe data. Contohnya, kita ingin mendeklarasikan suatu variabel dengan tipe data `unsigned int`. Namun, kita dapat memberikan nama alias yang lebih singkat dan mudah diingat untuk tipe data `unsigned int` tersebut. Di bawah merupakan bentuk umum deklarasi `typedef`:



Dimana `typedef` merupakan *keyword* yang menyatakan fungsi `typedef`, kemudian diikuti dengan tipeData yang merupakan tipe data awalan atau tipe data asli. Selanjutnya terdapat namaAlias yang merupakan nama alias dari sebuah tipe data asli.
Di bawah ini merupakan contoh penggunaan `typedef`:



Dengan deklarasi seperti di atas, itu artinya kita memberikan nama alias untuk tipe data `unsigned int` dengan nama alias `NAUI`

Di bawah ini merupakan program `typedef`:

```c
#include <stdio.h>

int main()
{
    typedef unsigned int NAUI;

    NAUI angka = 25;

    printf("Isi dari variabel angka adalah : %d", angka);

    return 0;

}
```

Output program:

```
Isi dari variabel angka adalah: 25
```

REFERENSI

[1] Abdul Kadir. 2015. *From Zero to a Pro*. Yogyakarta. Andi