BAB X
DATA TINGKAT LANJUT

Tujuan :

1. Menjelaskan tentang tipe data union
2. Menjelaskan penggunaan bitfield
3. Menjelaskan tentang tipe data enumerasi
4. Menjelaskan penggunaan typedef
5. Menjelaskan penggunaan ternary operator
6. Menjelaskan tentang konversi tipe data *(type casting)*

10.1 Union

Pada C, union memungkinkan suatu lokasi memori ditempati oleh dua atau lebih variabel yang bisa saja tipenya berlainan. Di bawah ini diberikan contoh pendefinisian tipe union yang dinamakan sebagai **bil_bulat**, yang digunakan untuk menyatakan data bertipe karakter atau integer.

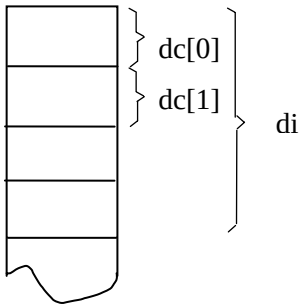
```
union bil_bulat {
     unsigned int di;
     unsigned char dc[2];
};
```

Berikutnya, pendeklarasian suatu variabel union bernama **bil_x** yang bertipe **bil_bulat** dilakukan dengan cara penulisan sebagai berikut

```
union bil_bulat bil_x;
```

Catatan : Cara lain untuk mendefinisikan atau mendeklarasikan union adalah seperti pada struktur.

Gambar 10.1 memperlihatkan **di** dan **dc** berbagi tempat pada lokasi yang sama (**di** dan **dc[0]** mempunyai alamat yang sama)



Gambar 10.1 Variabel bil_x yang bertipe union bil_bulat dalam memori

Dalam hal ini, kompiler dengan bijaksana akan menyediakan ruangan yang cukup untuk menampung *field* atau elemen pada union yang membutuhkan memori paling besar. Pada pendeklarasian variabel **bil_x** misalnya, memori yang ditempati variabel ini adalah 4 byte (yaitu ukuran dari tipe *int*).

Elemen dari sebuah union dapat diakses dalam bentuk sebagai berikut :

variabel_union.nama_elemen

misal : `bil_x.di = 321;`

adalah contoh untuk mengisikan 321 ke elemen union bernama di. Kalau dituliskan angka biner dari 321 = 101000001. Dengan pengisian nilai ini, maka **`dc[0]`** akan bernilai byte ke-0 dari **`di`**, sedangkan **`dc[1]`** bernilai byte ke-1 dari **`di`.**

```
/* File program : union1.c
Contoh pendeklarasian dan pengaksesan variabel union */

#include <stdio.h>

main()
{
     union
     {
          unsigned int di;
          unsigned char dc[2];
     } bil_x;  /* variabel union */

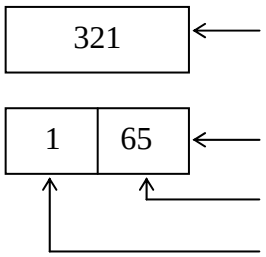
     bil_x.di = 321;
```

```
     printf("di = %d\n”, bil_x.di);
     printf(“dc[0] = %d dc[1] = %d\n", bil_x.dc[0],
          bil_x.dc[1]);
}
```

Contoh eksekusi :

```
di = 321
dc[0] = 65    dc[1] = 1
```

Program di atas menjelaskan cara ntuk mengakses byte ke-0 atau byte ke-1 dari **`di`**, **`dc[0]`** atau **`dc[1]`** yang digunakan.



Nilai 321 dalam kesatuan *unsigned int*

Nilai 321 jika dinyatakan dalam dua buah *unsigned char*

Byte rendah (byte ke-0)

Byte rendah (byte ke-0)

Seperti halnya pada struktur, suatu variabel union dapat dilewatkan ke dalam suatu fungsi sebagai parameter. Di bawah ini contoh program yang memberikan gambaran tentang cara mengubah isi suatu variabel union melalui pemanggilan suatu fungsi. Dalam hal ini, yang dilewatkan ke dalam fungsi berupa alamat dari variabel union.

```
/* File program : union2.c
Contoh untuk mengubah nilai variabel union melalui fungsi */

#include <stdio.h>

union bil_bulat{
     unsigned int di;
     unsigned char dc[2];          /* definisi tipe union */
};

void beri_nilai(union bil_bulat *x);    /*prototype fungsi */
```

```
main()
{
     union bil_bulat bil_x;   /* deklarasi var union */

     /* melewatkan alamat union */
     beri_nilai(&bil_x);
     printf("di = %d\n”, bil_x.di);
     printf(“dc[0] = %d  dc[1] = %d \n",bil_x.dc[0],
          bil_x.dc[1]);
}


void beri_nilai(union bil_bulat *x)
{
     x -> di = 321;           /* elemen di yang ditunjuk */
}                             /* oleh x diberi nilai 321 */
```

Contoh eksekusi :

```
di = 321
dc[0] = 65     dc[1] = 1
```

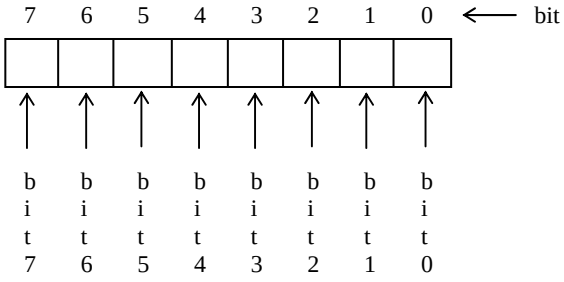
10.2 Bitfield

Suatu bit atau beberapa bit dalam sebuah data berukuran satu byte atau dua byte dapat diakses dengan mudah melalui *bitfield*. Dengan cara ini, suatu bit atau beberapa bit dapat diakses tanpa melibatkan operator pemanipulasi bit (seperti &, |). Selain itu, satu atau dua byte memori dapat dipakai untuk menyimpan sejumlah informasi.

Sebagai contoh, untuk memperoleh informasi masing-masing bit dari suatu data satu byte, penulisan medan bit berupa

```
struct info_byte
  { unsigned bit0:1;
    unsigned bit1:1;
    unsigned bit2:1;
    unsigned bit3:1;
    unsigned bit4:1;
    unsigned bit5:1;
    unsigned bit6:1;
    unsigned bit7:1;
  };
```

Jika disajikan dalam bentuk gambar, gambaran suatu struktur yang memiliki tipe seperti di atas adalah sebagai berikut :



Gambar 10.2 Susunan bit dari memori sebuah data bertipe **info_byte**

Pada pendefinisian struktur **info_byte** di atas,

- Nilai 1 setelah tanda titik-dua (**:**) menyatakan panjang dari *bitfield*
- unsigned menyatakan bahwa *bitfield* dinyatakan dalam keadaan tak-bertanda (untuk contoh berikutnya, nantinya setiap *bitfield* memiliki kemungkinan nilai berkisar 1 atau 0).

nama tipe struktur yang terdiri atas sejumlah *bitfield*

```
struct info_byte
  { unsigned bit0:1;
    unsigned bit1:1;
    unsigned bit2:1;
    unsigned bit3:1;
    unsigned bit4:1;
    unsigned bit5:1;
    unsigned bit6:1;
    unsigned bit7:1;
  };
```

panjang/jumlah bit

nama variabel *bitfield*

Catatan : sebuah variabel *bitfield* haruslah dideklarasikan berupa salah satu di antara int, *unsigned* dan *signed*

Contoh berikut memberikan gambaran tentang cara memberikan nilai kepada variabel struktur yang mengandung elemen berupa *bitfield*, dan cara mengakses setiap nilai dari *bitfield*.

```
/* File program : bitf1.c
Bitfield utk menampilkan bentuk biner dr karakter masukan */

#include <stdio.h>

main()
{
     struct info_byte    /* definisi tipe bitfield */
     {
          unsigned bit0:1;         /* bit ke-0 */
          unsigned bit1:1;         /* bit ke-1 */
          unsigned bit2:1;         /* bit ke-2 */
          unsigned bit3:1;         /* bit ke-3 */
          unsigned bit4:1;         /* bit ke-4 */
          unsigned bit5:1;         /* bit ke-5 */
          unsigned bit6:1;         /* bit ke-6 */
          unsigned bit7:1;         /* bit ke-7 */
     };
     /* deklarasi variabel union dan elemen bitfield */
     union
     {
          unsigned char karakter;
          struct info_byte byte;
     } ascii;

     printf(“Masukkan sebuah karakter : ”);
     scanf(“%c”, &ascii.karakter);

     printf(“\nKode ASCII dari karakter %c adalah %d\n”,
          ascii.karakter, ascii.karakter);
     printf("Bentuk biner dari nilai %d adalah ",
          ascii.karakter);
     printf("%d%d%d%d%d%d%d%d\n",ascii.byte.bit7,
          ascii.byte.bit6, ascii.byte.bit5,  ascii.byte.bit4,
          ascii.byte.bit3, ascii.byte.bit2, ascii.byte.bit1,
          ascii.byte.bit0);
}
```

Contoh eksekusi :

```
Masukkan sebuah karakter : A
Kode ASCII karakter A adalah 65
Bentuk biner dari nilai 65 adalah 01000001
```

Pada program di atas, setelah pernyataan :

```
     scanf(“%c”, &ascii.karakter);
```

dan user memasukkan karakter : ‘A’ , berarti nilai ascii.karakter = ‘A’. Maka hal itu memberikan efek elemen byte juga akan bernilai seperti karakter, sebab byte dan karakter berbagi data pada memori yang sama. Namun, walaupun adanya sifat demikian, pernyataan :

```
ascii.byte = 'A';
```

akan dianggap salah (saat kompilasi), sebab suatu variabel struktur yang mengandung elemen *bitfield* memang tidak diijinkan untuk diberi nilai secara langsung. Pengaksesan nilai dapat dilakukan melalui variabel *bitfield*, misalnya :

```
printf("%d", ascii.byte.bit7);
```

untuk mengambil nilai dari *bitfield* **bit 7**. Contoh lain

```
ascii.byte.bit7 = 0;
```

untuk mengubah **bit7** agar bernilai 0.

Kalau di depan sudah dibicarakan *bitfield* dengan panjang 1 bit, contoh berikut akan memberikan gambaran tentang *bitfield* dengan panjang 2 bit.

```
struct data_gambar
{
     unsigned piksel1:2;
     unsigned piksel2:2;
     unsigned piksel3:2;
     unsigned piksel4:2;
} koord;
```

Pada contoh di atas, variabel koord yang bertipe data_gambar akan menempati memori 1 byte (8 bit) dengan 4 informasi terkandung di dalamnya (masing-masing 2 bit), atau memegang nilai bulat antara 0 sampai dengan 3 ($2^2 – 1$).

Untuk memberikan nilai kepada piksel1 misalnya, bisa digunakan pernyataan sebagai berikut :

```
koord.piksel1 = 3;
```

yang mengisikan 3 ke dalam *bitfield* tersebut.

Bitfield biasanya dipakai untuk menghemat memori. Misalnya ada dua informasi dengan keterangan sebagai berikut :

- informasi pertama (**info_x**) memiliki kemungkinan nilai bilangan bulat antara 0 sampai dengan 3, dan
- informasi kedua (**info_y**) memiliki kemungkinan nilai bilangan bulat 0 atau 1 saja.

Seandainya kedua informasi itu disimpan dalam memori (secara terpisah) sebagai tipe char, maka akan diperlukan total memori sebesar 2 byte. Namun jika disajikan dalam bentuk *bitfield*, memori yang dibutuhkan cukup 1 byte. Dalam hal ini **info_x** akan dinyatakan dalam 2 bit dan **info_y** dinyatakan dalam 1 bit. Penuangan deklarasinya adalah sebagai berikut :

```
struct info
{
     unsigned info_x:2;
     unsigned info_y:1;
} status;
```

atau

```
struct info
{
     unsigned info_x:2;
     unsigned info_y:1;
     unsigned       :5;
} status;
```

Pada pendeklarasian terakhir :

```
     unsigned       :5;
```

fungsinya hanya untuk memperjelas bahwa total bit dari *bitfield* adalah 8 bit (1 byte). Perhatikan, bahwa karena 5 bit terakhir tidak diperlukan, maka nama *bitfield* boleh tidak disertakan. Kalaupun mau diberi nama (misalnya : **kosong**), maka bentuk deklarasinya adalah :

```
struct info
{
     unsigned info_x:2;
     unsigned info_y:1;
     unsigned kosong:5;
} status;
```

```
/* File program : BITF2.C */
#include <stdio.h>

main()
{
     /* definisi tipe bitfield */
     struct info
     {
          unsigned info_x:2;
          unsigned info_y:1;
          unsigned kosong:5;  /* bisa dihilangkan */
     } status;

     status.info_x = 3;
     status.info_y = 1;

     printf("info_x = %d\n", status.info_x);
     printf("info_y = %d\n", status.info_y);
}
```

Contoh eksekusi :

```
info_x = 3
info_y =  1
```

10.3 Enumerasi

Tipe enumerasi merupakan himpunan dari konstanta integer yang diberi nama. Contoh enumerasi yaitu berupa jenis kelamin manusia yang berupa

```
pria, wanita
```

Dalam C, suatu tipe data enumerasi dideklarasikan dengan bentuk :

enum nama_tipe_enumerasi {
konstanta_1, konstanta_2,…
} variabel_1, …, variabel_n;

Sedangkan contoh deklarasi variabel enumerasi :

```
enum manusia jns_kelamin;
```

Pada contoh di atas, **jns_kelamin** adalah variabel enumerasi yang bertipe manusia. Selanjutnya variabel **jns_kelamin** dapat diisi dengan konstanta pria dan wanita.

```
/* File program : enum1.c
Contoh penggunaan enumerasi */

#include <stdio.h>

main()
{
     enum manusia {                    /* definsi tipe */
          pria, wanita
          };

     enum manusia jns_kelamin;         /* deklarasi var */

     jns_kelamin = pria;               /* diisi dgn pria */
     printf("Isi jns_kelamin = %d\n", jns_kelamin);

     jns_kelamin = wanita;          /* diisi dgn wanita */
     printf("Isi jns_kelamin = %d\n", jns_kelamin);
}
```

Contoh eksekusi :

```
Isi jns_kelamin = 0
Isi jns_kelamin = 1
```

Dengan adanya pendefinisan seperti :

```
enum manusia {pria, wanita};
```

degnan sendirinya pria merupakan konstanta dengan nilai sama dengan 0, sedangkan wanita bernilai 1. Sehingga pernyataan

```
jns_kelamin = pria;
```

merupakan pernyataan untuk mengisikan konstanta pria (atau nilai 0) ke variabel **jns_kelamin**. Contoh lain, yaitu ;

```
enum data_hari {senin, selasa, rabu, kamis, jumat,
    sabtu, minggu};
```

Pada pendefinisian di atas,

`senin` menyatakan nilai `0`

`selasa` menyatakan nilai `1`

`rabu` menyatakan nilai `3`

`kamis` menyatakan nilai `4`

`jumat` menyatakan nilai `5`

`sabtu` menyatakan nilai `6`

`minggu` menyatakan nilai `7`

Pemakaian enumerasi biasanya untuk memperjelas dokumentasi program C, seperti yang ditunjukkan dalam contoh program di bawah ini.

```
/* File program : enum2.c
Contoh pemakaian enumerasi */

#include <stdio.h>

main()
{
     /* definisi tipe data enumerasi */
     enum data_hari {senin, selasa, rabu, kamis, jumat,
          sabtu, minggu};
     /* keterangan nama hari */
     static char str[][7] = {"SENIN", "SELASA", "RABU",
          "KAMIS", "JUMAT", "SABTU", "MINGGU"};

     /* deklarasi variabel enumerasi */
     enum data_hari hari_kerja;

     int jam_kerja;
     int total_jam = 0;

     /* cetak nama hari dari senin s/d jumat */
     for(hari_kerja=senin; hari_kerja<=jumat; hari_kerja++)
     {
          printf ("Jumlah jam kerja hari ");
          printf("%-6s (jam) : ", str[hari_kerja]);
          scanf("%d", &jam_kerja);
          total_jam = total_jam + jam_kerja;
     }
     printf("\nTotal jam kerja = %d\n", total_jam);
}
```

Contoh eksekusi :

```
Jumlah jam kerja hari SENIN  (jam) : 8
Jumlah jam kerja hari SELASA (jam) : 8
Jumlah jam kerja hari RABU   (jam) : 8
Jumlah jam kerja hari KAMIS  (jam) : 8
Jumlah jam kerja hari JUMAT  (jam) : 6

Total jam kerja = 38
```

Penggalan pernyataan berikut

```
for(hari_kerja=senin; hari_kerja<=jumat; hari_kerja++);
```

lebih memberi kejelasan daripada penulisan :

```
for(hari_kerja=0; hari_kerja<=5; hari_kerja++);
```

Jika dikehendaki, nilai urutan sebuah enumerasi juga bisa dirubah (yang secara *default* akan dimulai dari 0 dan naik satu demi satu berdasarkan urutan konstanta dalam pendefinisian). Sehingga dengan mendefinisikan seperti berikut :

```
enum {
     staff = 4, manajer, direktur
} jenjang_jab;
```

maka **staff** tidak lagi berupa nilai 0, melainkan berupa nilai 4. Dengan sendirinya, **manajer** bernilai 5 dan **direktur** bernilai 6.

```
/* File program : enum3.c
Contoh mengubah nilai default dari suatu tipe enumerasi */

#include <stdio.h>

main()
{
     /* definisi tipe data enumerasi */
     enum {
          staff = 4, manajer, direktur
     } jenjang_jab;

     for(jenjang_jab = staff; jenjang_jab <= direktur;
        jenjang_jab++)
          printf("%d\n", jenjang_jab);
}
```

Contoh eksekusi :

```
4
5
6
```

10.4 Typedef

Untuk kepentingan memperjelas dokumentasi program C, user bisa menamakan suatu tipe data dengan pengenal *(identifier)* yang lebih memberi arti atau mudah diingat. Caranya adalah dengan memakai *typedef*. Sebagai contoh pengenal BYTE dapat digunakan untuk menyatakan *unsigned char*.

Bentuk umum pernamaan suatu tipe data menggunakan *typedef* :

```
typedef tipe_data nama_baru;
```

Contoh :

```
typedef unsigned char BYTE;
typedef char karakter;

karakter a;
```

Contoh tsb menyatakan bahwa BYTE identik dengan *unsigned char*. Sesudah pendefinisian tersebut, BYTE dapat digunakan untuk mendeklarasikan variabel atau jenis parameter fungsi, bahkan juga keluaran fungsi. Misalnya :

```
BYTE kode;
```

untuk mendeklarasikan variabel kode agar bertipe BYTE (atau *unsigned char*) . Contoh lain :

```
BYTE beri_nilai_awal(void);
```

Menyatakan bahwa keluaran fungsi **beri_nilai _awal()** bertipe BYTE.

```
/* File program : typedef.c
Contoh penggunaan typedef */

#include <stdio.h>

/* BYTE merupakan nama baru dari unsigned char */
typedef unsigned char BYTE;

BYTE beri_nilai_awal(void);   /* deklarasi fungsi */

main()
{
     BYTE kode;          /* deklarasi variabel karakter */
```

```
     kode = beri_nilai_awal();
     printf("Isi kode = %u\n", kode);
}


BYTE beri_nilai_awal(void)
{
     return(143);
}
```

Contoh eksekusi :

```
Isi kode = 143
```

Contoh lain penamaan tipe dengan **typedef** :

- `typedef char *STRING;`

  menyatakan bahwa tipe **STRING** adalah tipe pointer yang menunjuk data *char (pointer to char).*

- ```
  typedef struct {
    unsigned char ascii;
    unsigned char atribut;
  } karakter_layar;
  ```

  Pada contoh ini, **karakter_layar** adalah nama lain dari

  ```
  struct data_karakter {
      unsigned char ascii;
      unsigned char atribut;
  };
  ```

10.5 Ternary Operator

C menyediakan sebuah operator yang tergolong sebagai operator ternary, yakni operator yang memiliki tiga buah operand. Operator tersebut dinamakan sebagai operator kondisi. Bentuk ungkapan yang menggunakan operator ini :

kondisi1 ? ungkapan1 : ungkapan2;

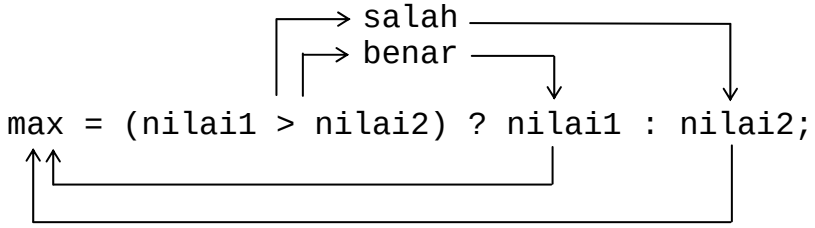
Maksud dari ungkapan kondisi :

- Jika kondisi bernilai benar, maka nilai ungkapan kondisi berupa ungkapan1
- Jika kondisi bernilai salah, maka nilai ungkapan kondisi berupa ungkapan2
  ```

Contoh penggunaan misalnya untuk memperoleh nilai terbesar di antara dua buah bilangan (berupa **nila1** dan **nilai2**). Misalkan nilai terbesar tersebut diberikan (di-*assign*) ke variabel **max**. Penggunaannya :

```
max = (nilai1 > nilai2) ? nilai1 : nilai2;
```

Pada contoh di atas, kalau kondisi (**nilai1** < **nilai2**) bernilai benar, maka **max** akan bernilai **nilai1**, dan keadaan sebaliknya akan bernilai **nilai2**.



Catatan : penulisan kondisi **(nilai1 < nilai2)** sebenarnya bisa ditulis menjadi **nilai1 < nilai2** tanpa menyertakan kurung, disebabkan operator > memiliki prioritas lebih tinggi daripada operator kondisi (?). Pemberian tanda kurung hanya untuk menambah kejelasan.

Pernyataan :

```
max = (nilai1 > nilai2) ? nilai1 : nilai2;
```

kalau ditulis dengan menggunakan if-else adalah sebagai berikut :

```
if (nilai1 > nilai2)
     max = nilai1;
else max = nilai2;
```

Sebagai contoh perhatikan program di bawah ini

```
/* File program : max.c
Menentukan nilai terbesar dengan ternary operator */

#include <stdio.h>

main()
{
     float nilai1, nilai2, max;

     printf("Masukkan dua buah nilai : ");
     scanf("%f %f", &nilai1, &nilai2);

     max = (nilai1 > nilai2) ? nilai1 : nilai2;
     printf("Nilai terbesar = %g\n", max);
}
```

Contoh eksekusi :

```
Masukkan dua buah nilai : 9 10.5
Nilai terbesar = 10.5
```

10.6 Type Cast

Type cast merupakan upaya untuk mengkonversikan suatu tipe data menjadi tipe yang lain.

Bentuk umum *type cast* adalah :

(tipe) ungkapan

dengan tipe dapat berupa pengenal tipe *char*, *int*.

Misalnya, jika **x** dideklarasikan bertipe int. Bila dikehendaki agar ungkapan : `x/2` menghasilkan nilai pecahan *(float)*, maka ungkapan `x/2` perlu ditulis menjadi :

```
(float) x/2;
```

Perbedaan penggunaan *type cast* dengan yang tidak menggunakannya dapat dilihat pada contoh program di bawah ini.

```
/* File program : typecast.c
Melihat efek cast dalam konversi tipe */

#include <stdio.h>

main()
{
        int x = 21;
        float y;

        y = x/2;
        printf("y = x/2           = %f\n", y);

        y = (float) x/2;
        printf("y = (float) x/2   = %f\n", y);

        y = (float) (x/2);
        printf("y = (float) (x/2) = %f\n", y);
}
```

Contoh eksekusi :

```
y = x/2           = 10.000000
y = (float) x/2   = 10.500000
y = (float) (x/2) = 10.000000
```

- Tampak bahwa jika ungkapan **y = x/2** tidak menggunakan type cast, maka variabel y akan bernilai **10.000000** untuk **x = 21**, tetapi jika ditulis **y = (float) x/2,** maka didapat nilai **y = 10.500000 .**
- Adanya **(float) x/2** mengakibatkan **x** bertipe float. Berdasarkan sifat konversi, jika salah satu operand bertipe real, dengan sendirinya yang lain juga akan bertipe real. Oleh karena itu ungkapan **(float) x/2** menghasilkan pembagian real.
- Pada ungkapan **y = x/2**, baik **x** maupun **2** bertipe integer, maka yang terjadi adalah operasi pembagian bulat, baru kemudian hasil pembagiannya dikonversikan secara otomatis (karena adanya tanda *assignment* '=') dengan tipe data dari y, sehingga **y = 10.000000.**
- Penulisan **(float) x/2** berbeda dengan **(float)** (**x/2**). Pada **(float)** (**x/2**)**,** yang dikonversikan ke float adalah hasil dari **x/2**, sedangkan operasi pembagian **x/2** sendiri dianggap sebagai operasi pembagian bulat.

Kesimpulan :

- Union memungkinkan suatu lokasi memori dapat ditempati oleh dua atau lebih variabel yang bisa saja tipenya berlainan.
- Seperti halnya pada struktur, suatu variabel union dapat dilewatkan ke dalam suatu fungsi sebagai parameter.
- Suatu bit atau beberapa bit dalam sebuah data berukuran satu byte atau dua byte dapat diakses dengan mudah melalui ***bitfield***. Dengan cara ini, suatu bit atau beberapa bit dapat diakses tanpa melibatkan operator pemanipulasi bit (seperti &, |). Selain itu, satu atau dua byte memori dapat dipakai untuk menyimpan sejumlah informasi.
- Tipe enumerasi merupakan himpunan dari konstanta integer yang diberi nama.
- Untuk kepentingan memperjelas dokumentasi program C, user bisa menamakan suatu tipe data dengan pengenal *(identifier)* yang lebih memberi arti atau mudah diingat dengan memakai ***typedef***.
- C menyediakan sebuah operator yang tergolong sebagai operator ternary, yakni operator yang memiliki tiga buah operand. Operator tersebut dinamakan sebagai operator kondisi, yang merupakan cara lain dari *if-else* untuk penyeleksian kondisi.
- *Type cast* merupakan upaya untuk mengkonversikan suatu tipe data menjadi tipe yang lain.

Latihan :

Modifikasilah potongan program di bawah ini dengan menggunakan ternary operator

```
if(total_pembelian >= 100.000)
      discount = 0.05 * total_pembelian;
else
      discount = 0;
```