BAB IV
PENGULANGAN PROSES

Tujuan :

1. Menjelaskan proses pengulangan menggunakan pernyataan for
2. Menjelaskan proses pengulangan menggunakan pernyataan while
3. Menjelaskan proses pengulangan menggunakan pernyataan do-while
4. Menjelaskan penggunaan pernyataan break
5. Menjelaskan penggunaan pernyataan continue
6. Menjelaskan penggunaan pernyataan goto
7. Menjelaskan loop di dalam loop (nested loop) dan contoh kasusnya
8. Menjelaskan penggunaan exit() untuk menghentikan eksekusi program dan contoh kasusnya

4.1 Pernyataan *for*

Mengulang suatu proses merupakan tindakan yang banyak dijumpai dalam pemrograman. Pada semua bahasa pemrograman, pengulangan proses ditangani dengan suatu mekanisme yang disebut *loop*. Dengan menggunakan *loop*, suatu proses yang berulang misalnya menampilkan tulisan yang sama seratus kali pada layar dapat diimpelementasikan dengan kode program yang pendek.

Pernyataan pertama yang digunakan untuk keperluan pengulangan proses adalah pernyataan *for*. Bentuk pernyataan ini :

```
for (ungkapan1; ungkapan2; ungkapan3)
  pernyataan;
```

Kegunaan dari masing-masing ungkapan pada pernyataan *for*.

- Ungkapan1 : digunakan untuk memberikan inisialisasi terhadap variabel pengendali *loop*.
- Ungkapan2 : dipakai sebagai kondisi untuk keluar dari *loop*.
- Ungkapan3 : dipakai sebagai pengatur kenaikan nilai variabel pengendali *loop*.

Ketiga ungkapan dalam *for* tersebut harus dipisahkan dengan tanda titik koma (;). Dalam hal ini pernyatan bisa berupa pernyataan tunggal maupun jamak. Jika pernyataannya

berbentuk jamak, maka pernyataan-pernyataan tersebut harus diletakkan di antara kurung kurawal buka ({) dan kurung kurawal tutup (}), sehingga formatnya menjadi :

```
for (ungkapan1; ungkapan2; ungkapan3)
{
   pernyataan;
   pernyataan;
   .
   .
   .
}
```

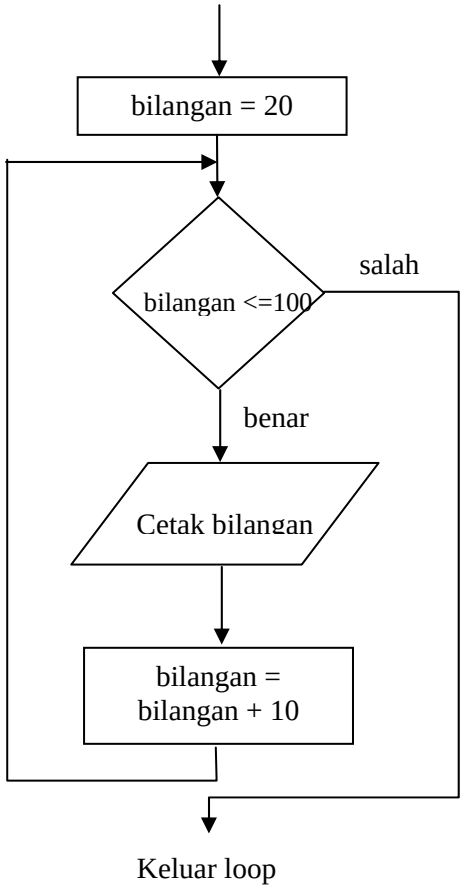
Contoh penggunaan *for*, misalnya untuk menampilkan deretan angka sebagai berikut :

```
20
30
40
50
.
.
.
100
```

Untuk keperluan ini, pernyataan *for* yang digunakan berupa :

```
for (bilangan = 20; bilangan <= 100; bilangan += 10)
  printf("%d\n", bilangan);
```

Kalau digambarkan dalam bentuk diagram alir, akan terlihat sbb :



Gambar 4.1. Diagram alir for

```
/* File program : for1.c
Contoh pemakaian for untuk membentuk deret naik */

#include <stdio.h>

main()
{
     int bilangan;

     for(bilangan = 20; bilangan <= 100; bilangan += 10)
          printf("%d\n", bilangan);
}
```

Contoh eksekusi :

```
20
30
40
50
60
70
80
90
100
```

Pada program di atas, kenaikan terhadap variabel pengendali *loop* sebesar 10 (positif), yang dinyatakan dengan ungkapan

```
bilangan += 10
```

yang sama artinya dengan

```
bilangan = bilangan + 10
```

Pada contoh yang melibatkan pernyataan *for* di atas, kenaikan variabel pengendali *loop* berupa nilai positif. Sebenarnya kenaikan terhadap variabel pengendali *loop* bisa diatur bernilai negatif. Cara ini dapat digunakan untuk memperoleh deret sebagai berikut :

```
60
50
40
30
20
10
```

Untuk itu selengkapnya program yang dibutuhkan adalah sebagai berikut :

```
/* File program : for2.c
   Contoh pemakaian for untuk membentuk deret turun */

#include <stdio.h>

main()
{
     int bilangan;

     for (bilangan = 60; bilangan >= 10; bilangan -= 10)
          printf("%d\n", bilangan);
}
```

Contoh eksekusi :

```
60
50
40
30
20
10
```

Kadang-kadang dijumpai adanya pernyataan *for* yang tidak mengandung bagian ungkapan yang lengkap (beberapa ungkapan dikosongkan). Dengan cara ini, pernyataan

```
for (bilangan = 20; bilangan <= 100; bilangan += 10)
     printf("%d\n", bilangan);
```

dapat ditulis menjadi :

```
bilangan = 20;      /* inisialisasi di luar for */
for (  ; bilangan <= 100;  )
                                   ungkapan
                                   kosong
 {
     printf("%d\n", bilangan);
     bilangan += 10;
 }
```

Tampak bahwa ungkapan yang biasa dipakai untuk inisialisasi variabel pengendali *loop* tak ada. Sebagai gantinya pengendalian *loop* diatur sebelum pernyataan for, berupa

```
bilangan = 20;
```

Pengosongan ini juga dilakukan pada ungkapan yang biasa dipakai untuk menaikkan nilai variabel pengendali *loop*. Sebagai gantinya, di dalam tubuh *loop* diberikan pernyataan untuk menaikkan nilai variabel pengendali *loop*, yaitu berupa

```
bilangan += 10;
```

Ungkapan yang tidak dihilangkan berupa bilangan <=100. Ungkapan ini tetap disertakan karena dipakai sebagai kondisi untuk keluar dari *loop*.

Sesungguhnya ungkapan yang dipakai sebagai kondisi keluar dari *loop* juga bisa dihilangkan, sehingga bentuknya menjadi

```
for (;;)
   pernyataan
```

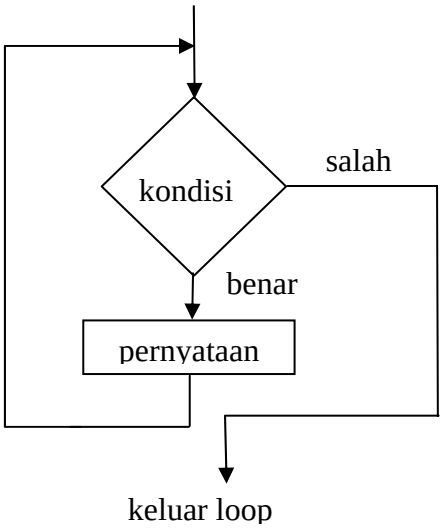
Suatu pertanyaan mungkin timbul “Lalu bagaimana caranya kalau ingin keluar dari *loop* pada bentuk di atas?”. Caranya adalah dengan menggunakan pernyataan yang dirancang khusus untuk keluar dari *loop*. Mengenai hal ini akan dibahas pada sub bab yang lain.

4.2 Pernyataan *while*

Pada pernyataan *while*, pengecekan terhadap loop dilakukan di bagian awal (sebelum tubuh loop). Lebih jelasnya, bentuk pernyataan *while* adalah sebagai berikut :

```
while (kondisi)
   pernyataan;
```

dengan pernyataan dapat berupa pernyataan tunggal, pernyataan majemuk ataupun pernyataan kosong. Proses pengulangan terhadap pernyataan dijelaskan pada gambar berikut :



Gambar 4.2. Diagram alir *while*

Dengan melihat gambar 4.2, tampak bahwa ada kemungkinan pernyataan yang merupakan tubuh loop tidak dijalankan sama sekali, yaitu kalau hasil pengujian kondisi *while* yang pertama kali ternyata bernilai salah.

Contoh pemakaian *while* misalnya untuk mengatur agar tombol yang ditekan oleh pemakai program berupa salah satu diantara 'Y','y', 'T' atau 't'. Impelementasinya :

```
/*File program : pilihan.c
Untuk membaca tombol Y atau T */

#include <stdio.h>

main()
{
     char pilihan;
     /* diberi nilai salah lebih dahulu */
     int sudah_benar = 0;

     printf("Pilihlah Y atau T.\n");

/* program dilanjutkan jika tombol Y,y,T atau t ditekan */

     while(!sudah_benar)
     {
        pilihan = getchar();        /* baca tombol */
        sudah_benar = (pilihan == 'Y') || (pilihan == 'y')||
          (pilihan == 'T') || (pilihan == 't');
     }

     /* memberi keterangan tentang pilihan */
     switch(pilihan)
     {
     case 'Y':
     case 'y':
          puts("\nPilihan anda adalah Y");
          break;
     case 'T':
     case 't':
          puts("\nPilihan anda adalah T");
     }
}
```

Contoh eksekusi :

```
Pilihlah Y atau T
Pilihan anda adalah Y
```

Inisialisasi terhadap variabel sudah_benar yang akan dijalankan pada kondisi *while* dengan memberi nilai awal bernilai *false* (`sudah_benar = 0`) dimaksudkan agar tubuh loop

```
{
     pilihan = getchar( ); /* baca tombol */
     sudah_benar = (pilihan == 'Y') || (pilihan== 'y')||
        (pilihan == 'T') || (pilihan == 't');
}
```

dijalankan minimal sekali.

Contoh lain pemakaian *while* dapat dilihat pada program yang digunakan untuk menghitung banyaknya karakter dari kalimat yang dimasukkan melalui keyboard (termasuk karakter spasi). Untuk mengakhiri pemasukan kalimat, tombol ENTER (‘\n’) harus ditekan. Karena itu, tombol ENTER inilah yang dijadikan kondisi penghitungan jumlah spasi maupun karakter seluruhnya. Lengkapnya, kondisi yang dipakai dalam *while* berupa :

```
while((kar = getchar()) != '\n')
```

Ungkapan di atas mempunyai arti :

- Bacalah sebuah karakter dan berikan ke variabel **kar**
- Kemudian bandingkan apakah karakter tersebut = ‘\n’ (ENTER)

Ungkapan menghasilkan nilai benar jika tombol yang ditekan bukan ENTER. Pada program kalau tombol yang ditekan bukan ENTER , maka :

- Jumlah karakter dinaikkan sebesar satu melalui pernyataan : `jumkar++;`
- Kalau karakter berupa SPASI, maka jumlah spasi dinaikkan sebesar satu, melalui pernyataan : `if (kar == ‘ ‘) jumspasi++;`

```
/* File program : jumkar.c
Menghitung jumlah kata dan karakter dalam suatu kalimat */

#include <stdio.h>

main()
{
     char kar;
     int jumkar = 0, jumspasi = 0;

     puts("Masukkan sebuah kalimat dan akhiri dgn ENTER.\n");
     puts("Saya akan menghitung jumlah karakter ");
     puts("pada kalimat tersebut.\n");

     while((kar = getchar()) != '\n')
     {
```

```
          jumkar++;
          if (kar == ' ') jumspasi++;
     }

     printf("\nJumlah karakter          = %d", jumkar);
     printf("\nJumlah SPASI             = %d\n\n", jumspasi);
}
```

Contoh eksekusi :

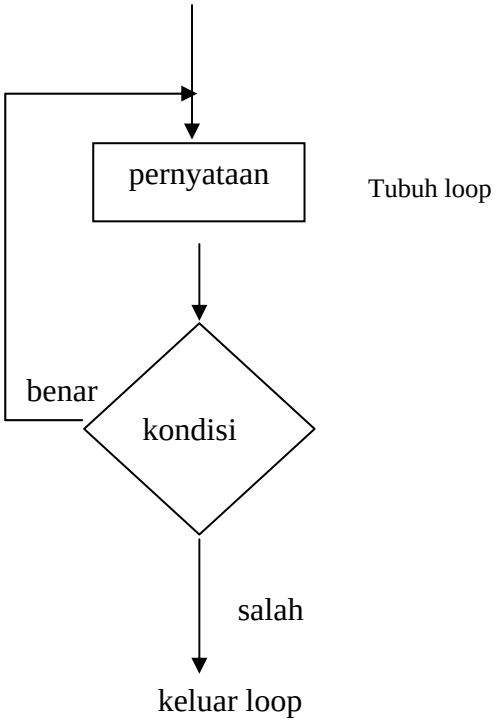
```
Masukkan sebuah kalimat, akhiri dgn ENTER.
Saya akan menghitung jumlah karakter pada kalimat tersebut.

Belajar bahasa C sangat menyenangkan

Jumlah karakter  = 36
Jumlah SPASI      = 4
```

4.3 Pernyataan *do-while*

Bentuk pernyataan ***do-while***

```
do
   pernyataan;
while (kondisi)
```

Pada pernyataan *do-while*, tubuh *loop* berupa pernyataan,dengan pernyataan bisa berupa pernyataan tunggal, pernyataan majemuk ataupun pernyataan kosong. Pada pernyataan *do*, mula-mula pernyataan dijalankan. Selanjutnya, kondisi diuji. Sendainya kondisi bernilai benar, maka pernyataan dijalankan lagi, kemudian kondisi diperiksa kembali, dan seterusnya. Kalau kondisi bernilai salah pada saat dites, maka pernyataan tidak dijalankan lagi. Untuk lebih jelasnya dapat dilihat pada Gambar 4.3. Berdasarkan Gambar 4.3 terlihat bahwa tubuh *loop* minimal akan dijalankan sekali.



Gambar 4.3. Diagram alir *do-while*

Program berikut memberikan contoh pemakaian *do-while* untuk mengatur penampilan tulisan "BAHASA C" sebanyak sepuluh kali.
Contoh:

```
i = 0;
do
{
   puts("BAHASA C");
   i++;
} while(i<10);
```

Pada program di atas, variabel pencacah dipakai untuk menghitung jumlah tulisan yang sudah ditampilkan pada layar. Selama nilai pencacah kurang dari 10, maka perintah

```
puts("BAHASA C");
```

akan dilaksanakan kembali

Penanganan pembacaan tombol pada contoh program **pilihan.c** yang memakai *while* di atas, kalau diimplementasikan dengan memakai *do-while* adalah sebagai berikut

```
/* File program : pilihan2.c
Untuk membaca tombol Y atau T */
```

```
#include <stdio.h>
main()
{
     char pilihan;
     int sudah_benar;

     printf("Pilihlah Y atau T.\n");

/* program dilanjutkan kalau tombol Y,y,T atau t ditekan */
     do
     {
         pilihan = getchar( ); /* baca tombol */
         sudah_benar = (pilihan == 'Y') || (pilihan== 'y')||
          (pilihan == 'T') || (pilihan == 't');
     } while(! sudah_benar);

     /* memberi keterangan tentang pilihan */
     switch(pilihan)
     {
     case 'Y':
     case 'y':
          puts("\nPilihan anda adalah Y");
          break;
     case 'T':
     case 't':
          puts("\nPilihan anda adalah T");
     }
}
```

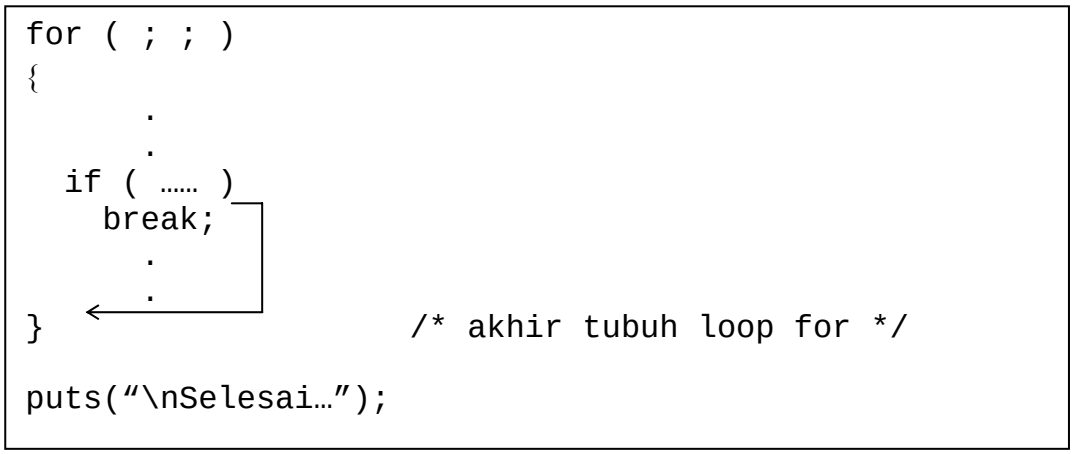
Contoh eksekusi :

```
Pilihlah Y atau T
Pilihan anda adalah T
```

Mula-mula tombol dibaca dengan menggunakan *getchar()* dan kemudian diberikan ke variabel pilihan. Sesudah itu, variabel **sudah_benar** akan diisi dengan nilai benar (1) atau salah (0) tergantung dari nilai pilihan. Kalau pilihan berisi salah satu diantara ‘Y’,‘y’, ‘T’ atau ‘t’, maka sudah berisi salah satu diantara ‘Y’, ‘y’, ‘T’ atau ‘t’, maka **sudah_benar** akan berisi benar. Nilai pada vaiabel **sudah_benar** ini selanjutnya dijadikan sebagai kondisi *do-while*. Pengulangan terhadap pembacaan tombol akan dilakukan kembali selama **sudah_benar** benilai salah.

4.4 Pernyataan break.

Pernyataan break sesungguhnya telah diperkenalkan pada pernyataan *switch*. Pernyataan ini berfungsi untuk keluar dari *loop for*, *do-while* dan *while*. Sedangkan pada *switch* yaitu untuk menuju ke akhir (keluar dari) struktur *switch*. Sebagai contoh dapat dilihat pada gambar 4.4. Kalau pernyataan *break* dijalankan maka eksekusi akan dilanjutkan ke pernyataan yang terletak sesudah akhir tubuh *loop for*.



Gambar 4.4 Ilustrasi pengaruh break

Pada contoh potongan program berikut, pembacaan dan penampilan terhadap tombol yang ditekan akan berakhir kalau tombol yang ditekan adalah ENTER (‘\n’). Pernyataan yang digunakan untuk keperluan ini :

```
if (kar == ‘\n’)
    break;   /* keluar dari loop for */
```

Yang menyatakan “Jika tombol yang ditekan berupa ENTER, maka keluarlah dari *loop for*”. Untuk lebih jelasnya, perhatikan program di bawah ini.

```
/* File program : tamat.c
Pemakaian break untuk keluar dari looping */

#include <stdio.h>

main()
{
    char kar;

    printf("Ketik sembarang kalimat");
```

```
    printf(" dan akhiri dengan ENTER\n\n");

    for ( ; ; )
    {
      kar = getchar();
      if(kar == '\n')
          break;
    }
    printf("Selesai\n");
}
```

Contoh eksekusi :

```
Ketik sembarang kalimat dan akhiri dengan ENTER :

Menulis apa saja
Selesai
```

Jika pernyataan *break* berada dalam loop yang bertingkat (*nested loop)*, maka pernyataan *break* hanya akan membuat proses keluar dari loop yang bersangkutan (tempat break dituliskan), bukan keluar dari semua loop.

4.5 Pernyataan Continue

Pernyataan *continue* digunakan untuk mengarahkan eksekusi ke iterasi (proses) berikutnya pada *loop* yang sama. Pada *do-while* dan *while*, pernyataan *continue* menyebabkan eksekusi menuju ke kondisi pengujian pengulangan, seperti yang dilukiskan pada Gambar 4.5. Pada *loop for*, pernyataan contunue menyebabkan bagian penaik variabel pengendali *loop* dikerjakan (ungkapan3 pada struktur *for*) dan kondisi untuk keluar dari *loop for* (ungkapan2 pada struktur *for*) diuji kembali.

Program ini digunakan untuk memasukkan data harus diulangi dan hal ini dikendalikan dengan *continue*. Untuk mengakhiri pemasukan data, data yang dimasukkan harus bernilai kurang dari 0. Perlu diketahui kondisi bernilai 1.

Gambar 4.5 Pengaruh *continue* pada *while* dan *do-while*

Menyatakan bahwa kondisi selalu dianggap benar. Untuk keluar dari *loop*, pernyataan yang digunakan berupa *break*.

Pengaruh *continue* pada *loop for* diperlihatkan pada dibawah ini. Program ini dipakai untuk menampilkan bilangan ganjil yang terletak antara 7 sampai dengan 25, kecuali 15.

```
/* File program : ganjil.c
menampilkan bilangan ganjil antara 7 - 25 kecuali 15 */

#include <stdio.h>

main()
{
    int x;

    for (x = 7; x <= 25; x += 2)
    {
      if  (x == 15)
         continue;
      printf("%4d", x);
    }
    printf("\n");
}
```

Contoh eksekusi :

```
  7   9  11  13  17  19  21  23  25
```

Pada program di atas, untuk menghindari agar nilai 15 tidak ditampilkan ke layar, pernyataan yang digunakan berupa

```
if ( x == 15)
     continue;
```

Artinya, jika kondisi `x == 15` bernilai benar, pernyataan *continue* menyebabkan pernyataan sisanya yaitu

```
printf(“%d”,x);
```

diabaikan dan eksekusi diarahkan kepada ungkapan :

```
x += 2
```

dan kemudian menguji kondisi :

```
x <= 25
```

Pada program di atas, pernyataan :

```
for (x = 7; x <= 25; x += 2)
{
  if  (x == 15)
     continue;
  printf("%4d", x);
}
```

dapat ditulis dalam bentuk lain sebagai berikut :

```
for (x = 7; x <= 25; x += 2)
    if (x != 15)
       printf(“%4d”, x);
```

4.6 Loop Di Dalam Loop

Dalam suatu *loop* bisa terkandung *loop* yang lain. *Loop* yang terletak di dalam *loop* biasa disebut dengan *loop* di dalam *loop* (nested *loop*). Salah satu contoh nested *loop* misalnya pada permasalahan untuk membuat tabel perkalian:

	1	2	3	4	5	6	7	8
1	1	2	3	4	5	6	7	8
2	2	4	6	8	10	12	14	16
3	3	6	9	12	15	18	21	24
4	4	8	12	16	20	24	28	32
5	5	10	15	20	25	30	35	40
6	6	12	18	24	30	36	42	48
7	7	14	21	28	35	42	49	56
8	8	16	24	32	40	48	56	64

Implementasi dalam program selengkapnya adalah sebagai berikut :

```
/* File program : tblkali.c
 Loop for bersarang untuk membuat tabel perkalian */
```

```
#include <stdio.h>

#define MAKS 8

main()
{
  int baris, kolom, hasil_kali;

  for (baris = 1; baris <= MAKS; baris++)
  {
     for (kolom = 1; kolom <= MAKS; kolom++)
     {
          hasil_kali = baris * kolom;
          printf ("%2d", hasil_kali);
     }
     printf("\n");    /* pindah baris */
  }
}
```

Bagian yang terletak dalam bingkai di depan dapat dapat diperoleh melalui

```
for (baris = 1; baris <= MAKS; baris++)
{
   hasil_kali = baris * kolom;
   printf (“%2d”, hasil_kali);
}
```

dengan **MAKS** didefinisikan bernilai 8. Bagian *loop* yang terdalam :

```
for (kolom  = 1; kolom <= MAKS; kolom++)
{
   hasil_kali = baris * kolom;
   printf (“%2d”, hasil_kali);
}
```

digunakan untuk mencetak suatu deret hasil perkalian dalam satu baris. Untuk berpindah ke baris berikutnya, pernyataan yang digunakan yaitu

```
printf(“\n”);
```

Adapun pencetakan untuk semua baris dikendalikan melalui

```
for (baris = 1; baris <= MAKS; baris++)
```

Pernyataan di atas mempunyai arti “dari baris ke-1 sampai dengan baris ke-MAKS”.

4.7 Pernyataan *goto*

Pernyataan *goto* merupakan intruksi untuk mengarahkan eksekusi ke pernyataan yang diawali dengan suatu label. Label sendiri berupa suatu pengenal (*identifier*) yang diikuti dengan tanda titik dua (:)

Contoh pemakaian *goto* ditujukan pada program dibawah ini:

Pernyataan

```
goto cetak;
```

Mengisyaratkan agar eksekusi dilanjutkan ke pernyataan yang diawali dengan label

```
cetak:
```

Pernyataan

```
if (++pencacah <= 10)
     goto cetak;
```

Mempunyai arti :

- Naikkan nilai **pencacah** sebesar 1
- Kemudian, jika **pencacah** kurang dari atau sama dengan 10 maka eksekusi menuju ke label **cetak.**

Penerapan *goto* biasanya dilakukan pada *loop* di dalam *loop* (*nested loop*), dengan tujuan memudahkan untuk keluar dari *loop* terdalam menuju ke pernyataan yang terletak di luar *loop* terluar.

4.8. Menggunakan *exit ( )* Untuk Menghentikan Eksekusi Program.

Suatu eksekusi program dapat dihentikan (secara normal) melalui pemanggilan fungsi *exit( ).* Hal ini biasa dilakukan, jika di dalam suatu eksekusi terdapat suatu kondisi yang tak dikehendaki. Prototipe dari fungsi *exit()* didefinisikan pada file **stdlib.h**, yang memiliki deklarasi sebagai berikut :

```
void exit(int status);
```

Menurut kebiasaan, nilai nol diberikan pada argumen *exit()* untuk menunjukkan penghentian program yang normal. Sedangkan untuk menunjukkan kesalahan, nilai yang

diberikan pada argumen fungsi diisi dengan nilai bukan-nol. Pada contoh program berikut, eksekusi program akan dihentikan hanya jika tombol 'X' ditekan

```
/* File program : keluar.c
Pemakaian exit() untuk menghentikan eksekusi program */

#include <stdio.h>
#include <stdlib.h>

main()
{
  char kar;

  printf("Tekanlah X untuk menghentikan program.\n");

  for ( ; ;)
  {
     while ((kar = getchar()) == 'X')
        exit(0);
  }
}
```

Kesimpulan :

- Pada semua bahasa pemrograman, pengulangan proses ditangani dengan suatu mekanisme yang disebut *loop*.
- Pernyataan-pernyataan yang bisa digunaan untuk keperluan pengulangan proses *(looping)* adalah :

  (a) Pernyataan *for*, dengan bentuk umum sebagai berikut:

  ```
  for (ungkapan1; ungkapan2; ungkapan3)
    pernyataan;
  ```

  Jika pernyataannya berbentuk jamak, maka pernyataan-pernyataan tersebut harus diletakkan di antara kurung kurawal buka ({) dan kurung kurawal tutup (})

  (b) Pernyataan *while*, dengan bentuk umum sebagai berikut:

  ```
  while (kondisi)
     pernyataan;
  ```

(c) Pernyataan *do-while*, dengan bentuk umum sebagai berikut :

```
do
  pernyataan;
while (kondisi)
```

- Pernyataan *break* berfungsi untuk keluar dari *loop for*, *do-while* dan *while*.
- Pernyataan *continue* digunakan untuk mengarahkan eksekusi ke iterasi (proses) berikutnya pada *loop* yang sama.
- Dalam suatu *loop* bisa terkandung *loop* yang lain (nested *loop*).
- Pernyataan *goto* merupakan intruksi untuk mengarahkan eksekusi ke pernyataan yang diawali dengan suatu label. Label sendiri berupa suatu pengenal (*identifier*) yang diikuti dengan tanda titik dua (:)
- Suatu eksekusi program dapat dihentikan (secara normal) melalui pemanggilan fungsi *exit( )*. Hal ini biasa dilakukan, jika di dalam suatu eksekusi terdapat suatu kondisi yang tak dikehendaki.

Latihan :

Buatlah potongan program untuk soal-soal di bawah ini

1. Gunakan loop *for* untuk menampilkan nilai 1 sampai dengan 10 dalam baris-baris yang terpisah.

2. Gunakan loop *for* untuk mendapatkan tampilan sbb :

   ```
   1
   22
   333
   4444
   55555
   ```

3. Gunakan loop *for* untuk menjumlahkan seluruh bilangan antara 10 sampai dengan 100 ke dalam sebuah variabel **total**. Asumsikan bahwa variabel **total** tidak diinisialisasi terlebih dahulu dengan nilai nol.

4. Gunakan loop *for* untuk menampilkan seluruh karaker dari A sampai dengan Z dalam baris-baris yang terpisah.

5. Hitunglah bilangan triangular dari masukan pengguna, yang dibaca dari keyboard dengan menggunakan *scanf().* Bilangan triangular adalah penjumlahan dari bilangan masukan dengan seluruh bilangan sebelumnya, sehingga bilangan triangular dari 7 adalah : `7 + 6 + 5 + 4 + 3 + 2 + 1`

6. Gunakan loop *while* untuk menampilkan bilangan integer antara 1 sampai dengan 10 di layar sbb : `123456768910`

7. Gunakan *nested while* loop untuk mendapatkan keluran sbb :

   1
   22
   333
   4444
   55555