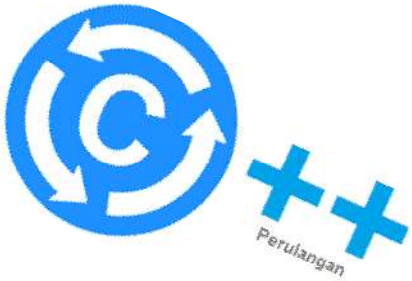
BAB 4.2

PEMOGRAMAN TERSTRUKTUR

STRUKTUR PERCABANGAN & PERULANGAN

Pengulangan: Latar Belakang

- **Melakukan** suatu **instruksi**, bahkan **aksi**, secara **berulang-ulang**
  - Komputer: memiliki performansi yang sama
  - Manusia: punya kecenderungan untuk melakukan kesalahan (karena letih atau bosan)



Jenis-jenis notasi pengulangan :

1. Berdasarkan kondisi pengulangan di akhir : **do... while**
2. Berdasarkan kondisi pengulangan di awal : **while**
3. Berdasarkan pencacah : **for**

Tampilkan kalimat **“Kamu ganteng ! tapi bohong"** sebanyak 10 x ke layar

Mungkin kamu akan menggunakan **cout** sebanyak 10 kali seperti ini

```
#include <iostream>
using namespace std;

int main(){
    cout << "Kamu ganteng ! tapi bohong" << endl;
    cout << "Kamu ganteng ! tapi bohong" << endl;
    cout << "Kamu ganteng ! tapi bohong" << endl;
    cout << "Kamu ganteng ! tapi bohong" << endl;
    cout << "Kamu ganteng ! tapi bohong" << endl;
    cout << "Kamu ganteng ! tapi bohong" << endl;
    cout << "Kamu ganteng ! tapi bohong" << endl;
    cout << "Kamu ganteng ! tapi bohong" << endl;
    cout << "Kamu ganteng ! tapi bohong" << endl;
    cout << "Kamu ganteng ! tapi bohong" << endl;
    return 0;
}
```

```
Kamu ganteng ! tapi bohong
Kamu ganteng ! tapi bohong
Kamu ganteng ! tapi bohong
Kamu ganteng ! tapi bohong
Kamu ganteng ! tapi bohong
Kamu ganteng ! tapi bohong
Kamu ganteng ! tapi bohong
Kamu ganteng ! tapi bohong
Kamu ganteng ! tapi bohong
Kamu ganteng ! tapi bohong

--------------------------------
```

Bagaimana kalau kita menampilkan nya sebanyak **1000 kali**.

Pasti capek dong ngetiknya...

Karena itu, kita harus menggunakan **PERULANGAN**.

B STRUKTUR PERULANGAN

Perulangan akan membantu kita mengeksekusi kode yang berulang-ulang, berapapun yang kita mau.

Secara umum, dibagi menjadi dua kelompok, yaitu:

- ***Counted Loop*** merupakan perulangan yang jelas dan sudah tentu banyak kali perulangannya.

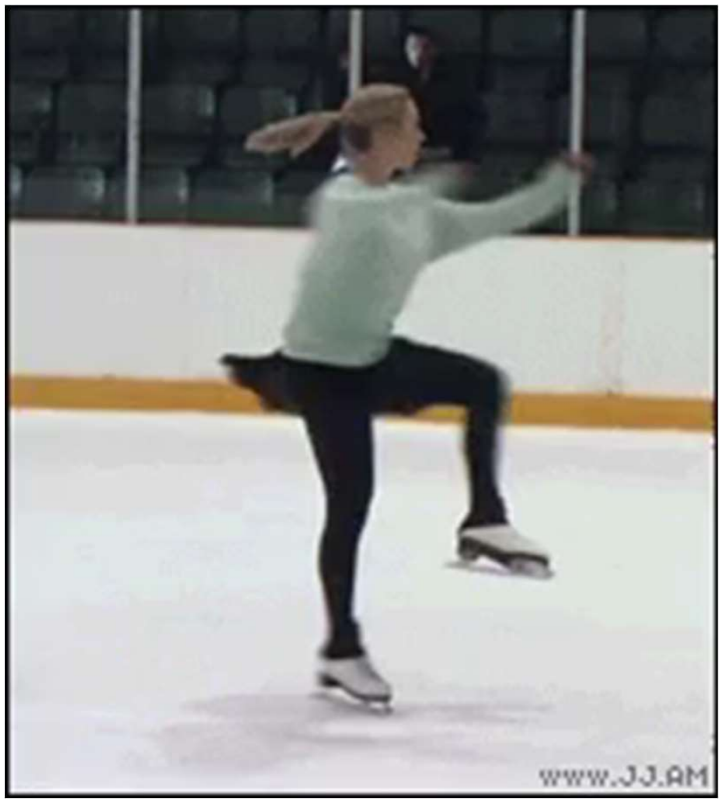
  Perulangan yang termasuk dalam *Counted Loop*:

  1) Perulangan **For**

- ***Uncounted Loop*** merupakan perulangan yang tidak jelas, berapa kali ia harus mengulang.

  Perulangan yang termasuk dalam *Uncounted Loop*:

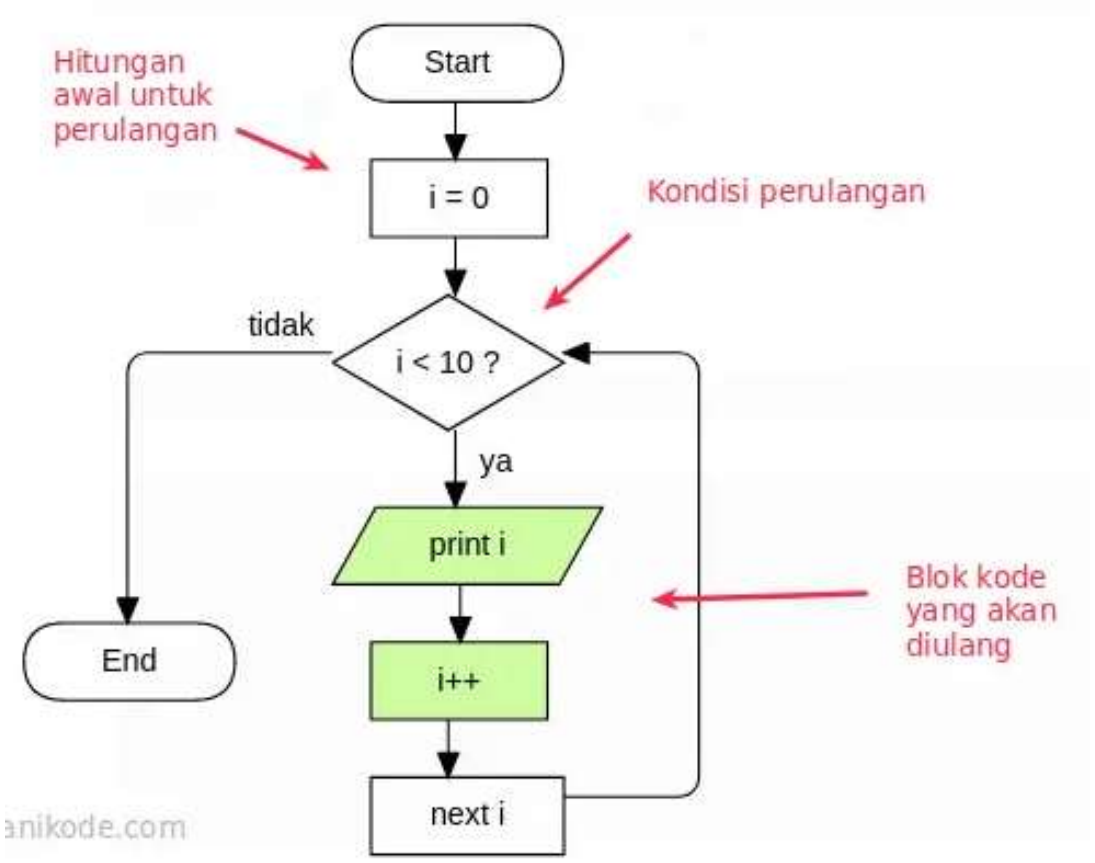
  1) Perulangan **While**
  2) Perulangan **Do ... While**


www.JJ.AM

Struktur perulangan *(looping)* adalah instruksi kode program yang bertujuan untuk mengulang beberapa baris perintah.

Dalam merancang perulangan, paling tidak kita harus mengetahui 3 komponen:

1) Kondisi **awal perulangan**.
2) Kondisi pada **saat perulangan**.
3) Kondisi yang harus dipenuhi agar **perulangan berhenti**.



‘ Masalah Dasar ‘

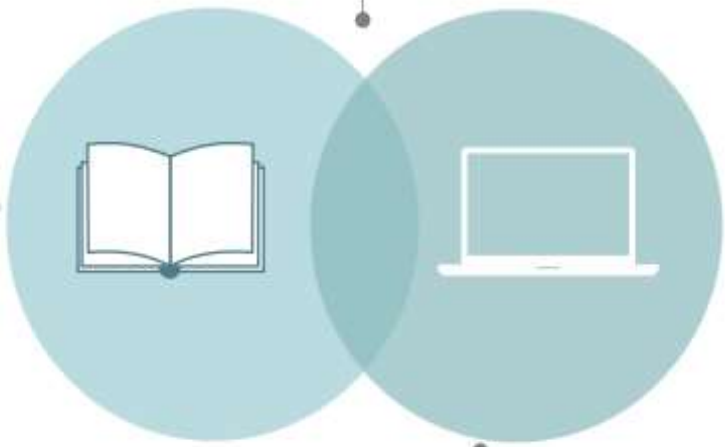
Adanya beberapa yang harus diulang dalam sebuah kasus.

- ✓ Pengulangan langsung dari awal.
- ✓ Pengulangan dengan mengunakan kondisi tertentu.
- ✓ Pengulangan dengan mengunakan kondisi tapi dengan pemenuhan 1x proses.
- ✓ Pengulangan yang didalamnya ada perlakuan pengulangan juga.

Selain memiliki *control structure* untuk penyeleksian, bahasa C++ juga memiliki *control structure* untuk pengulangan *(looping)* :

- Perulangan - for
- Perulangan - while
- Perulangan - do... While
- Perulangan - nested loop

FOR
For ini digunakan untuk pengulangan yang akhir pengulangannya telah dispesifikasikan.



WHILE
while digunakan untuk melakukan.pengulangan selama kondisi pengulangannya bernilai true

Kondisi pengulangan dilakukan diawal.

REPEAT
Repeat akan melakukan pengulangan sampai kondisi bernilai benar (true),

Kondisi pengulangan dilakukan diakhir.

- Struktur kondisi ” looping ”

- **Statement For Loop**

Perulangan **For** adalah jenis perulangan yang cukup banyak digunakan.

For adalah statemen untuk mengeksekusi banyak perintah secara berulang kali. Artinya, statemen ini berguna untuk membuat program yang dinamis.

Penggunaan perulangan **For** berbeda dengan perulangan while atau do while, yang mana pada perulangan **For** kita sudah dapat mengetahui jumlah dari proses perulangan tersebut (dalam jumlah tertentu).

Sementara untuk while digunakan saat kita belum mengetahui jumlah dari prosesnya.

Berikut umum Perulangan **For** :

```
for (inisialisasi; kondisi; modifier)
 {
    Pernyataan yang dieksekusi;
 }
```

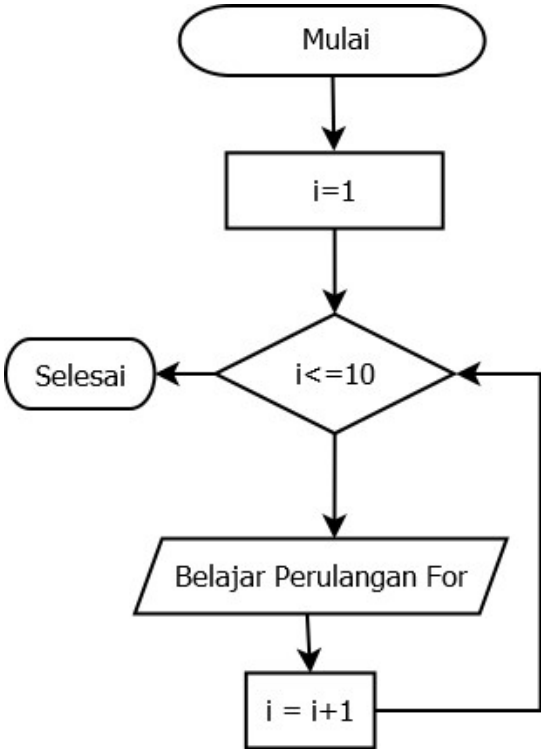
Berikut umum Perulangan **For** :

```
for (ungkapan1; ungkapan2; ungkapan3)
 {
    Pernyataan;
    Pernyataan;
    …
 }
```

Didalam perulangan terdapat 3 parameter penting:

- **`ungkapan1`** = pernyataan **`Inisialisasi`** : Bagian dimana kita membuat nilai awal perulangan. Nilai dapat dibuat di dalam variabel, nantinya dari nilai ini akan menjadi titik awal perulangan dilakukan.
- **`ungkapan2`** = **`Kondisi`** : Bagian dimana kita menentukan suatu kondisi/syarat perulangan dilakukan sehingga perulangan dapat berjalan. Perulangan akan terus dijalankan saat kondisi bernilai benar.
- **`ungkapan3`** = **`Modifier`** : Penyataan *control* untuk perulangan (sebagai pengatur variable yang digunakan di dalam **`ungkapan1`**), yakni bagian dimana kita melakukan *increment* atau *decrement* nilai awal yang sudah diinisialisasi.
- **`pernyataan, …`** = Blok kode yang dijalankan Ketika kondisi perulangan terpenuhi.

- CONTOH :



- ✓ Mulai
- ✓ Inisialisasi nilai awal untuk variabel i=1
- ✓ Cek kondisi apakah nilai i<=10 jika kondisi bernilai benar (*true*) maka kalimat "Belajar perulangan For" akan dicetak.
- ✓ Nilai variabel i dilakukan *increment* (ditambahkan 1)
- ✓ Mengecek lagi kondisinya apakah nilai i masih <= 10 ? Jika ya maka kalimat tersebut akan terus dicetak.
- ✓ Iterasi akan terus dilakukan hingga kondisi bernilai salah (*false*)
- ✓ Selesai

OUTPUT :

```
Belajar Perulangan For 1
Belajar Perulangan For 2
Belajar Perulangan For 3
Belajar Perulangan For 4
Belajar Perulangan For 5
Belajar Perulangan For 6
Belajar Perulangan For 7
Belajar Perulangan For 8
Belajar Perulangan For 9
Belajar Perulangan For 10
```

- CONTOH :

Jika inisialisasi nilai awal variabel **`i=10`**, dan kondisinya **`i>=1`**

Jika kondisi ini bernilai benar (*true*) maka pada bagian *modifier* dilakukan pengurangan *(decrement)* pada variabel i tersebut.

Perulangan terus dilakukan hingga nilai **`i=1`**.

```
1  #include <iostream>
2  #include <conio.h>
3
4  using namespace std;
5
6  int main() {
7
8     for (int i=10;i>=1;i--)
9        {
10          cout<<"Belajar Perulangan For "<<i<<endl;
11       }
12
13    getch();
14 }
15
```

```
Belajar Perulangan For 10
Belajar Perulangan For 9
Belajar Perulangan For 8
Belajar Perulangan For 7
Belajar Perulangan For 6
Belajar Perulangan For 5
Belajar Perulangan For 4
Belajar Perulangan For 3
Belajar Perulangan For 2
Belajar Perulangan For 1
```

CONTOH :

```cpp
#include <iostream>
using namespace std;
int main(){
    cout<<"Contoh Perulangan For Menaik (Increment)"<<endl;
    for(int MD=1;MD<9;MD++){
        cout<<MD<<endl;
    }

    // Fungsi cout<<endl dan \n adalah sama
    cout<<"\nContoh Perulangan For Menurun (Decrement)\n";
    for(int MD=8;MD>0;MD--){
        cout<<MD<<"\n";
    }

    return 0;
}
```

```
Contoh Perulangan For Menaik (Increment)
1
2
3
4
5
6
7
8

Contoh Perulangan For Menurun (Decrement)
8
7
6
5
4
3
2
1
```

- Struktur kondisi ” break & continue ”

Pernyataan **break** dan **continue** berkaitan dengan penggunaan **perulangan for (juga while dan do while)** :

```
1  #include <iostream>
2  #include <conio.h>
3
4  using namespace std;
5
6  int main() {
7
8     for (int i=1;i<=10;i++)
9         {
10        if (i==5){
11          break;
12          }
13        cout<<"Belajar Perulangan For "<<i<<endl;
14        }
15 getch();
16 }
```

➢ **Pernyataan Break**

Peryataan **break** digunakan untuk menghentikan atau keluar dari proses iterasi yang berjalan saat memenuhi suatu kondisi.

```
Belajar Perulangan For 1
Belajar Perulangan For 2
Belajar Perulangan For 3
Belajar Perulangan For 4
```

➢ Pernyataan Continue

Pernyataan **continue** digunakan untuk memecah suatu iterasi yang sedang berjalan saat memenuhi suatu kondisi, namun selanjutnya iterasi akan berjalan kembali.

Berikut contoh perulangan **for** dengan menggunakan pernyaatan **continue**, dimana akan melewati satu proses iterasi saat nilai **i=5**.

```
Belajar Perulangan For 1
Belajar Perulangan For 2
Belajar Perulangan For 3
Belajar Perulangan For 4
Belajar Perulangan For 6
Belajar Perulangan For 7
Belajar Perulangan For 8
Belajar Perulangan For 9
Belajar Perulangan For 10
```

```
1  #include <iostream>
2  #include <conio.h>
3
4  using namespace std;
5
6  int main() {
7
8      for (int i=1;i<=10;i++)
9          {
10         if (i==5){
11           continue;
12           }
13         cout<<"Belajar Perulangan For "<<i<<endl;
14         }
15 getch();
16 }
```

- Struktur kondisi ” looping ”

- **Statement while Loop**

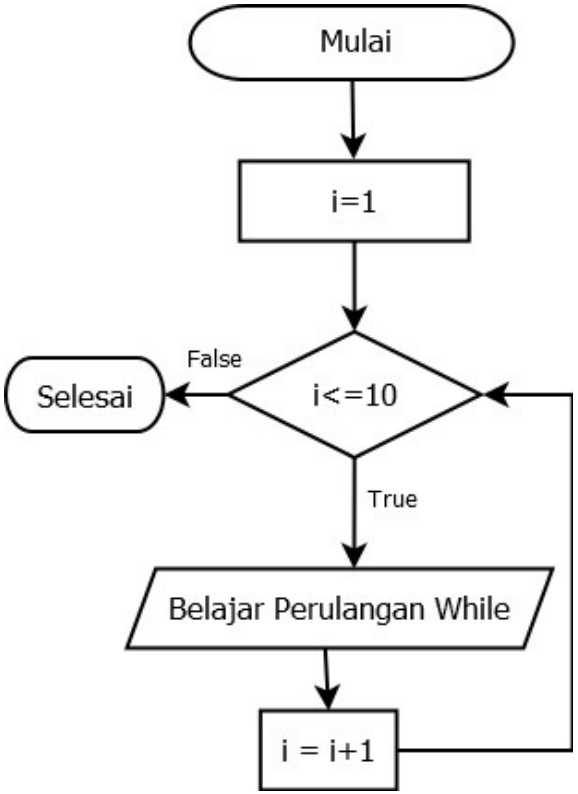
Perulangan **while** banyak digunakan pada program yang terstruktur. Perulangan ini banyak digunakan bila jumlah perulangannya belum diketahui. Berbeda dengan perulangan **for** - kita sudah dapat mengetahui jumlah dari proses perulangannya.

Proses perulangan akan terus berlanjut selama kondisinya bernilai benar *(true)* dan akan berhenti bila kondisinya bernilai salah *(false)*.

Berikut umum Perulangan **while** :

```
while (kondisi)
 {
   perintah yang dilakukan perulangan;
 }
```

- CONTOH :



- ✓ Mulai
- ✓ Inisialisasi nilai awal i=1
- ✓ Cek kondisi apakah nilai variabel i<=10 ? Jika benar maka akan dicetak kalimat "Belajar Perulangan While"
- ✓ Nilai variabel i ditambah 1, sekarang posisinya variabel 1 adalah 2.
- ✓ Dilakukan pengecekan lagi apakah variabel i masih memiliki nilai <=10 ? jika kondisi bernilai benar maka akan terus dicetak kalimat tersebut hingga nilai i lebih dari 10. Ketika kondisinya bernilai false maka program perulangan while akan berhenti.
- ✓ Selesai.

- CONTOH :

```
#include <conio.h>
#include <iostream>
using namespace std;
int main() {

    int i=1;
    while (i<=10)
    {
        cout<<"Belajar Perulangan While "<<i<<endl;
        i++;
    }
    getch();
}
```

OUTPUT :

```
Belajar Perulangan While 1
Belajar Perulangan While 2
Belajar Perulangan While 3
Belajar Perulangan While 4
Belajar Perulangan While 5
Belajar Perulangan While 6
Belajar Perulangan While 7
Belajar Perulangan While 8
Belajar Perulangan While 9
Belajar Perulangan While 10
```

- Struktur kondisi ” looping ”

- **Statement do...while Loop**

  Hampir sama dengan perulangan **while** hanya perbedaanya adalah pada perulangan **do while** kondisi di cek setelah dilakukan terlebih dahulu (minimal sekali) perulangan.

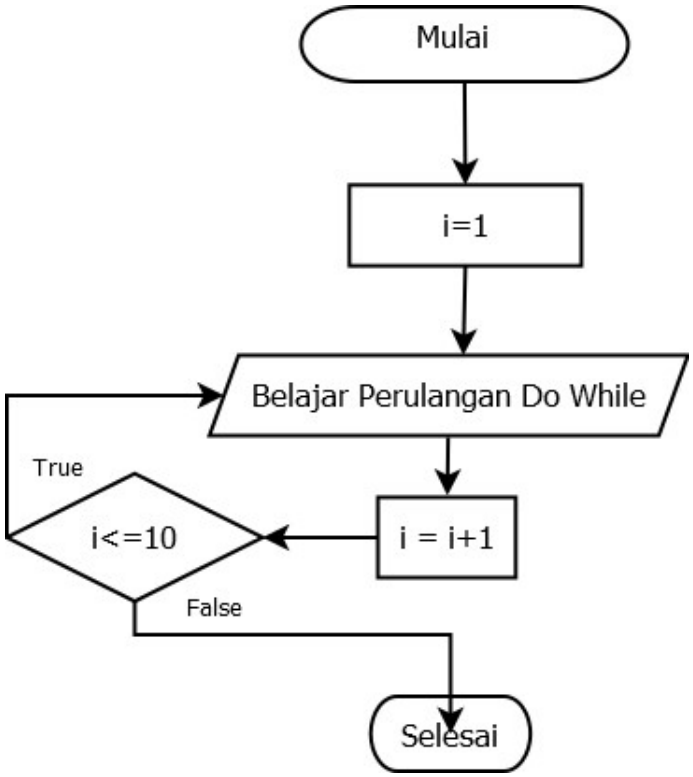
  Hal ini berbeda dengan perulangan **while** yang harus dicek kondisinya terlebih dahulu jika bernilai benar, baru program dijalankan.

  Dengan kata lain perulangan **Do While** merupakan kebalikan dari perulangan **while**.

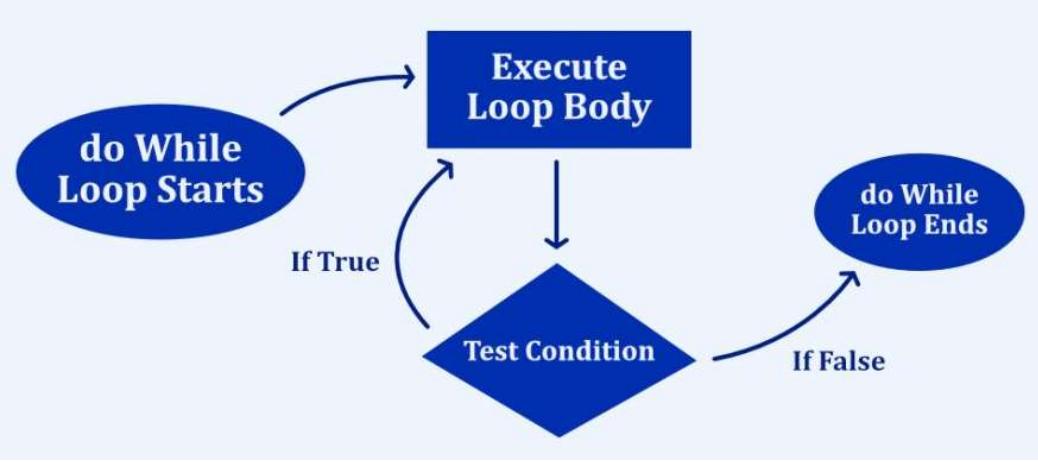
  Berikut umum Perulangan **do...while** :

```
do {
     perintah yang dilakukan perulangan;
   }
while (kondisi);
```

- CONTOH :



- ✓ Mulai
- ✓ Inisialisasi nilai awal i=1
- ✓ Dicetak kalimat "Belajar Perulangan Do While".
- ✓ Nilai variabel i ditambahkan 1.
- ✓ Dicek kondisi apakah nilai variabel i<=10 jika kondisi bernilai benar maka akan dicetak kembali kalimat tersebut dan nilai i ditambahkan 1 lagi.
- ✓ Kemudian akan dicek kembali sesuai dengan kondisi, jika masih bernilai benar maka akan terus dilakukan perulangan hingga kondisi bernilai salah.
- ✓ Selesai.



```cpp
#include <conio.h>
#include <iostream>
using namespace std;
int main() {

    int i=1;

    do {
        cout<<"Belajar Perulangan Do While "<<i<<endl;
        i++;
    } while (i<=10);
    getch();
}
```

```
Belajar Perulangan Do While 1
Belajar Perulangan Do While 2
Belajar Perulangan Do While 3
Belajar Perulangan Do While 4
Belajar Perulangan Do While 5
Belajar Perulangan Do While 6
Belajar Perulangan Do While 7
Belajar Perulangan Do While 8
Belajar Perulangan Do While 9
Belajar Perulangan Do While 10
```

- Latihan

```
#include <conio.h>
#include <iostream>
using namespace std;
int main() {

    int j=1, m, d;

    do {
      m= j *60;
          d=m*60;
      cout<<j<<" Jam "<<m<<" Menit "<<d<<" Detik"<<endl;
      j +=1;
    } while (j <=10);
    getch();
}
```

```
1 Jam 60 Menit 3600 Detik
2 Jam 120 Menit 7200 Detik
3 Jam 180 Menit 10800 Detik
4 Jam 240 Menit 14400 Detik
5 Jam 300 Menit 18000 Detik
6 Jam 360 Menit 21600 Detik
7 Jam 420 Menit 25200 Detik
8 Jam 480 Menit 28800 Detik
9 Jam 540 Menit 32400 Detik
10 Jam 600 Menit 36000 Detik
```

- Perbedaan antara " while & do...while "

Untuk membuktikannya - lihat contoh program berikut ini :

```
1  #include <conio.h>
2  #include <iostream>
3  using namespace std;
4  int main() {
5
6      int i=1;
7
8         //Perulangan While
9      while (i<1)
10     {
11         cout<<"Belajar Perulangan While "<<i<<endl;
12         i++;
13     }
14
15         //Perulangan Do While
16     do {
17          cout<<"Belajar Perulangan Do While "<<i<<endl;
18          i++;
19     } while (i<1);
20
21     getch();
22 }
```

```
Belajar Perulangan Do While 1
```

Karena perulangan **Do while** pengecekan kondisinya diakhir statement, artinya bahwa perulangan ini mengerjakan terlebih dahulu statement yang ada, baru kemudian di cek kondisinya,

karena bernilai salah *(false)* maka perulangan dihentikan, dan hanya mencetak 1 kali saja.

Berbeda dengan perulangan **While** yang dilakukan pengecekan diawal statement, sehingga ketika kondisi bernilai salah *(false)* maka dia tidak akan mengerjakan statement yang ada.

- Perbedaan antara ” **while & do...while** ”

Pada perulangan **while** pengecekan kondisi dilakukan di awal statemen, **while** adalah statemen yang digunakan untuk mengulangi sejumlah perintah selama *conditional expression* bernilai ***True***.

Sementara pada perulangan **do while** pengecekan kondisinya di akhir *statement*, Statemen **do** akan mengeksekusi perintah, kemudian **while** akan memeriksa *conditional expression*.

Jadi perulangan ’**do...while’** akan *mengecek kondisi di belakang (sesudah mengulang),* sedangkan ’**while’** akan *mengecek kondisi di depan atau awal (sebelum mengulang).*

• Perbedaan antara ” while & do...while ”

4 Perbedaan mendasar struktur perulangan while dan do-while:

1. Pada struktur perulangan **while** , pengecekan kondisi dilakukan di awal blok / pengecekan kondisi dilakukan sebelum eksekusi statemen.
2. Pada struktur perulangan **do-while** , pengecekan kondisi dilakukan di akhir blok / pengecekan kondisi dilakukan setelah eksekusi statemen.
3. Pada struktur perulangan **while** , jika kondisi yang didefinisikan tidak terpenuhi (bernilai salah) maka statemen-statemen yang terdapat dalam blok perulangan tidak akan pernah dieksekusi oleh program.
4. Pada struktur perulangan **do-while** , jika kondisi yang didefinisikan tidak terpenuhi (bernilai salah) maka tetap akan melakukan satu kali eksekusi statemen-statemen yang terdapat dalam blok perulangan.

- Struktur kondisi ” looping ”

- **Statement Nested For Loop**

Di dalam blok perulangan, kita juga dapat membuat perulangan.

Ini disebut dengan **Nested For loop** (perulangan For bersarang) atau perulangan di dalam perulangan.

Hal ini membuat eksekusi kode lebih kompleks namun memberi kita kebebasan untuk melakukan tugas yang lebih rumit.

Struktur Perulangan For Bersarang:

```
for(initialization; condition; update) {
    for(initialization; condition; update) {
        // code to be executed
    }
}
```

Mari kita coba lihat contohnya :

```cpp
#include <iostream>
using namespace std;

int main() {
    int rows = 5; // Menentukan jumlah baris
    for(int i = 1; i <= rows; i++) {
        for(int j = 1; j <= i; j++) {
            cout << "*";
        }
        cout << endl;  // Baris baru setelah setiap iterasi loop dalam
    }
    return 0;
}
```

Kita memiliki dua loop for, yaitu :

⇨ **Loop luar** : **(for(int i = 1; i <= rows; i++))** mengontrol jumlah baris yang dicetak,

⇨ **Loop dalam** : **(for(int j = 1; j <= i; j++))** mengontrol jumlah asterisk (*) yang dicetak dalam setiap baris.

Setiap kali loop dalam selesai, kita menambahkan baris baru dengan **cout << endl;**.

Kita lihat contoh lainnya :

```cpp
#include <iostream>
using namespace std;

int main(){
    for(int i = 0; i < 10; i++){
        for(int j = 0; j < 10; j++){
            printf("Perulangan ke (%d, %d)\n", i, j);
        }
    }

    return 0;
}
```

```
Perulangan ke (8, 7)
Perulangan ke (8, 8)
Perulangan ke (8, 9)
Perulangan ke (9, 0)
Perulangan ke (9, 1)
Perulangan ke (9, 2)
Perulangan ke (9, 3)
Perulangan ke (9, 4)
Perulangan ke (9, 5)
Perulangan ke (9, 6)
Perulangan ke (9, 7)
Perulangan ke (9, 8)
Perulangan ke (9, 9)
```

Pada perulangan tersebut, kita menggunakan dua perulangan **for**.

Perulangan pertama menggunakan **variabel i** sebagai *counter*,
sedangkan Perulangan kedua menggunakan **variable j** sebagai *counter*.

Keuntungan Menggunakan Perulangan For Bersarang (Nested For Loop)

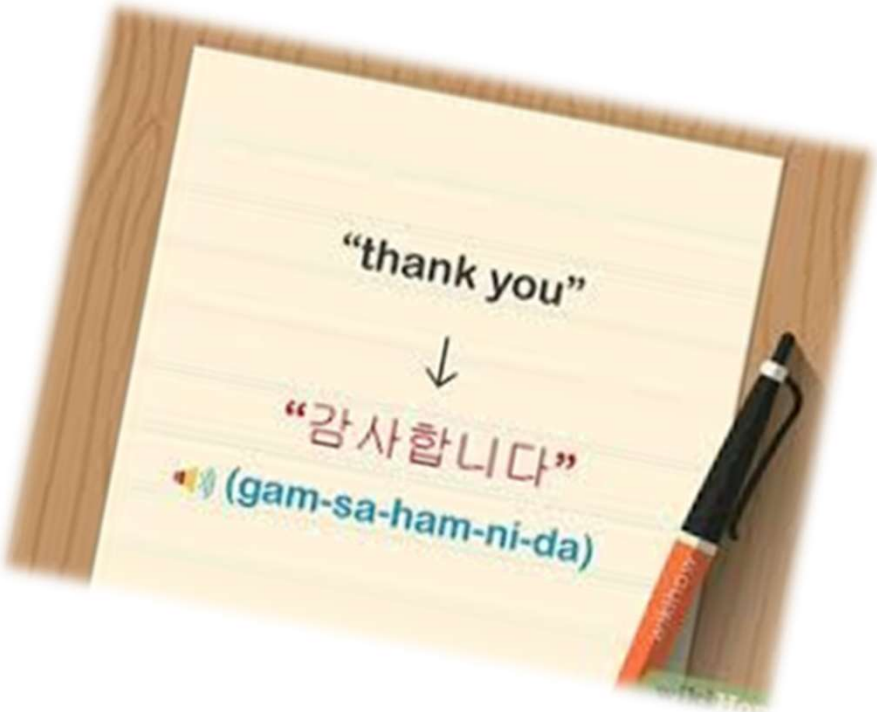
1. Efisiensi Kode: Menyederhanakan kode yang membutuhkan loop dalam loop.
2. Optimasi Tugas: Memungkinkan kita untuk menyelesaikan tugas yang membutuhkan beberapa parameter atau dimensi.
3. Kontrol Tingkat Tinggi: Memberikan kontrol tingkat tinggi terhadap alur program.

Memahami **Perulangan For Bersarang di C++** adalah penting bagi setiap programmer yang ingin menulis kode yang efisien dan efektif. Dengan penggunaan yang tepat, nested for loops bisa menjadi alat yang sangat kuat di tangan programmer yang cakap.

Kesimpulan

Perbedaan antara FOR, WHILE, dan DO-WHILE :

- ***For :*** untuk mengulang suatu proses yang telah diketahui jumlahnya.
- ***While : Pre Tested Loop*** untuk mengulang suatu proses yang belum diketahui jumlahnya. Pengecekan kondisi akan dilakukan terlebih dahulu. Jika kondisi masih bernilai true, maka looping akan terus berlanjut.
- ***Do-while : Post Tested Loop*** untuk mengulang suatu proses yang belum diketahui jumlahnya. Instruksi akan dijalankan lebih dahulu, kemudian dilakukan pengecekan kondisi apabila masih bernilai true maka looping akan terus berlanjut.


"thank you"
↓
"감사합니다"
(gam-sa-ham-ni-da)