PEMROGRAMAN BERORIENTASI OBJEK

Pengulangan

Telkom University

Budhi Irawan, S.Si, M.T

PENGULANGAN

- ***Pengulangan*** adalah suatu proses yang melakukan perulangan statement-statement dalam sebuah program secara terus-menerus sampai terdapat kondisi untuk menghentikannya.
- Struktur pengulangan akan sangat membantu dalam efisiensi program.

PENGULANGAN

```
#include <iostream>

using namespace std;

int main() {

  cout<<"Saya Sedang Belajar C++"<<endl;
  cout<<"Saya Sedang Belajar C++"<<endl;
  cout<<"Saya Sedang Belajar C++"<<endl;
  cout<<"Saya Sedang Belajar C++"<<endl;
  cout<<"Saya Sedang Belajar C++"<<endl;
  cout<<"Saya Sedang Belajar C++"<<endl;
  cout<<"Saya Sedang Belajar C++"<<endl;
  cout<<"Saya Sedang Belajar C++"<<endl;
  return 0;
}
```

PENGULANGAN

Dalam bahasa C++ terdapat *tiga jenis struktur pengulangan* yaitu :

- struktur **`for`**,
- struktur **`while`** dan
- struktur **`do-while`**.

STRUKTUR FOR

- Struktur ***pengulangan*** **FOR** biasanya digunakan untuk melakukan ***pengulangan*** yang telah diketahui banyaknya.
- Jenis ini merupakan jenis struktur ***pengulangan*** yang paling mudah dipahami.

STRUKTUR FOR

- Untuk melakukan ***pengulangan*** dengan menggunakan struktur ini, kita harus memiliki sebuah ***variabel indeknya***.
- Tipe data dari ***variabel*** yang akan digunakan sebagai ***indeks*** haruslah tipe data yang mempunyai urutan yang teratur, misalnya tipe ***data int*** (`0,1,2, ....`) atau ***char*** (`'a', 'b', 'c', ....`)

STRUKTUR FOR

- Bentuk umum dari strukur ***for*** adalah :

```
// Untuk pengulangan yang sifatnya menaik
// (increment)
For (variabel = nilai_awal; kondisi; variabel++){
   statement_yang_akan_diulang;
}

// Untuk pengulangan yang sifatnya menurun
// (decrement)
For (variabel = nilai_awal; kondisi; variabel--){
   statement_yang_akan_diulang;
}
```

STRUKTUR FOR

- Sebagai catatan bahwa jika akan melakukan ***pengulangan*** yang sifatnya *menaik* atau ***increment*** maka *nilai awal* dari ***variabel*** yang kita definisikan haruslah lebih kecil dari *nilai akhir* yang dituliskan dalam kondisi atau ***ekspresi***.
- Sebaliknya jika akan melakukan ***pengulangan*** yang sifatnya *menurun* atau ***decrement*** maka nilai awal harus *lebih besar* dari *nilai akhir*.

STRUKTUR FOR

Pengulangan Menaik Increement

```
#include <iostream>

using namespace std;

int main() {
  int C;
  for (C=0; C<10; C++) {
    cout<<"Saya Sedang Belajar C++"<<endl;
  }

  return 0;
}
```

STRUKTUR FOR

Pengulangan Menurun Decrement

```
#include <iostream>

using namespace std;

int main() {
  int C;
  for (C=10; C>0; C--) {
     cout<<"Saya Sedang Belajar C++"<<endl;
  }

  return 0;
}
```

STRUKTUR FOR

- Secara default, ***struktur for*** akan *menaikan atau menurunkan nilai dari sebuah variabel indeks dengan* ***nilai 1***, namun bagaimana jika ingin menaikan nilai tersebut dengan nilai yang lain ?
- Jawabannya adalah dengan menggantikan operator increment atau decrement dengan statement yang didefinisikan sendiri

STRUKTUR FOR BANYAK VARIABEL

- Struktur **FOR** dalam bahasa C++ dapat juga melibatkan lebih dari satu variabel, namun yang jelas satu diantarannya akan digunakan sebagai ***indeks pengulangan***.
- Untuk memahami konsepnya, berikut disajikan sebuah contoh program dimana didalamnya terdapat struktur **FOR** yang melibatkan 3 buah variabel yaitu : **A, B,** dan **C**

STRUKTUR FOR BANYAK VARIABEL

```
#include <iostream>

using namespace std;

int main() {
  char A;      // variabel A (bertipe char) akan digunakan
               // sebagai indeks pengulangan
  int B;       // variabel B akan digunakan untuk
               // menampung nilai penjumlahan
  int C;       // variabel C akan digunakan untuk
               // menampung nilai perkalian

  for (A='a', B=0, C=1; A<='e'; A++, B=B+5, C=C*10) {
     cout<<"Nilai A = "<<A<<endl;
     cout<<"Nilai B = "<<B<<endl;
     cout<<"Nilai C = "<<C<<endl;
     cout<<endl;
  }

  return 0;
}
```

STRUKTUR FOR BERSARANG

- Sama halnya seperti pada, pada struktur pengulangan juga dapat diterapkan juga ***pengulangan bersarang (nested looping)***.
- Pada bagian ini hanya akan dijelaskan pengulangan bersarang dengan menggunakan struktur ***FOR***.
- Konsepnya sangat sederhana yaitu dalam sebuah ***pengulangan*** terdapat ***pengulangan*** yang lainnya.

STRUKTUR FOR BERSARANG

- Bentuk umum dari ***struktur FOR bersarang*** dapat dirumuskan sebagai berikut :

```
for(variabel1=nilai_awal; kondisi1; variabel1++{
for(variabel2=nilai_awal; kondisi2; variabel2++{
for(variabel3=nilai_awal; kondisi3; variabel3++{
Statement-Statemen_yang_akan_diulang;
....
}
}
}
```

STRUKTUR FOR BERSARANG

- Jika kita lihat dari rumusan diatas, pada setiap ***pengulangan pertama*** program akan menyelesaikan ***pengulangan kedua***.
- Begitu juga pada setiap ***pengulangan kedua***, program akan menyelesaikan ***pengulangan ketiga***, begitu seterusnya.
- Untuk lebih jelasnya, contoh program berikut didalamnya terdapat dua s***truktur FOR bersarang***.

STRUKTUR WHILE

- ***Struktur Pengulangan*** jenis ini adalah pengulangan yang melakukan *pemeriksaan diawal blok struktur*.
- Seperti sudah diketahui sebelumnya, pengulangan hanya dilakukan *jika kondisi yang didefinisikan didalamnya terpenuhi (bernilai benar)*.

STRUKTUR WHILE

- Hal ini berarti *jika kondisi yang didefinisikan tidak terpenuhi (bernilai salah) maka statemen-statemen yang terdapat dalam blok pengulangan pun tidak akan pernah dieksekusi oleh program*
- Bentuk umum struktur ***pengulangan WHILE*** adalah sebagai berikut :

```
while (kondisi){
Statement_statement_yang_akan_diulang;
}
```

STRUKTUR WHILE

```
#include <iostream>

using namespace std;

int main() {
  int C;  // Mendeklarasikan varaiebl C sebagai
          // indeks pengulangan

  C = 0;  // Melakukan inisialisasi nilai terhadap
          // variabel C

  while (C<10) {
     cout<<"Saya sedang belajar C++"<<endl;
     C++; /* Statemen ini berguna untuk menaikkan nilai,
               dan setelah bernilai 10,
               maka pengulangan akan dihentikan */
  }

  return 0;
}
```

STRUKTUR `WHILE`

- Perlu diperhatikan bahwa untuk melakukan ***pengulangan*** dengan menggunakan ***struktur `WHILE`*** harus berhati-hati dalam menentukan inisialisasi awal dan memanipulasi nilai tersebut suapaya ***pengulangan*** akan ***berhenti*** sesuai dengan yang dinginkan.

STRUKTUR WHILE

- Bagi programmer pemula, hal ini biasanya sering terlupakan sehingga pengulangan akan dilakukan secara terus menerus karena kondisi yang didefinisikan selalu bernilai benar.

STRUKTUR DO-WHILE

- Berbeda dengan ***struktur WHILE*** yang melakukan pemeriksaan kondisi diawal ***blok perulangan***, pada struktur ***DO-WHILE*** kondisi justru ditempatkan ***dibagian akhir***.
- Hal ini menyebabkan ***struktur perulangan*** ini minimal akan melakukan ***satu kali proses*** walaupun kondisi yang didefinisikan tidak terpenuhi ***(bernilai salah)***

STRUKTUR DO-WHILE

- Berikut ini adalah bentuk umum penulisan dari struktur ***DO-WHILE*** :

```
do{
Statement_yang_akan_diulang;
}while (kondisi);
```

STRUKTUR DO-WHILE

- Sama seperti lainnya, ***struktur pengulangan*** jenis ini juga dapat digunakan untuk kasus-kasus diatas, hanya saja kita harus teliti dan berhati-hati dalam mendefinisikan kondisi yang terdapat didalamnya.
- Berikut ini contoh program yang diambil dari kasus sebelumnya, namun disini kita akan menggunakan ***struktur DO-WHILE***.

STRUKTUR DO-WHILE

```
#include <iostream>

using namespace std;

int main() {

  int C = 0;

  do {
    cout<<"Saya Sedang Belajar C++"<<endl;
    C++;
  } while (C < 10);

  return 0;
}
```

STRUKTUR DO-WHILE

- Contoh lain yang dapat digunakan untuk mengimplementasikan jenis srtuktur ***DO-WHILE*** adalah pembuatan program untuk *menentukan nilai faktor persekutuan terbesar dari dua buah bilangan bulat*.
- Misalnya dimasukan dua buah bilangan bulat yaitu 8 dan 12, maka faktor persekutuan terbesar dari kedua bilangan tersebut adalah bisa dilihat dari program berikut :

STRUKTUR DO-WHILE

```
#include <iostream>
using namespace std;
int main() {
  int Bil1, Bil2;
  int sisa;

  cout<<"Masukkan bilangan pertama  : "; cin>>Bil1;
  cout<<"Masukkan bilangan kedua    : "; cin>>Bil2;

  // Melakukan pertukaran nilai
  if (Bil1 < Bil2) {
    int temp = Bil1;
    Bil1 = Bil2;
    Bil2 = temp;
  }

  do {
    sisa = Bil1 % Bil2;
    Bil1 = Bil2;
    Bil2 = sisa;
  } while (sisa != 0);

  cout<<"\nFaktor persekutuan terbesar = "<<Bil1;

  return 0;
}
```

STATEMENT PELONCATAN

- Pada saat menggunakan ***struktur pengulangan***, seringkali dituntut untuk melakukan ***peloncatan statement***.
- Kata peloncatan artinya memaksa agar ***eksekusi statement*** berjalan sesuai urutan yang diinginkan, yaitu dengan cara ***meloncat*** dari ***statement*** yang satu ke ***statement*** yang lain.

STATEMENT PELONCATAN

Dalam bahasa C++ terdapat 4 perintah yang berguna untuk melakukan hal ini yaitu :

- **`Break`**
- **`Continue`**
- **`Goto`**
- **`Exit()`**

KEYWORD BREAK

- Keyword ini berfungsi untuk menghentikan proses ***pengulangan*** dan program akan langsung ***meloncat*** ke ***statemen*** yang berada di bawah blok pengulangan yang bersangkutan.
- Untuk dapat memahami penggunaannya, disini kita akan membuat contoh program yang dapat menuliskan teks ”Contoh Pengulangan dalam C++” sebanyak 10x

KEYWORD BREAK

- Namun perlu diperhatikan bahwa disini akan dipaksa ***proses pengulangan*** dengan mendefinisikan kondisi yang selalu ***bernilai benar.***
- Struktur pengulangan yang dipilih untuk menjawab kasus ini adalah ***struktur `WHILE`***.

KEYWORD BREAK

```
#include <iostream>

using namespace std;

// Mendeklarasikan tipe baru yang hanya memiliki
// nilai FALSE dan TRUE
enum BOOLEAN { FALSE, TRUE };

int main() {
  // Mendeklarasikan variabel sebagai indeks pengulangan
  // dan diisi dengan nilai 0
  int C = 0;

  while (TRUE) {
    cout<<"Contoh pengulangan dalam C++"<<endl;
    if (C == 9) break;
    C++;
  }
  cout<<"Nilai C = "<<C;

  return 0;
}
```

KEYWORD BREAK

- Statement ***WHILE (TRUE)*** merupakan statement yang memaksa program untuk selalu melakukan ***pengulangan***, sehingga program tersebut harus memiliki statement yang dapat menghentikan pengulangan tersebut.
- Untuk melakukan hal ini, maka digunakan ***keyword BREAK***.

KEYWORD BREAK

- Karena ingin dilakukan ***pengulangan*** sebanyak 10x maka harus ditempatkan ***keyword BREAK*** pada saat variabel *C* bernilai *9*.
- Kenapa *9* ? Jawabnya karena nilai ***indeks*** yang didefinisikan dimulai dari *0*.

KEYWORD BREAK

- Setelah itu program akan meloncat langsung ke ***statement*** yang terdapat di bawah ***blok pengulangan*** tanpa mengeksekusi lagi statement `C++;` (baris program yang berada dibawah ***keyword BREAK***) yang terdapat didalam ***blok pengulangan***.

KEYWORD CONTINUE

- ***Keyword*** ***`CONTINUE`*** berfungsi untuk melanjutkan ***proses pengulangan*** yang akan menyebabkan program meloncat ***ke statement awal*** yang terdapat dalam ***blok pengulangan***.
- Dengan kata lain, ***keyword*** ini akan menyebabkan program meloncat ke ***statement*** awal yang terdapat dalam ***pengulangan***.

KEYWORD CONTINUE

```
#include <iostream>

using namespace std;

enum BOOLEAN { FALSE, TRUE };

int main() {
  int X;

  BOOLEAN STOP = FALSE;
  while (!STOP) {
    cout<<"Masukkan nilai X : "; cin>>X;
    if (X <= 0) {
      cout<<"Nilai x tidak boleh negatif atau 0\n";
      continue;
    }
    STOP = TRUE;  // akan menghentikan pengulangan
  }

  if (STOP) {
    cout<<"Anda telah memasukkan nilai "<<X;
  }

  return 0;
}
```

KEYWORD CONTINUE

- Contoh lain yang dapat diambil untuk menunjukan cara kerja ***statement CONTINUE*** adalah program untuk mencetak bilangan genap dari rentang tertentu. Adapun kode programnya adalah sebagai berikut :

KEYWORD GOTO

- Penggunaan ***keyword GOTO*** ini dapat mewakili penggunaan ***keyword BREAK*** maupun ***CONTINUE,*** tergantung dimana menempatkan label.
- Label ini dalam kode program berfungsi untuk menyatakan lokasi yang akan dituju.
- Perlu diperhatikan bahwa untuk menuliskan suatu label harus menggunakan tanda titik dua (:) dibelakang nama label tersebut.

KEYWORD GOTO

- Dalam mendefinisikan nama label juga tidak perlu dilakukan pendeklarasian seperti halnya sebuah variabel karena label hanya digunakan untuk tanda saja sehingga tidak memiliki tipe data.

KEYWORD GOTO

```
#include <iostream>

using namespace std;

enum BOOLEAN { FALSE, TRUE };

int main() {
  int C = 0;

  while (TRUE) {
    cout<<C+1<<endl;
    if (C == 9) {
       goto MyLabel;    // Meloncat ke label MyLabel
    }
    C++;
  }

  MyLabel:  // Lokasi yang akan dituju

  return 0;
}
```

KEYWORD GOTO

- Jika nanti secara teliti, penggunaan ***keyword GOTO*** pada kasus ini sama halnya seperti penggunaan ***keyword BREAK***
- ***Keyword GOTO*** tidak hanya dapat digunakan dalam ***struktur pengulangan*** saja, melainkan dapat digunakan dimana saja.

KEYWORD EXIT ()

- ***Keyword*** ***`EXIT()`*** berfungsi untuk proses ***terminasi*** atau ***keluar*** dari program.

KEYWORD EXIT ()

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

enum BOOLEAN { FALSE, TRUE };

int main() {

  float X = 1, Y;

  while (TRUE) {
    cout<<"Masukkan nilai Y : "; cin>>Y;

    // Menghindari pembagian dengan NOL
    if (Y == 0) {
       cout<<"Nilai Y tidak boleh 0";
       exit(0);     // Keluar dari program
    }
    break;  // Keluar dari pengulangan
  }

  float Z = X / Y;
  cout<<X<<"/"<<Y<<" = "<<Z;

  return 0;
}
```

KEYWORD EXIT ()

- ***Parameter 0*** yang terdapat pada fungsi ***exit()*** di atas berfungsi untuk melakukan ***terminasi*** program secara normal.
- Parameter lain yang dapat dilewatkan kedalam fungsi ini adalah ***EXIT_SUCCESS*** (sama dengan 0) dan ***EXIT_FAILURE*** (nilai selain 0)
- Untuk menggunakan fungsi `exit()` perlu mendaftarkan file header ***<cstdlib>***.

KEYWORD EXIT ()

- Pada saat user memasukan ***nilai 0*** ke dalam ***variabel Y,*** program akan mengeksekusi fungsi ***exit()***.
- Ini menyebabkan ***program dihentikan***, namun jika nilai yang dimasukan adalah nilai selain ***0,*** maka program akan mengeksekusi statement ***BREAK***.
- Pada saat ini proses pengulangan akan dihentikan dan program akan mengeksekusi statemen-statemen yang terdapat dibawah ***blok pengulangan***.