PEMROGRAMAN BERORIENTASI OBJEK

Komentar, Identifier, Konstanta dan Variabel

Telkom University

Budhi Irawan, S.Si, M.T

KOMENTAR PROGRAM

- Dalam proses pengembangan sebuah program, pasti akan disibukan dengan penulisan kode-kode yang begitu banyak dan tampak rumit sehingga akan sulit untuk dipahami oleh orang lain.
- Untuk menangani masalah ini, sebagai programmer sebaiknya menambahkan ***komentar*** untuk menjelaskan algoritma dan keterangan-keterangan yang diperlukan dalam program.

KOMENTAR PROGRAM

Fungsi dari ***komentar*** program adalah untuk menuliskan informasi tentang kode program misalnya :

- Nama Pembuat Kode Program
- Kapan Kode tersebut dibuat atau dimodifikasi
- Instansi Pembuat Program
- Deskripsi program
- Tahun Pembuatan
- Dan lain-lain

PENGGUNAAN TANDA //

- **Tanda ini digunakan untuk menuliskan *komentar* yang banyaknya hanya satu baris**

  Contoh penggunaannya :

```
// Ini adalah komentar untuk satu baris
```

- **Apabila *komentar* dituliskan dua baris menggunakan tanda // maka tidak akan dianggap sebagai *komentar*, melainkan dianggap sebagai variabel yang tidak dikenal**

  Contoh :

```
// Ini adalah komentar
   untuk satu baris
```

PENGGUNAAN TANDA //

- **Selain itu perlu hati-hati dalam menggunakan tanda // karena tanda ini tidak dapat digunakan untuk *komentar* yang bersifat sisipan**

  Contoh :

```
//  Pendeklarasian variabel X;
```

- **Penulisan *komentar* seperti diatas juga akan menyebabkan kesalahan karena *x;* akan dianggap sebagai *komentar* dan tidak akan berperan sebagai variabel.**

  Penulisan yang benar adalah :

```
int x;  // Pendeklarasian variabel
```

PENGGUNAAN TANDA /*...*/

- **Tanda ini digunakan untuk menuliskan *komentar* yang banyaknya satu baris atau lebih. Komentar dimulai dengan tanda `/* sampai ditemukan tanda */`**

  Contoh :

```
/* Ini adalah komentar untuk satu baris  */
/* Ini adalah komentar yang lebih dari satu
   baris panjangnya dan bisa juga ditulis
   sebanyak2nya jika mau menuliskannya */
```

- **Dengan menggunakan tanda ini, dapat pula dituliskan *komentar* yang bersifat sisipan.**

  Contoh :

```
int /* ini adalah varibel */  x;
```

PENGGUNAAN TANDA /*...*/

- Pengecualian bahwa dalam hal ini tidak bisa dibuat komentar yang bersarang *(nested comment)*
  Contoh :

```
/* Ini adalah /* nested */ yang bersarang */
```

- Secara sepintas penulisan diatas seperti benar, namun sebenarnya penulisan seperti itu adalah salah karena ada tanda */ yang ditemukan pertama kali akan dianggap sebagai penutup tanda /* pertama. Dengan alasan ini maka untuk kasus diatas dianggap sebagai komentar adalah teks "`Ini adalah /* nested`" sedangkan teks yang terdapat setelah tanda */ yaitu teks "`yang bersarang */`" akan tetap dianggap sebagai bagian program dan akan dibaca saat proses kompilasi.

PENGGUNAAN TANDA /*...*/

- Jenis komentar seperti ini sering dijumpai pada bagian awal kode program seperti contoh berikut ini

```
/*************************************
 Nama file     : latihan1.cpp
 Oleh          : Si Plus Dua
 Dibuat        : 28 Agustus 2017
 Dimodifikasi  : 03 Oktober 2017
 Deskripsi     : Latihan Membuat Komentar
*************************************/
```

IDENTIFIER

- ***Identifier*** adalah suatu **pengenal** atau **pengidentifikasi** yang **dideklarasikan** agar ***compiler*** dapat mengenalinya.
- ***Identifier*** dapat berupa **nama variabel**, **konstanta**, **fungsi**, **kelas**, **template**, maupun **namespace**.
- ***Identifier*** yang berperan sebagai ***variabel*** dan ***konstanta*** berfungsi untuk menampung sebuah nilai yang digunakan dalam program.

IDENTIFIER

- Identifikasi ini dilakukan untuk mempermudah proses penanganan data atau nilai. Misalnya untuk memasukan dan menampilkan nilai. Contoh :

```
#include <iostream>

using namespace std;

int main()
{
  char Nama[20];
  int Usia;
  cout<<"Masukkan Nama Panggilan  : "; cin>>Nama;
  cout<<"Masukkan Usia : "; cin>>Usia;
  cout<<Nama<<'\n';
  cout<<Usia;

  return 0;
}
```

IDENTIFIER

- Pada program diatas terdapat dua buah ***identifier*** yaitu **Nama** dan **Usia**;
- Pada saat program dijalankan, identifier tersebut akan digunakan untuk menyimpan nilai yang dimasukan dari keyboard.
- Dalam library **iostream** C++ , standard operasi input dan output untuk pemrograman didukung oleh dua data streams : **cin** dengan operator **>>** untuk input dan **cout** dengan operator **<<** untuk output.

IDENTIFIER

- ***Identifier*** tidak boleh atau diawali dengan karakter yang berupa angka

  Contoh :

  `long 50;` : **SALAH karena Identifier berupa angka**

  `long 21x;` : **SALAH karena Identifier diawali oleh karakter berupa angka**

  `long x5;` : **BENAR karena Identifier tidak diawali** oleh angka

IDENTIFIER

- ***Identifier*** tidak boleh mengandung spasi

  Contoh :

```
Int Bilangan Bulat;      : SALAH karena mengandung spasi
Int Bilangan_Bulat;      : BENAR
Int BilanganBulat;       : BENAR
Int _BilanganBulat;      : BENAR
```

IDENTIFIER

- ***Identifier*** tidak boleh menggunakan karakter-karakter simbol (#, @, ?, !, $, dll)

  Contoh :

```
Int !satu;    : SALAH
Int dua@;     : SALAH
Int ti#ga;    : SALAH
```

IDENTIFIER

- ***Identifier*** tidak boleh menggunakan keyword yang terdapat pada C++
  Contoh :

  `long break;` : **SALAH ->** Break adalah keyword
  `Int return;` : **SALAH ->** return adalah keyword

- Nama ***Identifier*** sebaiknya disesuaikan dengan jangan sampai orang lain bingung hanya karena salah dalam penamaan ***identifier***.

KONSTANTA

- **Konstanta** adalah jenis ***identifier*** yang bersifat konstan atau tetap, artinya nilai dari ***konstanta*** didalam program tidak bisa diubah.
- ***Konstanta*** berguna untuk menentukan nilai yang merupakan tetapan, misalnya nilai **pi (π)**, **kecepatan cahaya** dan yang lainnya.
- Dalam C++, terdapat dua buah cara untuk membuat sebuah ***konstanta***, yaitu dengan menggunakan preprocessor directive `#define` dan menggunakan keyword `const`

KONSTANTA

- **Preprocessor Directive** merupakan sebuah baris kode perintah untuk preprocessor (perintah awal).
- Baris kode ini bukan merupakan kode statement, tetapi baris perintah untuk preprocessor sebelum mengeksekusi kode program.
- Perintah ini (Directives) selalu diawali dengan tanda hash (#) dan hanya satu line saja.
- Boleh memakai " ; " boleh tidak, tetapi mungkin pengguna Compiler lain tidak diizinkan menggunakan " ;".
- Perintah baris dengan tanda # inilah yang akan dikerjakan terlebih dahulu sebelum memroses/mengeksekusi kode.

KONSTANTA

- Menggunakan Preprocesor Directive :

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

#define MAX 5 // tanpa titik koma

int main() {
int A[MAX];

    for (int C=0; C<MAX; C++) {
    // mengisi nilai ke dalam A[C]
    A[C] = C * 10;
    // menampilkan nilai A[C]
    cout<<A[C]<<endl;
  }
 return 0;
}
```

KONSTANTA

Menggunakan Keyword `const`

- Bentuk umum penulisannya :

```
const tipe_data nama_constanta = nilai tetapan;
```

- Contoh Pendeklarasiannya :

```
const double PI = 3,14;
const int NILAI_MAX = 100;
const char MyChar = 'A';
```

KONSTANTA

- Menggunakan Keyword Const :

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

const int MAX = 5; // berikan tanda titik dua

int main() {
  int A[MAX];
  for (int C=0; C<MAX; C++) {
    // mengisi nilai ke dalam A[C]
    A[C] = C * 10;
    // menampilkan nilai A[C]
    cout<<A[C]<<endl;
  }

  return 0;
}
```

VARIABEL

- Berbeda dengan ***konstanta*** yang mempunyai nilai tetap, ***variabel*** adalah sebuah ***identifier*** yang mempunyai nilai dinamis.
- Arti kata dinamis adalah bahwa nilai ***variabel*** tersebut dapat kita ubah sesuai dengan kebutuhan dalam program.
- Berikut ini adalah bentuk umum pendeklarasian ***variabel*** dalam C++

```
tipe_data nama_variabel;
```

VARIABEL

- Contoh :

```
int A;
```

- Pada contoh diatas, dideklarasikan sebuah variabel bertipe **`int`** dengan nama **`A`**.
- Melalui cara seperti ini variabel tersebut sudah dapat digunakan untuk menampung nilai-nilai berupa bilangan bulat (sesuai dengan rentang yang ada pada tipe data integer)

VARIABEL

- Apabila akan mendeklarasikan beberapa ***variabel*** yang bertipe sama maka dapat menyingkat penulisannya dengan menggunakan bentuk :

```
Tipe_data nama_variabel1, nama_variabel2, nama_variabel3;
```

- Contoh :

```
int A, B, C;
```

INISIALISASI VARIABEL

- Dalam konteks ini, inisialisasi dapat didefinisikan sebagai proses pengisian nilai awal (nilai default) ke dalam suatu ***variabel***.
- Dalam C++, pengisian nilai dilakukan dengan menggunakan operator yang sama yaitu operator sama dengan (=)

INISIALISASI VARIABEL

- Bentuk umum untuk inisialisasi variabel adalah sebagai berikut :

```
tipe_data nama_variabel = nilai_awal;
```

contoh :

```
int A=9;
```

- Artinya dilakukan proses inisialisasi terhadap variabel **A** dengan nilai **9**

INISIALISASI VARIABEL

- Apabila ingin melakukan inisialisasi terhadap lebih dari satu **variabel** maka bentuk umum penulisannya adalah :

```
tipe_data nama_variabel1 = nilai_awal1, ......
...... tipe_data nama_variabel2 = nilai_awal2;
```

- Contoh :

```
int A=10, B=15, C=20;
```

- Artinya dilakukan proses inisialisasi terhadap **variabel `A`** akan diisi dengan nilai awal `10`, `B` dengan nilai `15` dan `C` diisi dengan nilai `25`.

INISIALISASI VARIABEL

- Inisialisasi nilai tidak harus dilakukan untuk semua ***variabel*** yang ada seperti yang ditunjukan pada kode berikut :

```
int A, B=15, C;
```

- Kali ini hanya variabel **B** yang diisi dengan nilai awal.

CONTOH TANPA DAN DENGAN INISIALISASI VARIABEL

```
#include <iostream>

using namespace std;

int main() {
  int A, B=15, C, D=20 ;

  cout<<"Nilai A Tanpa Inisialisasi Nilai : "<<A<<endl;
  cout<<"Nilai B Dengan Inisialisasi Nilai : "<<B<<endl;
  cout<<"Nilai C Tanpa Inisialisasi Nilai : "<<C<<endl;
  cout<<"Nilai D Dengan Inisialisasi Nilai : "<<D<<endl;

  return 0;
}
```

VARIABEL GLOBAL DAN VARIABEL LOKAL

- Berdasarkan ruang lingkupnya, ***variabel*** dibedakan menjadi dua yaitu ***Variabel Global*** dan ***Variabel Lokal***.
- Penentuan ***variabel*** untuk dijadikan sebagai ***variabel global*** ataupun ***variabel lokal*** tentu akan sangat tergantung kepada kasus program yang dihadapinya.

VARIABEL GLOBAL

- Apabila didalam kode program membutuhkan sebuah ***variabel*** yang dapat dikenali oleh semua lingkungan dalam program yang dibuat, maka ***variabel*** tersebut harus dideklarasikan sebagai ***variabel*** yang bersifat global
- Program dalam bahasa C++ selalu terdapat fungsi utama dengan nama **`main()`**
- Apabila dideklarasikan sebuah variabel diluar **`main()`** (atau fungsi lain) maka dengan sendirinya ***compiler*** akan menganggap variabel tersebut sebagai variabel global.

VARIABEL GLOBAL

```
#include <iostream>
using namespace std;
int A;   // Variabel A adalah variabel global
           // karena dideklarasikan di luar fungsi
// Membuat fungsi test()
void test() {
  // Mengisikan (assign) nilai ke dalam variabel A
  A = 20;
  cout<<"Nilai A di dalam fungsi test(): "<<A<<endl;
}

// Membuat fungsi main() atau fungsi utama
int main() {
  // Mengisikan (assign) nilai ke dalam variabel A
  A = 10;
  cout<<"Nilai A di dalam fungsi main(): "<<A<<endl;

  // Memanggil fungsi test()
  test();
  return 0;
}
```

VARIABEL GLOBAL

- Pada kode diatas, **variabel A** dideklarasikan sebagai ***variabel global.***
- Proses deklarasi ***variabel global*** harus dilakukan ***di luar fungsi.***
- Melalui cara seperti ini, **variabel A** akan dikenal oleh semua fungsi yang ada didalam program, yaitu **`test()`** dan **`main()`**

VARIABEL LOKAL

- ***Variabel lokal*** adalah ***variabel*** yang hanya dikenal oleh suatu fungsi saja.
- Proses deklarasi ***variabel lokal*** dilakukan didalam lingkup fungsi yang dimaksud.
- Pada contoh berikutnya ***Variabel A*** dideklarasikan dalam fungsi **`test()`**, dengan demikian **A** akan bersifat lokal dan hanya bisa diakses oleh fungsi **`test()`** dan tidak diizinkan untuk mengakses variabel tersebut dari dalam fungsi **`main()`**

VARIABEL LOKAL

```
#include <iostream>
using namespace std;
// Membuat fungsi test()
void test() {
  int A; // A bersifat lokal
         // dan hanya dikenal oleh fungsi test()
  A = 20;
  cout<<"Nilai A di dalam fungsi test(): "<<A<<endl;
}

// Membuat fungsi main() atau fungsi utama
int main() {

  // Memanggil fungsi test()
  test();
  return 0;
}
```

KET. TAMBAHAN

USING NAMESPACE STD

- **using** adalah deklarasi arahan / panggilan / pemberitahukan kepada kompiler untuk penggunaan deklarasi namespace dan anggota namespace.
- **namespace** adalah penyedia metode untuk mencegah konflik nama dalam proyek-proyek besar. Simbol pendeklarasian di dalam blok namespace ditempatkan di lingkup bernama yang akan mencegah mereka pada keliruan pengenalan pada scope yang lainnya. Jika pendeklarasiannya seperti diatas, itu berfungsi untuk memanggil namespace yang telah dibuat.
- **std** adalah nama namespace tersebut yang sudah tersedia dan tidak perlu dibuat ulang menggunakan fungsi namespace, cukup dipanggil. Std merupakan wadah urutan yang merangkum ukuran data dan array dinamis. Di dalam std terdapat halnya seperti cout, cin, endl dan lain-lain.

INCLUDE

- Include adalah salah satu pengarah prepocessor directive yang tersedia pada C++. Preprocessor selalu dijalankan terlebih dahulu pada saat proses kompilasi terjadi. Bentuk umumnya :

```
# include <nama file>
```

- Tidak diakhiri dengan tanda semicolon, karena bentuk tersebut bukanlah suatu bentuk pernyataan, tetapi merupakan prepocessor directive. Baris tersebut menginstrusikan kepada kompiler untuk menyisipkan file lain dalam hal ini file yang berakhiran .h (file header) yaitu file yang berisi C++ standard library.

INCLUDE

- `# include <iostream.h>` : diperlukan pada program yang melibatkan objek **cout** dan **cin**
- `# include <conio.h>` : diperlukan bila melibatkan **clrscr( )**, yaitu perintah untuk membersihkan layar dan fungsi **getch**( ) untuk menerima sembarang input keyboard dari user.
- `# include <iomanip.h>` : diperlukan bila melibatkan **setw( )** yang bermanfaat untuk mengatur lebar dari suatu tampilan data.
- `# include <math.h>` : diperlukan pada program yang menggunakan operasi **sqrt( )** yang bermanfaat untuk operasi matematika kuadrat.

INCLUDE

- Include adalah salah satu pengarah prepocessor directive yang tersedia pada C++. Preprocessor selalu dijalankan terlebih dahulu pada saat proses kompilasi terjadi. Bentuk umumnya :

```
# include <nama file>
```

- Tidak diakhiri dengan tanda semicolon, karena bentuk tersebut bukanlah suatu bentuk pernyataan, tetapi merupakan prepocessor directive. Baris tersebut menginstrusikan kepada kompiler untuk menyisipkan file lain dalam hal ini file yang berakhiran .h (file header) yaitu file yang berisi C++ standard library.

FUNGSI MAIN

- Program C++ terdiri dari satu atau lebih fungsi, dan di antara salah satunya harus ada fungsi main dan hanya boleh ada satu **main** pada tiap program C++.
- Setiap program C++ akan dan pasti akan memulai eksekusi programnya pada **fungsi main** ini, meskipun main bukan fungsi yang pertama ditulis di program.
- Melihat bentuk seperti itu dapat kita ambil kesimpulan bahwa batang tubuh program utama berada didalam **fungsi main( )**.
- Berarti dalam setiap pembuatan program utama, maka dapat dipastikan seorang pemrogram menggunakan minimal sebuah fungsi.

FUNGSI MAIN

- Tanda { harus ada pada setiap awal dari sebuah fungsi dan tentu saja harus diakhiri dengan tanda }.
- Tanda ini digunakan untuk menunjukkan cakupan (scope) dari sebuah fungsi, dimana untuk menunjukkan fungsi ini dimulai dan berakhir.