PEMROGRAMAN BERORIENTASI OBJEK

Pointer dan Referensi

Telkom University

Budhi Irawan, S.Si, M.T

PENDAHULUAN

- ***Pointer*** merupakan salah satu fitur C++ yang relatif cukup berbahaya karena dapat mengakibatkan sistem operasi pada komputer menjadi ***"crash" (rusak)***.
- Penggunaan ***pointer*** dengan cara yang salah juga dapat menyebabkan ***"bug"*** yang sangat sulit untuk ditemukan pada program yang dibuat.

VARIABEL POINTER

- ***Pointer*** dapat didefinisikan sebagai suatu ***variabel yang menyimpan alamat memori***.
- Apabila dideklarasikan sebuah variabel dengan tipe data tertentu, maka untuk mendapatkan alamat dari variabel tersebut adalah dengan menggunakan operator **&**
- Alamat inilah yang kemudian akan disimpan kedalam variabel yang bertipe ***pointer***.

VARIABEL POINTER

- Untuk mendeklrasikan variabel sebagai ***pointer***, maka hanya ditambahkan ***tanda asterisk (*)*** didepan nama variabelnya. Berikut ini adalah bentuk umum dari pendeklarasian variabel yang bertipe ***pointer*** :

```
tipe_data *nama_pointer;
```

VARIABEL POINTER

- Pemberian spasi pada ***tanda asterik (*)*** tidak akan mengubah arti, dengan demikian penulisan seperti sebelumnya bisa diganti dengan :

```
tipe_data* nama_pointer; atau
tipe_data * nama_pointer;
```

VARIABEL POINTER

- Tipe data tersebut berguna untuk menyatakan bahwa ***pointer*** yang dideklarasikan akan ditempati oleh data data dengan tipe tertentu
- Sebagai contoh akan dideklarasikan ***pointer P*** yang ditempati oleh tipe data **`long`**, maka bentuk pendeklarasiannya adalah sebagai berikut :

```
// Mendeklarasikan pointer P
// yang akan menunjuk ke tipe long
long *P;
```

VARIABEL POINTER

- Jika kita memiliki sebuah variabel yang bertipe **`long`** (misalnya **`X`**), maka kita dapat memerintahkan ***pointer*** **`P`** diatas untuk menunjuk ke alamat yang ditempati oleh variabel **`X`**. Untuk melakukan tersebut kita perlu menuliskan kode seperti berikut :

```
// Mendeklarasikan variabel X dengan tipe long
long X;
// Mendeklarasikan pointer P
long *P;
// Memerintahkan P untuk menunjuk alamat
// dari variabel X
P = &X;
```

VARIABEL POINTER

- Apabila kita analisis potongan kode diatas, sebenarnya konsepnya sangat sederhana. Kita tahu bahwa P adalah ***poniter*** (*berisi alamat*) dan **&X** juga berisi alamat, maka kita dapat menyimpan alamat dari variabel **X** tersebut ke ***pointer*** **P**.
- Kita tidak diizinkan untuk memasukan sebuah nilai (*bukan alamat*) kedalam ***pointer*** **P**, seperti ditunjukan kode berikut :

```
// SALAH, karena X berupa nilai
// (bukan berupa alamat)
p = X;
```

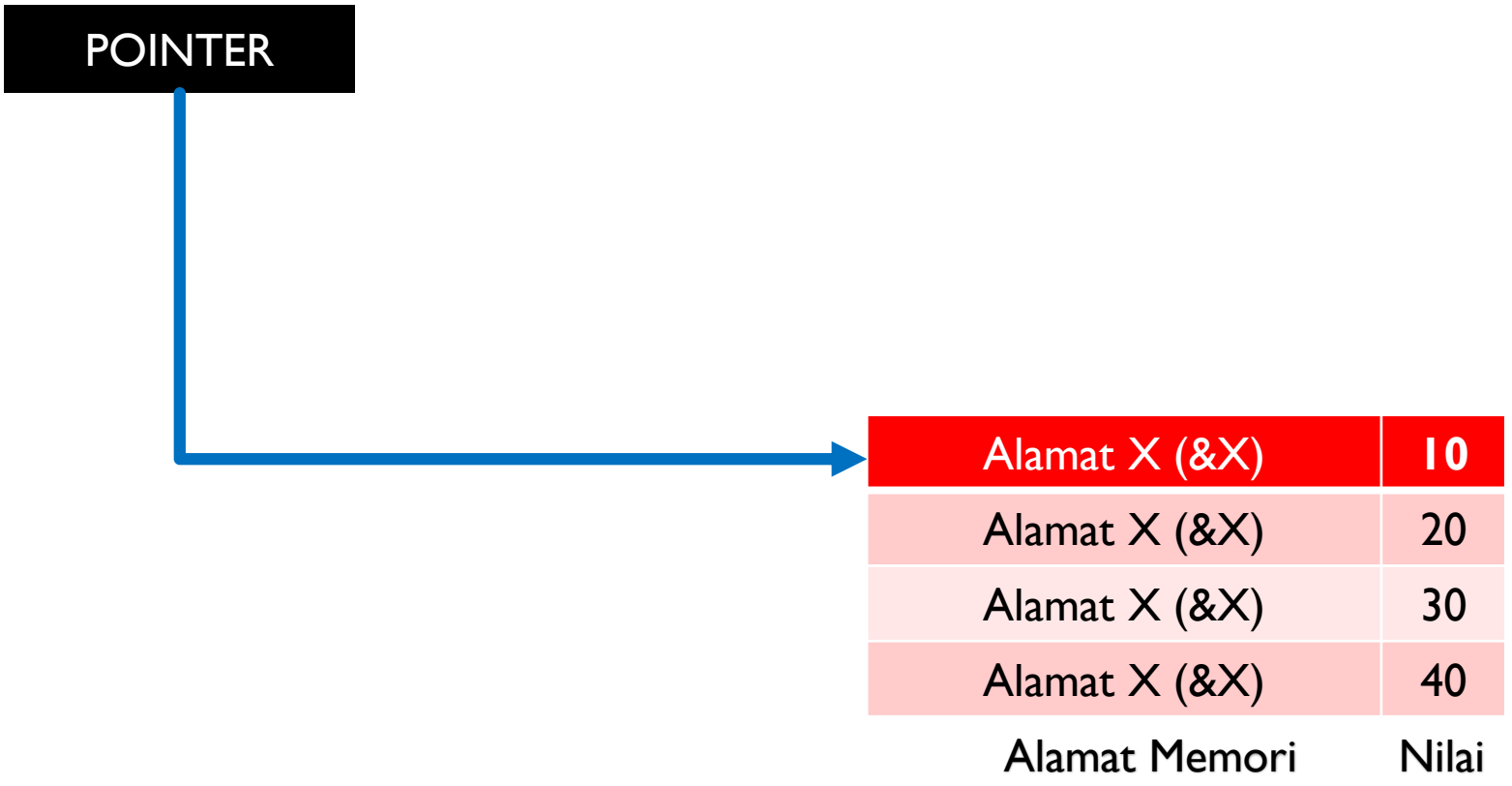
VARIABEL POINTER

- Apabila kita memang ingin mengisikan nilai kedalam alamat yang disimpan oleh ***pointer*** **P**, maka seharusnya kita menggunakan ***tanda asterik (*)*** didepan ***pointer*** tersebut, yaitu dengan mengubah kode diatas menjadi seperti dibawah ini :

```
// BENAR, karena P adalah nilai yang berada
// pada pointer P
*p = X;
```

- Sebagai catatan ***P** ini disebut dengan ***dereference pointer***

VARIABEL POINTER



VARIABEL POINTER

- Pada gambar di atas Alamat **1** dari memori ditempati oleh ***variabel* X** yang bertipe **long** dan Nilai dari ***variabel* X** adalah **10.** Sebelumnya tadi ada ***pointer* P** yang menunjuk ke alamat **X**, maka untuk mendapatkan nilai **X** kita dapat menggunakan ***dereference pointer***, yaitu ***P**. Dengan demikian, dapat disimpulkan bahwa jika :

```
P = &X; // Keduanya menyimpan alamat
```

maka

```
*P = X; // Keduanya menyimpan nilai
```

VARIABEL POINTER

```
#include <iostream>
using namespace std;
int main() {
  long *P;
  long X;
  P = &X;
  X = 10; // Mengisikan nilai 10 ke dalam variabel X
  cout<<"Nilai X    : "<<X<<endl;
  cout<<"Nilai *P   : "<<*P<<endl;
  cout<<"Nilai P    : "<<P<<endl;
  cout<<"Nilai &X   : "<<&X<<endl;
  *P = 200;   // Mengisikan nilai 200 ke dalam *P
  cout<<"Nilai X    : "<<X<<endl;
  cout<<"Nilai *P   : "<<*P<<endl;
  cout<<"Nilai P    : "<<P<<endl;
  cout<<"Nilai &X   : "<<&X<<endl;
  return 0;
}
```

VARIABEL POINTER

- Apabila diamati hasil program diatas, pada saat pengisian variabel **`X`** dengan nilai `10`, **`*P`** juga akan bernilai `10`.
- Begitu juga pada saat dimasukan nilai kedalam **`*P`** dengan nilai `200`, variabel **`X`** juga akan berubah nilainya menjadi `200`.
- Hal ini menunjukan bahwa **`*P`** akan selalu sama dengan **`X`**, dan ini semua disebabkan karena **`*P`** dan variabel **`X`** tersebut menempati alamat yang sama, yaitu **`0x22ff08`**

MEMASUKAN NILAI KEDALAM POINTER

- Nilai yang dimaksud disini adalah berupa alamat dan bukan berupa nilai data.
- Walaupun tampak mudah tapi kita juga harus berhati-hati dalam melakukan hal ini.
- Perlu diperhatikan bahwa tipe data dari ***pointer*** harus sama dengan tipe data dari variabel yang akan menempatinya.
- Hal ini merupakan hal yang biasa terabaikan oleh para programmer pemula.

MEMASUKAN NILAI KEDALAM POINTER

- Sebagai contoh kita mendeklarasikan pointer **P** ke tipe double dan kita juga memiliki variabel **X** yang bertipe integer.
- Pada kasus ini tidak diizinkan untuk menyimpan alamat dari variabel **X** ke pointer **P** karena tipenya berbeda. Berikut ini adalah potongan program yang menunjukan konsep diatas :

```
double *P;
int X; // Mendeklarasikan variabel yang bertipe int
P = &X;
/* SALAH, karena P hanya dapat menyimpan alamat dari
   variabel-variabel yang bertipe double saja */
```

POINTER TANPA TIPE

- Sebelumnya kita telah mengetahui bahwa ***pointer*** harus diisikan dengan alamat dari variabel yang bertipe sama dengan ***pointer*** tersebut.
- Sebenarnya ada cara khusus untuk membuat ***pointer*** yang kita deklarasikan tersebut dapat menunjuk ke semua tipe data, yaitu dengan mendeklarsikan ***pointer*** tersebut sebagai ***pointer*** tanpa tipe.
- ***Pointer*** semacam ini sering dinamakan dengan ***"void ponter"***.

POINTER TANPA TIPE

- Bentuk umum untuk mendeklarasikan pointer tanpa tipe ini adalah sebagai berikut :

```
void *nama_pointer;
```

POINTER TANPA TIPE

```
#include <iostream>
using namespace std;
int main() {
  void *P;   // Mendeklarasikan pointer P sebagai pointer tanpa tipe
             // Mendeklarasikan variabel X, Y dan Z dengan tipe berbeda
             int X; long Y; double Z;
  // Memerintahkan P untuk menunjuk ke alamat dari variabel X
  P = &X;
  X = 100;  // Mengisikan variabel X dengan nilai 100
            // Menampilkan hasil
            cout<<"Nilai X  : "<<X<<endl;
            cout<<"Nilai P  : "<<P<<endl;
            cout<<"Nilai &X : "<<&X<<endl;
            cout<<endl;
  // Memerintahkan P untuk menunjuk ke alamat dari variabel Y
  P = &Y;
  Y = 2000; // Mengisikan variabel Y dengan nilai 2000
  // Menampilkan hasil
  cout<<"Nilai Y  : "<<Y<<endl;
  cout<<"Nilai P  : "<<P<<endl;
  cout<<"Nilai &Y : "<<&Y<<endl;
  cout<<endl;
  // Memerintahkan P untuk menunjuk ke alamat dari variabel Z
  P = &Z;
  Z = 21.0378;    // Mengisikan variabel Z dengan nilai 21.0378
  // Menampilkan hasil
  cout<<"Nilai Z  : "<<Z<<endl;
  cout<<"Nilai P  : "<<P<<endl;
  cout<<"Nilai &Z : "<<&Z<<endl;
  return 0;
}
```

INIALISASI PADA POINTER

- Setiap kita mendeklarasian sebuah ***pointer***, maka ***pointer*** tersebut akan menunjuk lokasi acak di memori, oleh karena itu kita perlu mengatur pointer yang kita deklarasikan tersebut kedalam **`NULL`** , atau tidak menunjuk lokasi manapun.

INISIALISASI PADA POINTER

```
#include <iostream>

using namespace std;

int main() {
  int *P;  // Mendeklarasikan pointer P
           // yang akan menunjuk tipe data int

  cout<<"Alamat yang ditunjuk oleh pointer P:
"<<P;

  return 0;
}
```

INIALISASI PADA POINTER

- ***Pointer*** seperti diatas disebut sebagai pointer yang belum diinisialisasi.
- Untuk melakukan inisialisasi ***pointer***, kita dapat mengisikan nilai **`NULL`** kedalam pointer yang kita deklarasikan.

INISIALISASI PADA POINTER

```
#include <iostream>

using namespace std;

int main() {
  int *P;  // Mendeklarasikan pointer P
           // yang akan menunjuk tipe data int

  P = NULL; // Inisialisasi pointer P
            // dengan nilai NULL

  cout<<"Alamat yang ditunjuk oleh pointer P: "<<P;

  return 0;
}
```

Nilai 0 pada hasil diatas menunjukan bahwa pointer tersebut tidak menunjuk ke lokasi manapun.

KONSTANTA PADA POINTER

- Sama halnya seperti variabel biasa, ***pointer*** juga dapat bernilai ***konstan***
- Cara untuk mendeklarasikannya pun sama, yaitu dengan menggunakan ***keyword `const`***.
- Namun perlu untuk diperhatikan bahwa penempatan keyword **`const`** yang salah akan menyebabkan perbedaan arti.
- ***Keyword `const`*** dapat ditempatkan sebelum tipe data, sesudah tipe data ataupun pada keduanya (sebelum dan sesudah)

SEBELUM TIPE DATA

- Bentuk umumnya adalah sebagai berikut :

```
Const tipe_data * nama_pointer;
```

- Bentuk di atas berarti bahwa pointer akan menunjuk ke nilai dengan tipe data tertentu, dimana ***nilai tersebut bersifat tetap***.

SEBELUM TIPE DATA

```
#include <iostream>
using namespace std;
int main() {
  const int *P1;    // P1 akan menunjuk ke tipe data int
                    // yang bersifat tetap

  int X, Y;
  X = 5;
  Y = 10;
  P1 = &X;

//  *P1 = 2;  // SALAH, karena nilai yang ditunjuk oleh P1
              // (nilai *P1) bersifat tetap

  P1 = &Y;   // BENAR,
             // karena pointer P1 tidak bersifat tetap

// Menampilkan hasil
  cout<<"Nilai X  : "<<X<<endl;
  cout<<"Nilai Y  : "<<Y<<endl;
  cout<<"Nilai &X : "<<&X<<endl;
  cout<<"Nilai &Y : "<<&Y<<endl;
  cout<<"Nilai P1 : "<<P1<<endl;

  return 0;
}
```

SEBELUM TIPE DATA

- Pada program diatas kita melihat bahwa mula-mula **P1** menunjuk ke alamat dari variabel X
- Pendeklarasian **`const int *P1`** menyatakan bahwa **`pointer P1`** menunjuk ke nilai yang tetap, bukan alamat yang tetap sehingga kita masih diizinkan untuk mengubah **`pointer P1`** supaya menunjuk ke alamat yang baru, yaitu alamat dari **`variabel Y`**.

SESUDAH TIPE DATA

- Bentuk umumnya adalah sebagai berikut :

```
tipe_data * const nama_pointer;
```

- Berbeda dengan sebelumnya, di sini ***pointer bersifat tetap sehingga alamatnya tidak dapat diubah***.
- Meskipun demikian, nilai dari variabel yang ditunjuk dapat diubah.

SESUDAH TIPE DATA

```
#include <iostream>

using namespace std;

int main() {
  int X=5, Y=10;

  // P2 akan menunjuk ke alamat yang tetap,
  // yaitu alamat X
  int * const P2 = &X;

  *P2 = 2;    // BENAR,
              // karena nilai *P2 dapat diubah

//  P2 = &Y; // SALAH,
              // karena pointer P2 bersifat tetap
// Menampilkan hasil
  cout<<"Nilai X   : "<<X<<endl;
  cout<<"Nilai Y   : "<<Y<<endl;
  cout<<"Nilai P2  : "<<P2<<endl;
  cout<<"Nilai &X  : "<<&X<<endl;
  cout<<"Nilai &Y  : "<<&Y<<endl;
  return 0;
}
```

SEBELUM & SESUDAH TIPE DATA

- Bentuk umumnya adalah sebagai berikut :

```
const tipe_data * const nama_pointer;
```

- Bentuk yang ketiga ini merupakan bentuk gabungan dari bentuk yang pertama dan kedua, artinya disini ***dideklarasikan pointer konstan yang menunjuk ke suatu nilai yang konstan pula.***
- Dengan kata lain ***pointer*** ini menunjuk ***alamat yang tetap*** dimana alamat tersebut juga berisi ***nilai yang tetap***

SEBELUM & SESUDAH TIPE DATA

```
#include <iostream>
using namespace std;
int main() {
  int X=5, Y=10;
  /* P3 akan menunjuk ke alamat yang tetap,
     yaitu alamat X
     dan nilai yang ditunjuk juga tetap */
  const int * const P3 = &X;

//  *P3 = 2;  // SALAH,
              // karena nilai *P3 tidak dapat diubah

//  P3 = &Y;  // SALAH,
              // karena pointer P3 bersifat tetap

// Menampilkan hasil
 cout<<"Nilai X  : "<<X<<endl;
  cout<<"Nilai Y  : "<<Y<<endl;
  cout<<"Nilai P3 : "<<P3<<endl;
  cout<<"Nilai &X : "<<&X<<endl;
  cout<<"Nilai &Y : "<<&Y<<endl;
  return 0;
}
```

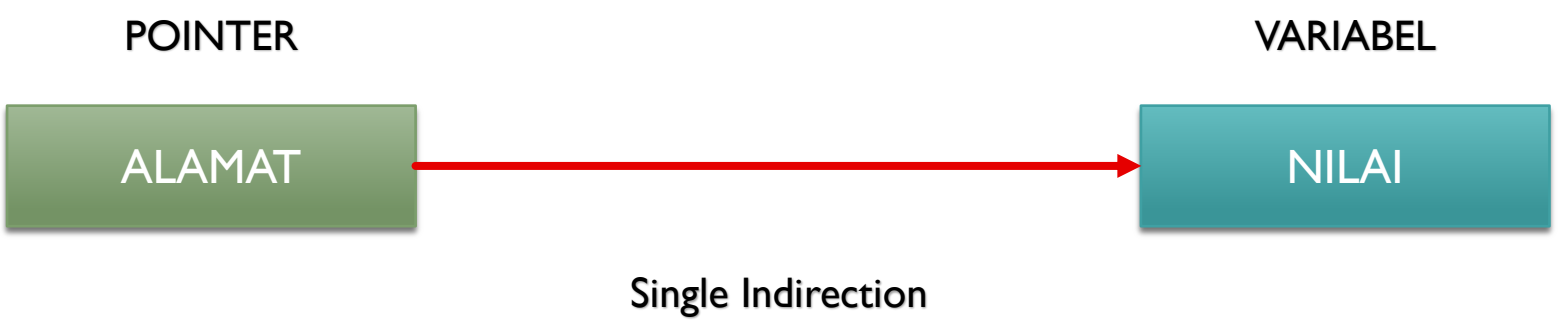
SEBELUM & SESUDAH TIPE DATA

- Pada program di atas, mula-mula nilai **`X`** sama dengan **`5`**, sedangkan alamat dari **`X`** ditunjuk **`pointer P3`**.
- Oleh karena **`P3`** adalah pointer yang bersifat tetap dan juga menunjuk ke nilai yang tetap, maka statemen **`*P3 = 2`** dan **`P3 = &Y`** diatas tidak diizinkan oleh ***compiler***

POINTER KE POINTER

- Dalam C/C++, kita dapat mempunyai pointer yang menunjuk ke pointer lain yang telah menunjuk ke alamat tertentu.
- Situasi ini sering disebut dengan ***multiple indirection*** atau ***pointer ke pointer***.
- ***Pointer ke pointer*** ini memang sering membingungkan.
- Sebenarnya konsepnya sederhana, kita tahu bahwa pointer normalnya berisi alamat dari **sebuah objek** (***variabel***) yang menyimpan nilai berupa data.

POINTER KE POINTER



Single Indirection

Multiple Indirection

POINTER KE POINTER

- Pada ***Single Indirection***, ***pointer*** langsung menunjuk ke alamat dari suatu variabel, sedangkan pada ***multiple indirection,*** terdapat ***pointer*** lain yang menunjuk ke ***pointer*** yang sedang menunjuk ke alamat dari suatu variabel.
- Untuk mendeklarasikan sebuah ***pointer*** yang akan menunjuk ke ***pointer*** lain diperlukan penggunaan tanda ***asterik*** sebanyak ***dua kali (**)***

POINTER KE POINTER

```
int **P;
```

- Maka **P** bukan merupakan ***pointer*** yang menunjuk ke tipe data **int**, melainkan ***pointer*** yang menunjuk ***pointer***, yaitu ***pointer*** ke tipe data int.

POINTER KE POINTER

```
#include <iostream>
using namespace std;
int main() {
  int X = 25; // Mendeklarasikan variabel X  dengan nilai default 25
  int *P1;    // Mendeklarasikan pointer P1 yang akan menunjuk ke tipe data int
  int **P2;   // Mendeklarasikan pointer P2 yang akan menunjuk ke pointer P1

  P1 = &X;
  P2 = &P1;

  // Menampilkan nilai
  cout<<"Nilai X       : "<<X<<endl;
  cout<<"Nilai *P1     : "<<*P1<<endl;
  cout<<"Nilai *P2     : "<<*P2<<endl;
  cout<<"Nilai **P2    : "<<**P2<<endl;
  cout<<endl;

  // Menampilkan alamat
  cout<<"Nilai &X      : "<<&X<<endl;
  cout<<"Nilai P1      : "<<P1<<endl;
  cout<<"Nilai P2      : "<<P2<<endl;

  return 0;
}
```

POINTER KE POINTER

- Nilai **`X`** dapat diakses dengan menggunakan **`*P1`** atau **`**P2`**, sedangkan alamat dari variabel **`X`** dapat diakses dengan menggunakan **`&X`**, **`P`** ataupun **`*P2`** (catatan : **`*P2`** merupakan nilai yang menempati alamat yang ditunjuk oleh ***pointer*** **`P2`**).
- ***Pointer*** **`P2`** tidak menunjuk ke tipe data int, melainkan menunjuk ke ***pointer*** **`P1`**.
- **`P2`** sendiri menyimpan alamat dari pointer **`P1`**

REFERENSI

- ***Referensi*** ***(reference)*** merupakan alias atau nama lain (julukan) dari sebuah variabel.
- Untuk bisa membuat ***referensi*** maka digunakan tanda **&** dibelakang tipe data yang akan diacu.
- Bentuk umum pembuatan ***referensi*** adalah :

```
tipe_data& nama_alias = nama_variabel;
```

bisa juga ditulis di depan nama alias, seperti berikut :

```
tipe_data &nama_alias = nama_variabel;
```

REFERENSI

```
#include <iostream>
using namespace std;
int main() {
  int X;  // Mendeklarasikan variabel X

  // Membuat alias dari variabel X  dengan nama AliasX
  int& AliasX = X;
  // Mengisikan nilai ke dalam variabel X
  X = 12;

  // Menampilkan nilai yang disimpan dalam variabel X dan AliasX
  cout<<"Nilai X       : "<<X<<endl;
  cout<<"Nilai AliasX  : "<<AliasX<<endl;
  cout<<endl;

  // Mengisikan nilai ke dalam AliasX
  AliasX = 35;

// Menampilkan kembali nilai yang disimpan dalam variabel X dan AliasX
  cout<<"Nilai X       : "<<X<<endl;
  cout<<"Nilai AliasX  : "<<AliasX<<endl;
  return 0;
}
```

REFERENSI

- Pada program di atas tampak bahwa pada saat kita memasukan nilai **`12`** ke dalam variabel **`X`**, **`AliasX`** juga akan bernilai **`12`**.
- Begitu juga sebaliknya, pada saat kita memasukan nilai **`35`** ke dalam **`AliasX`**, variabel **`X`** juga akan bernilai **`35`**.
- Ini membuktikan bahwa keduanya sebenarnya merupakan satu buah variabel yang memiliki dua nama.

ALAMAT REFERENSI

- Perlu untuk diperhatikan bahwa ***alias (referensi)*** dan variabel yang diacu akan menempati alamat yang sama di memori.
- Alasannya, seperti yang disebutkan sebelumnya, keduanya bukan merupakan dua buah variabel yang berbeda, melainkan ***satu variabel yang mempunyai dua nama***.

ALAMAT REFERENSI

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

int main() {

  int X = 50; // Mendeklarasikan variabel X dengan nilai default 50

  // Membuat alias dari variabel X dengan nama AliasX
  int& AliasX = X;

  // Menampilkan nilai yang disimpan dalam variabel X dan AliasX
  cout<<"Nilai X        : "<<X<<endl;
  cout<<"Nilai AliasX   : "<<AliasX<<endl;
  cout<<endl;

  // Menampilkan alamat dari variabel X dan AliasX
  cout<<"Alamat X   : "<<&X<<endl;
  cout<<"Alamat AliasX  : "<<&AliasX<<endl;

  return 0;
}
```

REFERENSI KONSTAN

- Dalam C++, dapat didefinisikan sebuah ***referensi*** yang ***bersifat tetap (constant reference)***, artinya ***nilai dari referensi ini tidak dapat diubah.***
- Yang menjadi ***konstanta*** disini adalah hanya referensinya saja, sedangkan variabelnya tidak bersifat konstan.

REFERENSI KONSTAN

```
#include <iostream>
using namespace std;
int main() {
  // Mendeklarasikan varaibel X dengan nilai default 5
  int X = 5;

  // Mendeklarasikan referensi konstan
  const int& AliasX = X;

  // Menampilkan nilai dari AliasX dan variabel X
  cout<<"Nilai X      : "<<X<<endl;
  cout<<"Nilai AliasX : "<<AliasX<<endl<<endl;

  // Mengubah nilai X
  X = 12;  // BENAR, hal ini diperbolehkan karena X TTidak bersifat konstan

  // Menampilkan kembali nilai dari AliasX dan variabel X
  cout<<"Nilai X      : "<<X<<endl;
  cout<<"Nilai AliasX : "<<AliasX<<endl<<endl;

  // Mengubah nilai AliasX, AliasX = 25; ,SALAH, hal ini tidak diizinkan oleh compiler

  return 0;
}
```