Pointer

Variabel merupakan suatu nilai yang disimpan dalam memory yang dapat diakses dengan identifier. Variabel ini sesunggunhnya disimpan pada suatu alamat didalam memory. Dimana setiap alamat memory akan berbeda dengan yang lainnya (unik).

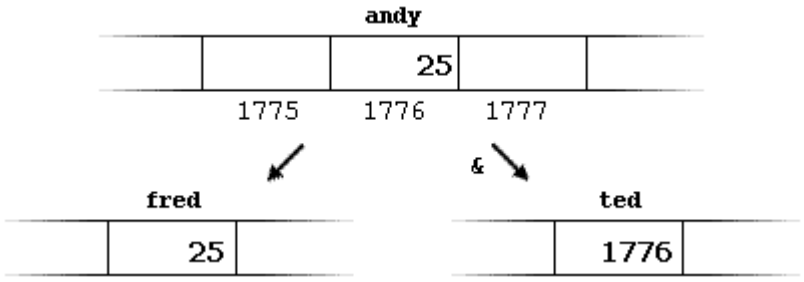
Operator Alamat (Address operator (&))
Pada saat pendeklarasian variable, user tidak diharuskan menentukan lokasi sesungguhnya pada memory, hal ini akan dilakukan secara otomatis oleh kompilerdan operating sysem pada saat run-time. Jika ingin mengetahui dimana suatu variable akan disimpan, dapat dilakukan dengan memberikan tanda *ampersand* (&) didepan variable , yang berarti ***"address of"***. Contoh:

```
ted = &andy;
```

Akan memberikan variable **ted** alamat dari variable **andy**, karena variable **andy** diberi awalan karakter *ampersand* (&), maka yang menjadi pokok disini adalah alamat dalam memory, bukan isi variable. Misalkan **andy** diletakkan pada alamat **1776** kemudian dituliskan instruksi sbb :

```
andy = 25;
fred = andy;
ted = &andy;
```

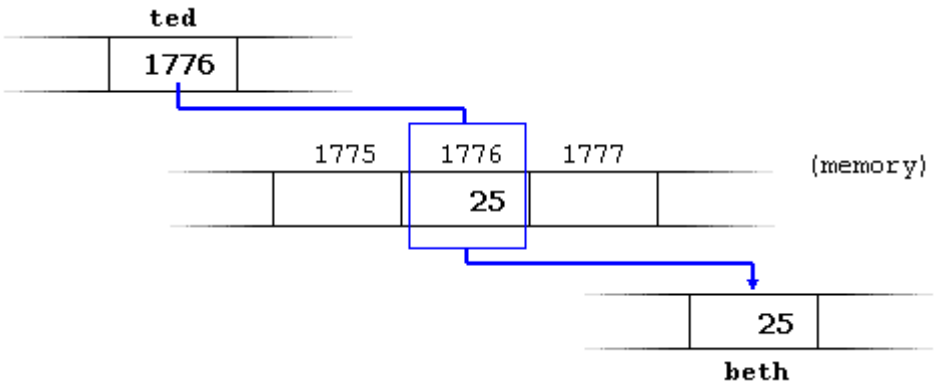
Maka hasilnya :



Operator Reference (*)
Dengan menggunakan pointer, kita dapat mengakses nilai yang tersimpan secara langsung dengan memberikan awalan operator *asterisk* (*) pada identifier pointer, yang berarti ***"value pointed by"***. Contoh :

```
beth = *ted;
```

(dapat dikatakan:"beth sama dengan nilai yang ditunjuk oleh ted") **beth** = **25**, karena **ted** dialamat **1776**, dan nilai yang berada pada alamat **1776** adalah **25**.



Ekspresi dibawah ini semuanya benar, perhatikan :

andy == 25
&andy == 1776
ted == 1776
*ted == 25

Ekspresi pertama merupakan *assignation* bahwa **andy=25;**. Kedua, menggunakan operator alamat (address/derefence operator (&)), sehingga akan mengembalikan alamat dari variabel **andy**. Ketiga bernilai benar karena *assignation* untuk **ted** adalah **ted = &andy;**. Keempat menggunakan reference operator (*) yang berarti nilai yang ada pada alamat yang ditunjuk oleh **ted**, yaitu **25**. Maka ekspresi dibawah ini pun akan bernilai benar :

*ted == andy

Deklarasi variable bertipe pointer
Format deklarasi pointer :

***type * pointer_name*;**

Dimana ***type*** merupakan tipe dari data yang ditunjuk, bukan tipe dari pointer-nya. Contoh :

int * number;
char * character;
float * greatnumber;

Contoh :

```
// my first pointer
#include <iostream.h>

int main ()
{ int value1 = 5, value2 = 15;
  int * mypointer;

  mypointer = &value1;
```

```
  *mypointer = 10;
  mypointer = &value2;
  *mypointer = 20;
  cout << "value1==" << value1 << "/ value2==" << value2;
  return 0;
}
```

Output : value1==10 / value2==20

Perhatikan bagaimana nilai dari **value1** dan **value2** diubah secara tidak langsung. Pertama **mypointer** diberikan alamat **value1** dengan menggunakan tanda ampersand (&). Kemudian memberikan nilai **10** ke nilai yang ditunjuk oleh **mypointer**, yaitu alamat dari **value1**, maka secara tidak langsung **value1** telah dimodifikasi. Begitu pula untuk **value2**.

Contoh :

```
// more pointers
#include <iostream.h>

int main ()
{
  int value1 = 5, value2 = 15;
  int *p1, *p2;

  p1 = &value1;     // p1 = address of value1
  p2 = &value2;     // p2 = address of value2
  *p1 = 10;         // value pointed by p1 = 10
  *p2 = *p1;        // value pointed by p2 = value pointed by p1
  p1 = p2;          // p1 = p2 (value of pointer copied)
  *p1 = 20;         // value pointed by p1 = 20

  cout << "value1==" << value1 << "/ value2==" << value2;
  return 0;

}
```

Output : value1==10 / value2==20

Array dan Pointer

Identifier suatu array equivalen dengan alamat dari elemen pertama, pointer equivalen dengan alamat elemen pertama yang ditunjuk. Perhatikan deklarasi berikut :

```
int numbers [20];
int * p;
```

maka deklarasi dibawah ini juga benar :

```
p = numbers;
```

p dan **numbers** equivalen, dan memiliki sifat (*properties*) yang sama. Perbedaannya, user dapat menentukan nilai lain untuk pointer **p** dimana **numbers** akan selalu menunjuk nilai yang sama seperti yang telah didefinisikan. **p**, merupakan *variable pointer*, **numbers** adalah *constant pointer*. Karena itu walaupun instruksi diatas benar, tetapi tidak untuk instruksi dibawah ini :

```
numbers = p;
```

karena **numbers** adalah array (constant pointer), dan tidak ada nilai yang dapat diberikan untuk identifier konstant (*constant identifiers*).

Contoh : **Output :**

```
// more pointers
#include <iostream.h>

int main ()
{
  int numbers[5];
  int * p;
  p = numbers;  *p = 10;
  p++;  *p = 20;
  p = &numbers[2];  *p = 30;
  p = numbers + 3;  *p = 40;
  p = numbers;  *(p+4) = 50;
  for (int n=0; n<5; n++)
    cout << numbers[n] << ", ";
  return 0;
}
```

10, 20, 30, 40, 50,

Inisialisasi Pointer
Contoh:

```
int number;
int *tommy = &number;
```

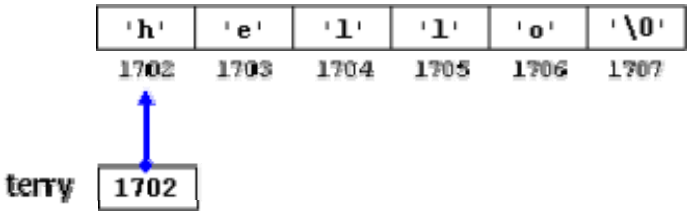
Equivalen dengan :

```
int number;
int *tommy;
tommy = &number;
```

Seperti pada array, inisialisasi isi dari pointer dapat dilakukan dengan deklarasi seperti contoh berikut :

```
char * terry = "hello";
```

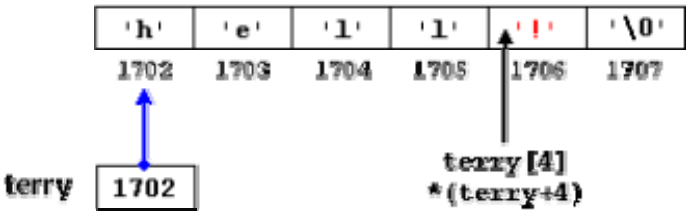
Misalkan "hello" disimpan pada alamat 1702 dan seterusnya, maka deklarasi tadi dapat digambarkan sbb :



terry berisi nilai **1702** dan bukan **'h'** atau **"hello"**, walaupun **1702** menunjuk pada karakter tersebut. Sehingga jika akan dilakukan perubahan pada karakter **'o'** diganti dengan tanda **'!'** maka ekspresi yang digunakan ada 2 macam :

```
terry[4] = '!';
*(terry+4) = '!';
```

Penulisan **terry[4]** dan ***(terry+4)**, mempunyai arti yang sama. Jika digambarkan :



Pointer Arithmatika

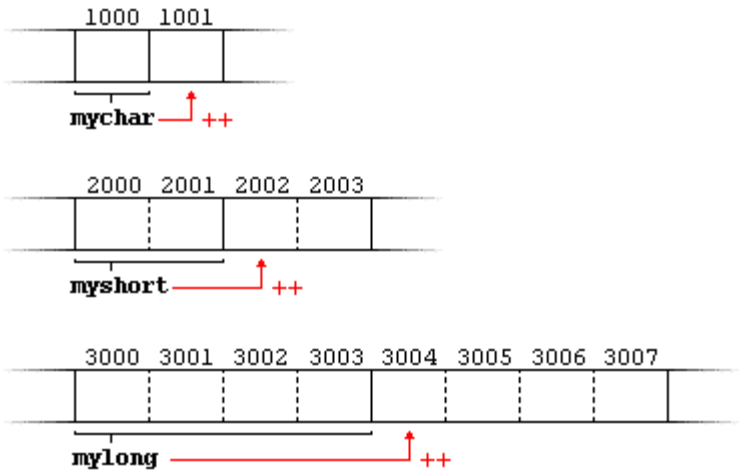
Contoh, *char* memerlukan 1 byte, *short* memerlukan 2 bytes dan *long* memerlukan 4. Terdapat 3 buah pointer :

```
char *mychar;
short *myshort;
long *mylong;
```

ekspresi diatas akan menunjuk pada lokasi dimemory masing-masing **1000**, **2000** and **3000**, sehingga jika dituliskan :

```
mychar++;
myshort++;
mylong++;
```

mychar, akan bernilai 1001, myshort bernilai 2002, dan mylong bernilai 3004. Alasannya adalah ketika terjadi pertambahan maka akan ditambahkan dengan tipe yang sama seperti yang didefinisikan berupa ukuran dalam bytes.



Perhatikan ekspresi dibawah ini :

```
*p++;
*p++ = *q++;
```

Ekspresi pertama equivalen dengan *(p++) dan yang dilakukan adalah menambahkan **p** (yaitu alamat yang ditunjuk, bukan nilai yang dikandungnya).
Ekspresi kedua, yang dilakukan pertama adalah memberikan nilai *q ke *p dan kemudian keduanya ditambahkan 1 atau dengan kata lain :

```
*p = *q;
p++;
q++;
```

void pointer
Tipe pointer *void* merupakan tipe khusus. *void* pointers dapat menunjuk pada tipe data apapun, nilai integer value atau float, maupun string atau karakter. Keterbatasannya adalah tidak dapat menggunakan operator asterisk (*), karena panjang pointer tidak diketahui, sehingga diperlukan operator *type casting* atau *assignations* untuk mengembalikan nilai *void* pointer ketipe data sebenarnya.
Contoh :

```
// integer increaser
#include <iostream.h>

void increase (void* data, int type)
{
  switch (type)
  {
    case sizeof(char) : (*((char*)data))++; break;
    case sizeof(short): (*((short*)data))++; break;
    case sizeof(long) : (*((long*)data))++; break;
  }
}
```

```
int main ()
{
  char a = 5;
  short b = 9;
  long c = 12;
  increase (&a,sizeof(a));
  increase (&b,sizeof(b));
  increase (&c,sizeof(c));
  cout << (int) a << ", " << b << ", " << c;
  return 0;
}
```

Output : 6, 10, 13

Pointer untuk functions

C++ memperbolehkan operasi dengan pointer pada function. Kegunaan yang utama adalah untuk memberikan satu function sebagai parameter untuk function lainnya. Deklarasi pointer untuk function sama seperti prototype function kecuali nama function dituliskan diantara tanda kurung () dan operator asterisk (*) diberikan sebelum nama.

Contoh :

```
// pointer to functions
#include <iostream.h>

int addition (int a, int b)
{ return (a+b); }

int subtraction (int a, int b)
{ return (a-b); }

int (*minus)(int,int) = subtraction;

int operation (int x, int y, int (*functocall)(int,int))
{
  int g;
  g = (*functocall)(x,y);
  return (g);
}

int main ()
{
  int m,n;
  m = operation (7, 5, addition);
  n = operation (20, m, minus);
  cout <<n;
  return 0;
}
```

Output :
8

Dari contoh diatas, **minus** merupakan pointer global untuk function yang mempunyai 2 parameters bertipe **int**, kemudian diberikan untuk menunjuk function **subtraction**, ditulis dalam satu baris instruksi :

```
int (* minus)(int,int) = subtraction;
```