PERTEMUAN 3
POINTER

A. TUJUAN PEMBELAJARAN

Setelah menyelesaikan pertemuan ini, mahasiswa mampu mempraktekkan:

1. Pengenalan Pointer
2. Reference operator (&)
3. Dereference operator (*)
4. Mendeklarasikan variabel tipe pointer
5. Pointer dan Array
6. Pointer to Pointer

B. URAIAN MATERI

1. Pengenalan Pointer

   Dilihat dari fungsinya, pointer dapat diartikan sebagai suatu nilai yang menunjuk atau menyatakan alamat suatu lokasi memori. Dilihat dari fisiknya, Pointer adalah sebuah sebuah variable yang berisikan alamat suatu lokasi memory.

   Pointer, juga disebut link atau reference, didefinisikan sebagai sebuah objek, seringkali sebuah variabel, yang menyimpan lokasi (yaitu alamat mesin) dari beberapa objek lain, biasanya sebuah struktur yang berisi data yang ingin kita manipulasi. Jika kita menggunakan pointer untuk menemukan semua data yang kita minati, maka kita tidak perlu khawatir tentang di mana sebenarnya data itu disimpan, karena dengan menggunakan pointer, kita dapat membiarkan sistem komputer itu sendiri menemukan data saat diperlukan.

2. Reference operator (&)

   Alamat yang menempatkan variabel di dalam memori adalah apa yang kita sebut reference ke variabel itu. Reference ke variabel ini dapat diperoleh dengan didahului variabel pengenal(&), yang dikenal sebagai operator

reference, dan yang secara harfiah dapat diterjemahkan sebagai "alamat". Sebagai contoh:

ted = &andy;

Ini akan menugaskan ke alamat variabel andy, karena ketika mendahului nama variabel andy dengan operator reference (&) kita tidak lagi berbicara tentang isi variabel itu sendiri, tetapi tentang referencenya (yaitu, alamatnya di Penyimpanan).

sekarang kita akan berasumsi bahwa andy ditempatkan selama runtime di alamat memori 1776. Angka ini (1776) hanyalah asumsi untuk membantu memperjelas beberapa konsep dalam tutorial ini, tetapi kenyataannya, kita tidak dapat mengetahui sebelum runtime nilai sebenarnya yang akan dimiliki alamat variabel dalam memori.

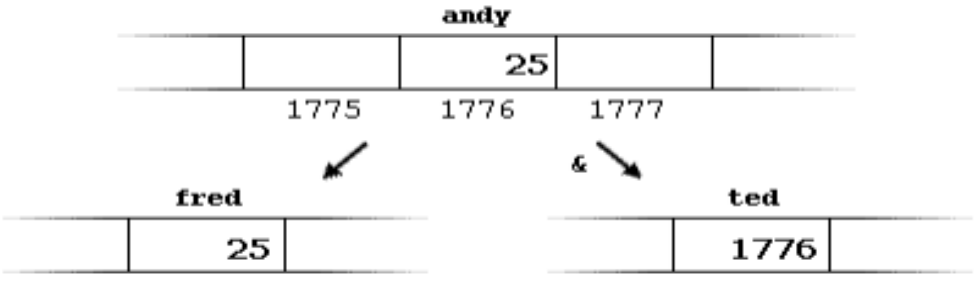
Pertimbangkan fragmen kode berikut:

andy = 25;

fred = andy;

ted = &andy;

Nilai yang dikandung setiap variabel setelah eksekusinya ditunjukkan pada diagram berikut:



Gambar 3.3Reference operator

Pernyataan kedua menyalin konten variabel andy (25) ke Fred. Ini adalah operasi penugasan standar, seperti yang telah kami lakukan berkali-kali.

Akhirnya, pernyataan ketiga tidak menyalin nilai yang terkandung dalam andy, melainkan referensi ke sana (yaitu alamatnya, yang kami asumsikan adalah 1776). Alasannya adalah bahwa dalam operasi penugasan ketiga ini kita mempersiapkan pengenal andy dengan operator referensi (&) dan, oleh karena

itu, kita tidak lagi mengacu pada nilai andy, tetapi mengacu pada referensinya (alamatnya di memori).

Variabel yang menyimpan referensi ke variabel lain (seperti yang disebutkan di contoh sebelumnya) disebut pointer. Pointer adalah fungsi yang sangat kuat dari bahasa C ++ yang memiliki banyak kegunaan dalam pemrograman tingkat lanjut. Nanti kita akan melihat bagaimana jenis variabel ini digunakan dan dideklarasikan.

3. Dereference operator (*)

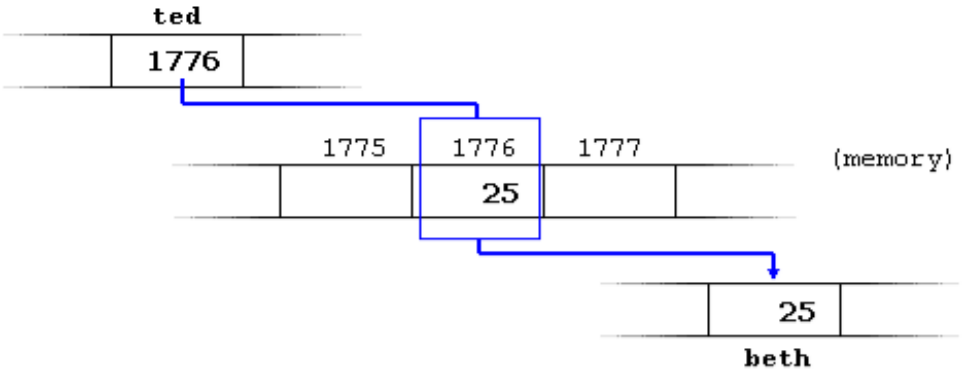
Kita baru saja melihat bahwa variabel yang menyimpan referensi ke variabel lain disebut pointer. Pointer harus "menunjuk" ke variabel yang referensinya disimpan.

Dengan pointer, kita bisa langsung mengakses nilai yang disimpan dalam variabel yang ditunjuknya. Untuk melakukan ini, cukup beri tanda bintang (*) sebelum pegangan pointer, yang bertindak sebagai operator dereferensi dan dapat diterjemahkan secara harfiah ke "valuepointedby".

Jadi, ikuti nilai dari contoh di atas saat kita menulis:

beth = *ted;

(yang dapat kita baca sebagai: "Beth sama dengan nilai yang ditunjukkan Ted") Beth akan mengambil nilai 25, karena Ted adalah 1776 dan nilai 1776 yang ditunjukkan adalah 25.



Gambar 3.4Dereference operator

Anda harus membedakan dengan jelas bahwa ekspresi ted mengacu

pada nilai 1776, sedangkan * ted (dengan tanda bintang * di depan pengidentifikasi) mengacu pada nilai yang disimpan di alamat 1776, yang dalam hal ini adalah 25. Perhatikan perbedaan antara include atau tidak menyertakan operator dereference (saya akan menambahkan komentar penjelasan tentang bagaimana masing-masing dari dua ekspresi ini dapat dibaca):

beth = ted; // bethequaltoted (1776)

beth = *ted; // bethequaltovaluepointedbyted (25)

Perhatikan perbedaan antara operator referensi dan dereferensi:

• & adalah operator referensi dan dapat dibaca sebagai "address of"

• * adalah operator dereferensi dan dapat dibaca sebagai "value pointed by"

Oleh karena itu, mereka memiliki arti yang saling melengkapi (atau berlawanan). Variabel yang direferensikan oleh & dapat didereferensi dengan *.

Sebelumnya, kami melakukan dua operasi penugasan berikut:

andy = 25;

ted = &andy;

Tepat setelah dua pernyataan ini, semua ekspresi berikut akan memberikan true sebagai hasil:

andy == 25

&andy == 1776

ted == 1776

*ted == 25

Ekspresi pertama cukup mudah mengingat bahwa operasi penugasan yang dilakukan pada andy adalah andy = 25. Yang kedua menggunakan operator referensi (&), yang mengembalikan alamat variabel andy, yang kami asumsikan adalah 1776. Yang ketiga adalah no-brainer karena ekspresi kedua benar dan operasi penugasan dilakukan pada tedted = &andy. Ekspresi keempat menggunakan operator dereferencing (*), yang, seperti yang baru kita lihat, dapat dibaca sebagai "valuepointedby ", dan nilai yang ditunjukkan oleh ted sebenarnya adalah 25.

Setelah semua ini, Anda dapat menyimpulkan bahwa ekspresi berikut juga benar selama alamat yang ditunjukkan oleh ted tetap tidak berubah:

*ted == andy

4. Mendeklarasikan variabel tipe pointer

Karena kemampuan pointer untuk merujuk langsung ke nilai yang ditunjuknya, deklarasinya harus menunjukkan tipe data apa yang dirujuk pointer. Menunjuk ke karakter tidak sama dengan menargetkan int atau float.

Deklarasi pointer mengikuti format ini:

type * name;

Di mana type adalah tipe data dari nilai yang harus ditunjuk pointer. Tipe ini bukanlah tipe pointer itu sendiri! tetapi tipe data yang ditunjuk pointer. Sebagai contoh:

int * number;

char * character;

float * greatnumber;

ni adalah deklarasi tiga pointer. Masing-masing dimaksudkan untuk merujuk pada jenis data yang berbeda, tetapi pada kenyataannya mereka semua adalah pointer dan semuanya menggunakan jumlah memori yang sama (ukuran pointer dalam memori tergantung pada platform tempat kode akan dijalankan). Namun, data yang mereka referensikan tidak menempati ruang atau tipe yang sama: poin pertama ke int, yang kedua ke char, dan yang terakhir ke float. Meskipun ketiga variabel sampel ini semuanya adalah pointer yang menempati ukuran yang sama dalam memori, mereka disebut tipe yang berbeda bergantung pada tipe yang mereka tunjuk: int *, char *, dan float *.

saya ingin menekankan bahwa asterisk (*) yang kita gunakan ketika kita mendeklarasikan pointer hanya berarti itu adalah pointer (itu adalah bagian dari penentu tipe kompositnya) dan tidak boleh bingung dengan operator dereferencing yang kami sebutkan sebelumnya. tetapi juga ditulis dengan asterisk (*). Mereka hanyalah dua hal berbeda yang diwakili oleh tanda yang sama.

Sekarang lihat kode ini:

```
// myfirstpointer
#include<iostream>
Using namespace std;
int main ()
{
int firstvalue, secondvalue;
 int * mypointer;

mypointer = &firstvalue;
*mypointer = 10;
mypointer = &secondvalue;
*mypointer = 20;
cout<< "firstvalueis " <<firstvalue<<endl;
cout<< "secondvalueis " <<secondvalue<<endl;
return 0;
}
```

Perhatikan bahwa meskipun kami tidak pernah secara langsung menentukan nilai untuk first value atau secondvalue, keduanya berakhir dengan nilai yang ditentukan secara tidak langsung oleh mypointer.Berikut prosedurnya:

Pertama, kita menetapkan referensi ke nilai pertama sebagai nilai mypointer menggunakan operator referensi (&). Dan kemudian kita menetapkan nilai 10 ke lokasi yang dituju mypointer, karena pada titik ini sebenarnya menunjuk ke lokasi firstvalue, sehingga mengubah nilai dari firstvalue. Untuk mendemonstrasikan bahwa sebuah pointer dapat mengambil beberapa nilai yang berbeda selama program yang sama, saya mengulangi proses tersebut dengan secondvalue danpointer yang sama, mypointer.

Berikut adalah contoh yang sedikit lebih rumit:

```
// more pointers
#include<iostream>
usingnamespacestd;
int main ()
{
int firstvalue = 5, secondvalue = 15;
int * p1, * p2;
p1 = &firstvalue;
p2 = &secondvalue;
 *p1 = 10;
*p2 = *p1;
 p1
      p1 = p2;
     *p1 = 20;
  cout << "firstvalue is " << firstvalue << endl;
  cout << "secondvalue is " << secondvalue << endl;
  return 0;}
```

Saya menambahkan komentar di setiap baris tentang bagaimana kode dapat dibaca: ampersand (&) sebagai "addressof" dan tanda bintang (*) sebagai ""valuepointedby".

Perhatikan bahwa ada ekspresi dengan pointer p1 dan p2 dengan dan tanpa operator dereferensi (*). Arti ekspresi yang menggunakan operator dereferencing (*) sangat berbeda dari yang tidak: jika operator ini mendahului nama pointer, ekspresi tersebut merujuk ke nilai yang direferensikan, sedangkan ketika nama pointer muncul tanpa operator ini, itu adalah mengacu pada nilai pointer itu sendiri (yaitu, arah pointer menunjuk).Hal lain yang mungkin menarik perhatian Anda adalah baris:

```
int * p1, * p2;
```

Ini mendeklarasikan dua pointer yang digunakan dalam contoh sebelumnya. Namun, perhatikan bahwa ada tanda bintang (*) untuk setiap pointer, jadi keduanya bertipe int * (pointer ke int). Jika tidak, tipe variabel kedua yang dideklarasikan pada baris ini akan menjadi int (dan bukan int *) karena hubungan prioritas. Jika kami telah menulis:

```
int * p1, p2;
```

p1 sebenarnya akan menjadi tipe int *, tetapi p2 akan menjadi tipe int (spasi tidak penting untuk tujuan ini). Ini karena aturan prioritas operator. Bagaimanapun, untuk sebagian besar pengguna pointer, ingatlah untuk memberi satu tanda bintang per pointer.

5. Pointer dan Array

Konsep dari sebuah array sangat erat kaitannya dengan sebuah pointer. Faktanya, pengenalarray berhubungan dengan alamat elemen pertamanya, seperti pointer yang berhubungan dengan alamat elemen pertama yang ditunjuknya. Jadi sebenarnya, konsepnya sama. Misalnya, mari kita ambil dua pernyataan ini:

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

Operator penugasan berikut akan valid:

```
p = numbers;
```

Kemudian p dan bilangan-bilangan tersebut akan setara dan akan memiliki sifat yang sama. Satu-satunya perbedaan adalah kita dapat mengubah nilai penunjuk p ke yang lain, sedangkan angka selalu menunjuk ke yang pertama dari 20 elemen tipe int yang mendefinisikannya. Jadi tidak seperti p, yang merupakan pointer bersama, angka-angka tersebut adalah larik dan larik dapat dianggap sebagai penunjuk konstan. Oleh karena itu, tugas berikut tidak akan valid:

```
numbers = p;
```

Karena numbers adalah array, ini bertindak sebagai penunjuk konstan dan kami tidak dapat menetapkan nilai ke konstanta.

Karena properti variabel, semua ekspresi yang mengandung pointer dalam contoh berikut benar-benar valid:

```
// morepointers
#include <iostream>
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;
}
```

Dalam pertemuan tentang array, kita menggunakan tanda kurung siku ([]) beberapa kali untuk menunjukkan indeks elemen dalam array yang ingin kita rujuk. Nah, operator braket ini [] juga merupakan operator dereferensi yang dikenal sebagai operator lapisan. Mereka mendereferensi variabel yang diikuti dengan cara yang sama seperti *, tetapi juga menambahkan angka dalam tanda kurung ke alamat yang akan dideferensi. Sebagai contoh:

```
a[5] = 0;
*(a+5) = 0;
```

Kedua ekspresi ini setara dan valid baik jika a adalah pointer atau jika a adalah array.

6. Pointer inisislisasi

Saat mendeklarasikan pointer kita mungkin ingin secara eksplisit menentukan variabel mana yang kita ingin mereka tunjuk:

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

Perilaku kode ini setara dengan:

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

Saat inisialisasipointer terjadi, kami selalu menetapkan nilai referensi yang ditunjukkan oleh pointer (tommy), bukan nilai yang diarahkan ke (* tommy). Perlu dicatat bahwa saat mendeklarasikan pointer, tanda bintang (*) hanya menunjukkan bahwa itu adalah pointer, bukan operator dereferencing (meskipun keduanya menggunakan tanda yang sama: *). Ingatlah bahwa ini adalah dua fungsi karakter yang berbeda. Oleh karena itu, kita harus berhati-hati untuk tidak mengacaukan kode di atas dengan:

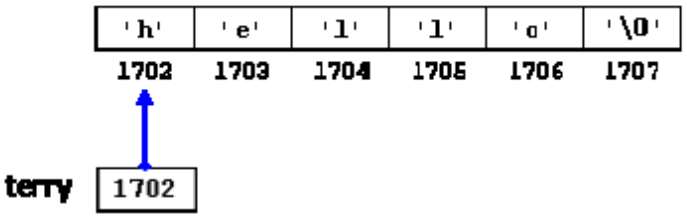
```
int number;
int *tommy;
*tommy = &number;
```

Ini salah, dan bagaimanapun juga, tidak masuk akal jika Anda memikirkannya. Seperti halnya array, compiler mengizinkan kasus khusus yang kita inginkan untuk menginisialisasi konten yang ditunjukkan oleh pointer dengan konstanta sambil mendeklarasikan pointer:

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

Dalam hal ini, memori dicadangkan untuk menampung "Halo" dan Terry kemudian diberikan penunjuk ke karakter pertama dalam blok memori itu. Jika kita membayangkan bahwa "halo" disimpan di lokasi yang dimulai pada

alamat 1702, kita dapat merepresentasikan pernyataan di atas sebagai berikut:



Gambar 3.5 Inisialisasi Pointer

Penting untuk dicatat bahwa Terry berisi nilai 1702, bukan 'h' atau 'hello', meskipun 1702 adalah alamat keduanya. Pointer Terry menunjuk ke sebuah string dan dapat dibaca sebagai sebuah array (ingat bahwa sebuah array seperti pointer konstan). Misalnya, dengan salah satu elemen ini kita dapat mengakses elemen kelima dari array dengan dua ekspresi :

*(terry+4) terry[4]

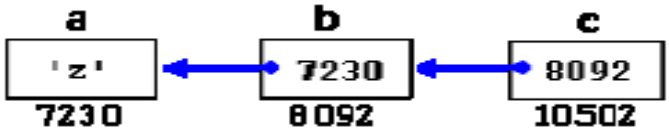
Both expressions have a value of 'o' (the fifth element of the array).

7. Pointer to pointer

C ++ memungkinkan penggunaan pointer yang mengarah ke pointer, yang, pada gilirannya, mengarah ke data (atau bahkan ke pointer lain). Untuk melakukan itu, kita hanya perlu menambahkan asterisk (*) untuk setiap level referensi dalam deklarasinya:

```
char a;
char * b;
char ** c;
a = 'z';
b = &a;
c = &b;
```

Dengan asumsi lokasi memori yang dipilih secara acak untuk setiap variabel dari 7230, 8092, dan 10502 dapat direpresentasikan sebagai berikut:



Gambar 3.6 Pointer to Pointer

Nilai setiap variabel ditulis di setiap sel. Di bawah sel adalah alamat masing-masing di memori. Baru dalam contoh ini adalah variabel c, yang dapat digunakan pada tiga tingkat petunjuk yang berbeda, masing-masing sesuai dengan nilai yang berbeda:

- •c has typechar** and a valueof 8092
- • *c has typechar* and a valueof 7230
- • **c has typecharand a valueof 'z'

C. SOAL LATIHAN/TUGAS

1. Apa yang di maksud dengan pointer?
2. Sebutkan dan jelaskan jenis-jenis operator pointer!
3. Apa yang di maksud dengan variabel pointer?
4. Buatlah program sederhana menggunakan pointer!
5. Buatlah program dengan menggunakan kombinasi antara pointer dengan array!

D. REFERENSI

Soulié, Juan. 2007. *C++ Language Tutorial, Juni 2007. Diambil dari :*

https://www.cplusplus.com/files/tutorial.pdf (6 November 2020)

Moh.Sjukani . 2014. ALGORITMA (Algoritma & Struktur Data 1) dengan *C, C++ dan Java, Edisi 9, Mitra Wacana Media.*