Pointer adalah variabel yang menunjuk pada variabel lainnya. Sebelumnya pernah dijelaskan mengenai variabel, menyatakan bahwa. Variabel merupakan sebuah memori perwakilan dari alamat memori pada komputer.

Hal yang sebenarnya terjadi adalah pointer itu menyimpan alamat memori yang dia tunjuk. Pada pointer kita dimungkinkan untuk menunjuk suatu memori, mengubah nilai dan menyalin nilai memori tersebut secara tidak langsung (perantara melalui variabel pointer).

Sebelum kita mempelajari pointer, ada dua hal yang perlu anda ketahui. Dalam pointer terdapat dua macam operator yang akan kita gunakan, yaitu Address-of (&) dan Dereference operator (\*).

**Address-of Operator (&),** adalah operator yang memungkinkan kita untuk mendapatkan/melihat alamat memori yang dimiliki oleh variabel tersebut. Cara menggunakanya adalah dengan meletakan tanda & di depan identitas saat pemanggilan variabel. Hal itu akan membuat compiler memberikan alamat memori bukan isi/nilai dari memori tersebut.

#include <iostream>

using namespace std;

int main()

{

int var = 2;

cout<<&var<<" memiliki nilai "<<var<<endl;

return 0;

}

**Dereference Operator (\*),** adalah operator yang memungkinkan mendapatkan isi/nilai dari sebuah memori berdasarkan alamat memori.

#include <iostream>

using namespace std;

int main()

{

int var = 2;

cout<<var<<endl;

cout<<&var<<endl;

cout<<\*&var<<endl;

return 0;

}

Pengertian Pointer

Pointer adalah sebuah variabel yang menyimpan alamat memori dari variabel lainnya, kita dimungkinkan untuk mengubah nilai dari variabel yang ditunjuk oleh pointer dan menyalin. Selama program berjalan kita bebas mengubah tujuan dari pointer, mengubah untuk menunjuk ke alamat memori variabel lain atau menunjuk ke alamat memori yang bukan merupakan variabel.

[[[Gambar]]]

Mendirikan Pointer

Setelah anda mengerti fungsi-fungsi dari dua operator yang telah di jelaskan di atas. Sekarang anda dapat mendirikan pointer dengan menggunakan dua operator tersebut.

Bentuk Penulisan

tipeData \*identitas;

//atau

tipeData \*identitas = &var;

Contoh Penulisan

int \*pInt;

double \*pDouble = &myVar;

pada umumnya pointer sebenarnya adalah variabel, peraturan yang dimiliki variabel juga berlaku pada pointer, jadi tidak jauh beda dengan variabel. pointer hanya mendapatkan beberapa perbedaan yaitu penambahan dua operator yang akan membuat variabel menjadi variabel pointer.

Untuk mendirikan sebuah variabel pointer kita hanya menambahkan dereference operator sebelum identitas. Operator dereference tidak harus melekat pada identitas, operator tersebut juga bisa di letakan setelah tipe data atau di antara tipe data dan identitas. dari berbagai cara penulisan tersebut memiliki makna yang sama yaitu satu operator dereference hanya akan berlaku pada saru variabel.

Pointer tidak hanya berlaku pada variabel, kita juga dapat melakukanya pada function, objek dan lain-lain. Dan cara implementasi pointer selain pada variabel, caranya masih sama seperti kita melakukanya pada variabel.

Cara Mengakses Pointer

Variabel pointer adalah variabel yang memiliki alamat memori sebagai nilai dari variabel pointer tersebut. Dan pada pointer kita dimungkinkan untuk mengakses nilai dari pointer itu sendiri dan mengakses nilai dari alamat memori yang dimiliki(ditunjuk) oleh pointer.

Pointer merupakan variabel, untuk mengakses pointer tidak jauh beda dengan cara mengakses variabel. Untuk mengakses nilai dari pointer kita hanya cukup memanggil identiatas dari pointer tersebut.

pInt

pemanggilan itu akan menghasilkan nilai dari pointer yang berupa alamat memori dari variabel yang ditunjuk oleh pointer tersebut.

Karena pointer hanya dapat mememiliki nilai berupa alamat memori, untuk mengubah nilai dari pointer atau mengubah tujuan dari pointer kita membutuhkan operator address-of (&) pada operand sumber.

pInt = &myVar

operand sumber akan menghasilkan alamat memori dari myVar, dan hal itu merupakan nilai yang dibutuhkan oleh variabel pointer.

Sebelum anda mempraktikan untuk mengubah nilai dari variabel pointer ada satu hal yang perlu anda tau. Bahwa variabel pointer hanya dapat meneriman alamat memori dari variabel yang memiliki tipe data yang sama.

Contoh Program :

#include <iostream>

using namespace std;

int main()

{

int var1 = 2, var2 = 5;

int \*pVar = &var1;

cout<<"var1 = "<<&var1<<endl;

cout<<"pVar = "<<pVar<<endl;

cout<<"====="<<endl<<endl;

pVar=&var2;

cout<<"var2 = "<<&var2<<endl;

cout<<"pVar = "<<pVar<<endl;

return 0;

}

Variabel pointer dapat menyimpan alamat memori bukan berarti anda dapat bebas memberi alamat memori secara langsung pada pointer.

int \*pInt = 0x012345;

Hal di atas tidak dimungkinkan untuk dilakukan.

Karena isi/nilai dari variable pointer merupakan sebuah alamat memori maka untuk mengakses nilai dari variabel yang dutunjuk oleh pointer kita membutuhkan operator dereference (\*).

\*pInt

Dalam pointer kita dimugkinkan untuk menyalin dan mengubah nilai pada variabel yang ditunjuk oleh pointer.

\*pInt = 2

Sebelum anda mencoba mengubah nilai dari memori yang ditunjuk oleh pointer anda harus yakin bahwa memori itulah yang ingin anda nilainya diubah. Jangan pernah mengubah memori yang bukan milik program anda (memori yang belum di alokasi), karena hal itu terlalu memiliki resiko yang cukup serius untuk komputer anda. Dan satu hal yang perlu anda ingat adalah, berilah nilai awal di saat deklarasi pointer. Itu dapat menimalisir kemungkinan kecelakaan.

Contoh Program :

#include <iostream>

using namespace std;

int main()

{

int var = 2;

int \*pVar = &var;

cout<<"var = "<<var<<endl;

cout<<"pVar = "<<\*pVar<<endl;

cout<<"====="<<endl<<endl;

\*pVar= 109;

cout<<"var = "<<var<<endl;

cout<<"pVar = "<<\*pVar<<endl;

return 0;

}

Ukuran pointer

Setiap kita medirikan pointer, pointer itu akan membutuhkan memori. Dan besar memori itu sama pada setiap tipe data yang digunakan.

Besar memori dari pointer tergantung pada mesin kompiler. jika kompiler merupakan 32bit maka pointer akan memakan memori sebanyak 4 bytes, Jika menggunakan 64bit maka pointer memakan memori sebanyak 8 bytes.

#include <iostream>

using namespace std;

int main()

{

int var = 2;

int \*pVar = &var;

cout<<sizeof(pVar)<<endl;

return 0;

}

Void Pointer

Void pointer adalah pointer yang menggunakan tipe data bertipe void, pointer yang mengunakan tipe data void dimungkinkan untuk bebas menunjuk ke berbagai tipe data dari variabel atau obyek.

Contoh penulisan

void \*ptr;

Contoh program

#include <iostream>

int main()

{

int myInt = 1;

float myFloat = 2;

double myDouble = 3;

void \*ptr = nullptr;

ptr = &myInt;

std::cout<<ptr<<" = "<<&myInt<<"\n";

ptr = &myFloat;

std::cout<<ptr<<" = "<<&myFloat<<"\n";

ptr = &myDouble;

std::cout<<ptr<<" = "<<&myDouble<<"\n";

return 0;

}

Pada contoh di atas membuktikan bahwa variabel pointer “ptr” yang menggunakan tipe data void bebas untuk menunjuk ke berbagai tipe data dari variabel atau obyek. Tapi kelemahan untuk pointer menggunakan tipe data void adalah nilai dari memori yang ditunjuk tidak bisa didapatkan menggunakan operator dereference.

std::cout<<\*ptr; //error

itu juga akan menjelaskan kenapa pointer hanya dapat menunjuk ke variabel atau obyek yang memiliki tipe data yang sama. Yang sebenarnya terjadi adalah bahwa pointer menunjuk bukan menunjuk ke variabel atau obyek tersebut tetapi menunjuk ke alamat memori dari variabel atau obyek yang di tunjuk.

Memori sebenarnya hanya menyimpan nilai tidak menyimpan tipe dari nilai tersebut, yang menyimpan tipe dari nilai memori adalah variabel. jadi ketika CPU menerima perintah untuk membaca nilai dari memori melalui pointer void, CPU tidak akan dapat melakukanya karena tanpa tipe data CPU tidak mengerti bagaimana cara membaca nilai tersebut.

Hal tersebut bukan berarti tidak ada cara untuk mendapatkan nilai dari memori yang ditunjuk oleh pointer void. Jika CPU tidak bisa melakukanya, tapi kita bisa membantunya dengan memberitahukan tipe dari nilai tersebut. untuk melakukan itu kita bisa menggunakan static\_cast<>().

Contoh Program

#include <iostream>

int main()

{

int myInt = 1;

void \*ptr = nullptr;

ptr = &myInt;

std::cout<<ptr<<" = "<<&myInt<<" = "<<\*static\_cast<int\*>(ptr)<<"\n";

return 0;

}

Pointer Konstanta

Konstanta adalah variabel yang memiliki nilai yang tetap dan tidak bisa diubah. Saat mendirikan pointer kita dimungkinkan untuk menggunakan fitur konstanta. Ini dapat membuat variabel pointer bersifat konstanta pada nilai pointer atau nilai memori yang ditunjuk.

Contoh Penulisan

const int \*PTR = &var;

pada materi sebelumnya diberitahukan bahwa kita dibebaskan meletakan keyword const sebelum atau sesudah tipe data, kedua cara tersebut akan berpengaruh sama.

const int \*PTR = &var;

int const \*PTR = &var;

dari penulisan di atas akan membuat nilai dari memori yang ditunjuk oleh variabel pointer tidak dapat diubah.

Contoh Program

#include <iostream>

int main()

{

int myInt = 1;

const int \*PTR = nullptr;

PTR = &myInt;

\*PTR = 2; //error

return 0;

}

Kita dapat meletakan keyword const sebelum atau setelah tipe data. kita juga dapat meletakan keyword const setelah operator dereference (\*), tapi hal itu akan memberi pengaruh berbeda.

Contoh penulisan

int \*const PTR = &var;

dari penulisan di atas akan memuat nilai dari variabel di atas tidak bisa di ubah. Kita tidak dapat merubah tujuan kemana pointer itu mengarah.

Contoh program

#include <iostream>

int main()

{

int myInt = 1;

int \*const PTR = nullptr;

PTR = &myInt; //error

\*PTR = 2;

return 0;

}

Bukan hanya itu kita juga dimungkinkan untuk menggunakan dua keyword const yang memiliki fungsi yang berbeda dalam satu deklarasi. Itu akan membuat nilai dari pointer dan nilai dari memori yang di tunjuk tidak bisa di ubah (nilai tetap).

const int \*const PTR = &var;

jika kalian susah untuk membaca, menghafal perbedaanya atau sulit untuk membedakanya kalian bisa membaca dengan cara terbalik untuk dapat mengerti perbedaanya (apa yang terjadi pada deklarasi).

const int \*PTR = &var; //pointer PTR dereference ke (menunjuk ke) integer konstanta

int \*const PTR = &var; //pointer konstanta PTR dereference ke integer

const int \*const PTR = &var; //pointer konstanta PTR dereference ke integer konstanta

Pointer to Pointer

Pointer tidak hanya bisa menunjuk memori dari variabel biasa atau obyek, kita juga dapat mengarahkanya ke variabel pointer lainnya. Untuk membuat variabel pointer yang dapat menunjuk ke variabel pointer lainnya kita membutuhkan tambahan operator dereference (\*). Jika ingin menunjuk pointer \*ptr kita akan membutuhkan variabel pointer \*\*myptr yang memiliki tingkatan yang lebih tinggi. Untuk menunjuk ke variabel pointer \*\*myptr kita membutuhkan variabel pointer \*\*\*youptr.

Kesimpulanya adalah jika ingin menunjuk ke variabel pointer, kita harus mendirikan variabel pointer yang memiliki satu level lebih tinggi (memiliki satu operator dereference lebih banyak).

Contoh program

#include <iostream>

using namespace std;

int main()

{

int myInt = 1;

int \*ptr1 = &myInt;

int \*\*ptr2 = &ptr1;

int \*\*\*ptr3 = &ptr2;

cout<<ptr3<<endl;

cout<<\*ptr3<<endl;

cout<<\*\*ptr3<<endl;

cout<<\*\*\*ptr3<<endl;

return 0;

}

Dalam pointer ke pointer seperti pada contoh di atas kita juga dimungkinkan untuk mengakses nilai dari pointer yang di tunjuk, atau juga bisa lebih dalam dari itu.