Salah satu tujuan diciptakan bahasa pemrograman C++ adalah menambahkan fitur OOp (Object Oriented programming) pada bahasa pemrogramman C. Class hampir mirip seperti Data Structures(Struct) pada bahasa C, class memiliki fitur yang lebih dari pada Data Structures (Struct) yang sama-sama bisa menampung banyak Data Member (variabel) dan class juga bisa menampung Function Member (method).

Bentuk penulisan

class nama\_class{

akses\_specifier :

data\_member;

…

akses\_specifier2 :

data\_member;

…

}nama\_object;

Untuk mendirikan dan mendefinisikan class, kita butuh untuk memberi keyword “class” dan dilanjutkan dengan nama class tersebut, setelah nama, seperti kita mendirikan function, kita membutuhkan sepasang tanda { } dan pada akhir jangan lupa untuk menaruh tanda titik koma ( ; ). Untuk badan atau definsi dari class akan diapit oleh tanda { }, disitulah tempat untuk daftar beberapa data(member) yang kita perlukan. Dan nama\_object diantara tanda “…}”dan “;” merupakan object dari class tersebut, merupakan pilihan, bebas untuk diikut sertakan atau tidak diikut sertakan.

Jika anda pernah belajar bahasa programming selain bahasa C/C++ sebelumnya, atau yang belum pernah masuk ke dunia programming. Dan pernah mendengar kata Member, property dan Atribut dan bingung mengenai perbedaan Member, property dan Atribut. Ketiganya memiliki makna yang sama. Penggunaan nama dipengaruhi dalam bahasa pemrograman yang anda pelajari/gunakan dan dalam bahasa pemrograman C/C++ banyak programmer C/C++ menggunakan nama “member” (anggota).

Untuk “akses\_specifier :” dan “akses\_specifier2:”merupakan tempat untuk mendefinisikan keyword Akses Specifier (private, protected dan public) yang digunakan untuk mengolongkan dan mengubah sifat akses anggota dari class yang ada di bawah keyword tersebut. untuk penjelasan yang lebih dalam mengenai Akses Specifier akan kami jelaskan pada artikel berikutnya.

Sebenarnya membuat deklarasi class sangat mirip seperti membuat struct yang pernah penulis sampaika pada artikel sebelumnya. Jika kita hanya menulis nama object, setelah “{…}” dan sebelum ‘ ; ’, dan tanpa memberikan nama atau identifier disebelah keyword class maka anda akan membuat struct tanpa pengenal. Nama diantara kedua tanda (object) tersebut akan berlaku sebagai object dan kita tidak bisa membuat object di luar definisi struct.

Ketika kita mendirikan class, kita seperti membuat sebuah kerangka atau blueprint untuk object, nama dari class tersebut yang akan digunakan untuk tipe data dari object. Class dan Object merupakan fitur yang sangat membantu untuk mendirikan sebuah program besar, menjadikan sebuah code atau program yang ditulis oleh programmer mudah untuk dimengerti dan lebih terstruktur, dan juga mudah dalam pemeliharaan. Fungsi yang dsediakan oleh Class lebih banyak dari Data Structures “struct”. Fungsi dari class yaitu untuk membuat kerangka dan membuat kelompok dari variabel (member) bukan hanya variabel yang bisa kita masukan dalam definisi class tapi juga kita dimungkinkan untuk memasukan function (method) dalam class, constructor, destructor dan overloading. Diluar definisi class kita juga dimungkinkan untuk membuat realasi seperti inheritance dan overriding.

Contoh penulisan

class mahasiswa{

public:

int nim;

string nama;

float nilai;

void printData(){

cout<<"NIM \t= "<<nim<<endl;

cout<<"Nama \t= "<<nama<<endl;

cout<<"Nilai \t= "<<nilai<<endl;

}

}anton;

Diatas adalah sepenggal program yang mencontohkan bagaimana cara mendirikan dan mendefinisikan class dengan nama class “mahasiswa”. Di dalam class pada baris pertama isi dari class terdapat akses specifier “public” yang berarti semua member atau anggota dibawah label itu akan bersifat bebas untuk diakses dari dalam maupun dari luar class.

Garis kedua sampai empat isi atau definisi dari class terdapat tiga variabel “nim”, “nama” dan “nilai” dengan tipe data int, string dan float. Yang merupakan member variable / field dari class. Dan pada baris berikutnya pada isi atau definisi class terdapat anggota yang berupa member function / method, yang mempunyai definisi atau tugas untuk mencetak nilai dari member diatasnya. Dan terakhir, baris terakhir dari luar class “anton” diantara tanda “}” dan “;” adalah sebuah object. merupakan pilihan anda sesuai kebutuhan anda, untuk tidak memberi object setelah class, memberi sebuah object setelah class seperti di atas, atau memberi lebih dari satu object setelah class tersebut, untuk setiap object akan dipisahkan dengan tanda koma “ , ”. kita juga dapat mendirikan object di luar dari definisi class.

Dalam membuat member function atau method, kita bisa melakukan dengan dua cara yang pertama adalah dengan cara yang telah diperlihatkan pada contoh program di atas, yaitu dengan mendirikan dan mendefinisikan member function di dalam definisi class. Kedua adalah dengan cara mendirikan function di dalam class dan melakukan definisi di luar definisi class. Mirip seperti kita membuat function dengan mendirikan function prototype terlebih dahulu dan mendefinisikan function tersebut sesudah deklarasi function tersebut, kita juga bisa mendefinisikan dibawah main function.

Pada class kita juga dimungkinkan untuk mendirikan function prototype, dan mendefinisikanya di luar definisi class. Untuk memberi definisi pada function tersebut kita dapat melakukanya dengan menggunakan Scope Resolution Operator ( :: ), yang diletakan di antara nama class dan nama member function.

Untuk mendefinisiakan function menggunakan Scope Resolution Operator, kita bebas menaruh definisi tersebut dimana pun seperti, tepat di bawah definsi class dari function prototype tersebut atau di bawah main function dan juga bisa di definisikan diluar file, tapi kita tidak bisa melakukan definisi sebelum deklarasi function prototype.

Bentuk umum penulisan

//Function prototype

return\_type nama\_function(arguments);

//Definisi Function

return\_type nama\_class :: nama\_function(){

…

}

Contoh penulisan

class mahasiswa{

public:

int nim;

string nama;

float nilai;

void printData();

};

void mahasiswa::printData(){

cout<<"Definisi Function menggunakan Scope Resolution Operator"<<endl;

}

Kita juga bisa menggunakan Scope Resolution Operator untuk menginisialisasi member variabel dari class tapi hanya dapat digunakan pada member variabel yang berjenis static. Untuk penjelasan mengenai static member penulis akan sampaikan pada artikel berikutnya.

**Cara mendirikan Object**

Seperti apa yang telah dijelaskan, bahwa class hanyalah sebuah sebuah rancangan, bisa dibilang mereka tidak nyata. Dan object merupakan object atau benda nyata yang bisa kita gunakan dan memiliki sifat seperti rancangan yang telah kita buat pada class. Untuk mendirikan sebuah object terdapat dua cara yaitu.

Pertama, seperti contoh sebelumnya, bahwa kita dapat membuat object sesudah definisi dari class yaitu tepat diantara tanda “}” dan “;”. Memberi object setelah definisi class tidak diharuskan, tapi dapat dilakukan, kita bisa membuat 1 object atau lebih object dengan memisahkan masing-masing object dengan tanda koma “,”. Object yang didirikan bersamaan dengan class akan mempunyai sifat akses global.

Contoh penulisan

class mahasiswa{

public:

int nim;

string nama;

float nilai;

void printData(){

cout<<"NIM \t= "<<nim<<endl;

cout<<"Nama \t= "<<nama<<endl;

cout<<"Nilai \t= "<<nilai<<endl;

}

}anton, budi, tono;

Kedua, kita cukup mendirikan seperti halnya kita membuat variabel biasa. Mengganti tipe data yang biasanya kita menggunakan tipe data yang disediakan oleh c++ menggantinya dengan nama class yang kita tuju, dan diikuti dengan nama object sebabai identifier.

Contoh penulisan

mahasiswa anton;

mahasiswa budi, tono;

contoh program

#include <iostream>

using namespace std;

class mahasiswa{

public:

int nim;

string nama;

float nilai;

void printData(){

cout<<"NIM \t= "<<nim<<endl;

cout<<"Nama \t= "<<nama<<endl;

cout<<"Nilai \t= "<<nilai<<endl;

}

}anton,tono;

int main() {

mahasiswa tanto,budi,adi;

return 0;

}

Sama seperti kita mendirikan object dari struct, union dan lain-lain, dalam mendirikan object class kita bebas untuk menambahkan keyword “class” atau tidak menambahkannya sebelum nama class tersebut. semua bentuk penulisan memiliki fungsi yang sama.

Di atas adalah contoh program dari cara penggunaan kedua cara untuk membuat object. object anton dan tono didirikan setelah mendirikan class tersebut di antara tanda “ } ” dan “ ; ”. dan object tanto, budi dan adi didirikan di dalam jangkauan main function, seperti mendirikan variabel biasa, object didirikan dengan memberi nama class dan di ikuti dengan nama object tersebut. jika anda coba untuk mengkompilasi contoh program di atas, contoh itu tidak menampilkan pengeluaran, disini penulis hanya memberi contoh bahwa kedua cara yang dimaksud, bekerja dengan baik.

**Cara Mengakses Object**

Setelah kita berhasil mendirikan object, object itu akan mempunyai rancangan, data-data yang telah kita rencanakan dalam pembuatan class sebelmnya. Kita bebas mengakses, memberi nilai, mengubah nilai pada member class tersebut melalui object yang telah kita buat. Untuk mengakses Object kita membutuhkan ”Member Access Operator ( . )” , kita membutuhkan tanda “ . “ diantara nama object dan nama member / anggota class.

Contoh penulisan

anton.nim;

anton.nama;

anton.nilai;

anton. printData();

Contoh program

#include <iostream>

using namespace std;

class mahasiswa{

public:

int nim;

string nama;

float nilai;

void printData(){

cout<<"NIM \t= "<<nim<<endl;

cout<<"Nama \t= "<<nama<<endl;

cout<<"Nilai \t= "<<nilai<<endl;

}

};

int main() {

mahasiswa anton;

anton.nim=1234;

anton.nama="Anton Cpluzer";

anton.nilai=95;

anton.printData();

return 0;

}

**Cara Inisialisasi Object**

Sejauh ini jarang sekali pernulis lihat orang yang inisialisasi member dari class langsung tidak melalui constructor, yang biasa penulis lihat adalah inisialisasi pada struct, mungkin dikarenakan member untuk class bukan hanya bisa digunakan untuk variabel. tapi juga method. Dan kebanyakan orang menulis hampir ribuan baris code hanya untuk class. Dan itu akan membingungkan jika inisialisasi langsung pada saat pembuatan object dari class tersebut, karena. Untuk dapat meberi nilai awal kita harus tau urutan dari deklarasi member variabel dalam class tersebut, dan sulit untuk mengabaikan beberapa member yang seharusnya tidak di inisialisasi, dengan arti kita harus inisialiasi, memberi nilai dengan urut berdasarkan urutan variabel yang anda dirikan dalam class tidak peduli kalau bariabel itu ada dimanapun dan harus urut, dan jika kita menginisialisasikan member satu persatu langsung pada deklarasi member tersebut, kemungkinan akan sulit untuk mencari dan mengubah nilai default tersebut. Meskipun itu sulit dan membingungkan tapi kita masih di mungkinkan untuk menginisialisasi object pada member di class. ada tiga cara untuk inisialisasi object atau class, yaitu :

1. Non-static Member Initialization, merupakan cara untuk memberikan nilai awal (Inisialisasi) di saat mendirikan variabel sebagai member class. Maka nilai awal itu akan berlaku pada semua object yang menggunakan class tersebut. dan cara ini hanya tersedia pada C++11.

Contoh program

#include <iostream>

using namespace std;

class mahasiswa{

public:

int nim=1; //inisialisasi

string nama="default name"; //inisialisasi

float nilai=0; //inisialsiasi

void setterNilai (float aNilai){

nilai=aNilai;

}

void printData(){

cout<<"NIM \t= "<<nim<<endl;

cout<<"Nama \t= "<<nama<<endl;

cout<<"Nilai \t= "<<nilai<<endl;

}

};

int main() {

mahasiswa anton,budi;

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

anton.printData();

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

budi.printData();

return 0;

}

2.Uniform Initialization, merupakan cara kedua untuk inisialisasi member pada object. nilai awal tersebut akan kita berikan kepada member class melalui object tersebut, disebutkan dengan urut di saat mendirikan object tersebut. tidak memiliki sifat seperti cara sebelumnya, inisialisasi ini hanya akan berlaku pada object itu sendiri. Tapi kelemahanya, cara ini hanya bisa dilakukan jika semua member dari class bersifat public.

Contoh penulisan

mahasiswa anton = {123,"Antonia uvava osas",98.5}; //atau

mahasiswa albert {234,"Albert Insting", 101.5}; //C++11

Contoh program

#include <iostream>

using namespace std;

class mahasiswa{

public:

int nim;

string nama;

float nilai;

void printData(){

cout<<"NIM \t= "<<nim<<endl;

cout<<"Nama \t= "<<nama<<endl;

cout<<"Nilai \t= "<<nilai<<endl;

}

};

int main() {

mahasiswa anton = {123,"Antonia uvava osas",98.5};

mahasiswa albert {234,"Albert Insting", 101.5};

anton.printData();

albert.printData();

return 0;

}

3.Member Initialization In Constructor, merupakan cara ketiga yang penulis maksud pada pengalaman penulis di atas. Cara ini adalah cara yang sering penulis gunakan dan penulis lihat pada code-code programmer C++ lain dalam menginisialisasi object. itu karena cara ini memiliki banyak kelebihan,seperti. Lebih aman dalam inisialisasi class dan juga fleksibel. Tidak seperti uniform inisialisasi, dengan cara ini kita dapat bebas menginisialisasi tanpa mengkhawatirkan urutan deklarasi member variabel pada class tersebut dan tidak perlu kawatir tentang member yang bersifat private atau protected. karena kita bisa mengatur argument yang diberikan untuk inisialisasi member class dan kita juga bisa menggunakan constructor overloading untuk membuat inisialisasi lebih fleksibel. Untuk penjelasan overloading dan constructor yang lebih dalam, akan penulis sampaikan pada artikel selanjutnya. dibawah ini adalah contoh potongan program,cara inisialisasi object dengan menggunakan constructor.

Bentuk penulisan Constructor

Class nama\_class{

nama\_class (parameters){

definisi…

}

};

Di atas adalah bentuk penulisan dasar dari constructor. Sama seperti cara kita mendirikan function tapi untuk membuat constructor, nama constructor tersebut harus menggunakan nama classnya. Dengan begitu constructor akan dieksekusi langsung ketika pembuatan object dari class tersebut.

Contoh penulisan Constructor (Member Initialization)

mahasiswa(int pNim, string pNama, float pNilai){

nim = pNim;

nama = pNama;

nilai = pNilai;

}

//atau (2)

mahasiswa(int pNim, string pNama, float pNilai):nim(pNim){

nama = pNama;

nilai = pNilai;

}

//atau (3)

mahasiswa(int pNim, string pNama, float pNilai):nim(pNim),nama(pNama),nilai(pNilai){}

//atau (4)

mahasiswa(int pNim, string pNama, float pNilai):nim{pNim},nama{pNama},nilai{pNilai}{}

pada contoh penulisan nomer 2,3 dan 4 adalah contoh inisialisasi menggunakan Initializer List. Dengan menggunakan tanda “ : ”, dilanjutkan dengan inisialisasi anggota lainya di dalam class, dan bisa tanpa menggunakan badan constructor. Pada contoh-contoh di atas meskipun penulisanya berbeda tapi tidak memiliki perbedaan untuk inisialisasi member, untuk mengetahui perbedaannya secara detail akan penulis sampaikan di artikel selanjutnya.

Contoh penulisan (Inisialisasi pada Object)

mahasiswa budi(135, "Budi Kubudi", 86.7);

mahasiswa albert {234,"Albert Insting", 101.5};

mahasiswa anton = {123,"Antonia uvava osas",98.5};

Contoh program

#include <iostream>

using namespace std;

class mahasiswa{

public:

int nim;

string nama;

float nilai;

mahasiswa(int pNim, string pNama, float pNilai){

nim = pNim;

nama = pNama;

nilai = pNilai;

}

//overloading

mahasiswa(string pNama, int pNim, float pNilai):nim(pNim),nama(pNama),nilai(pNilai){}

mahasiswa(int pNim, float pNilai, string pNama):nim{pNim},nama{pNama},nilai{pNilai}{}

void printData(){

cout<<"NIM \t= "<<nim<<endl;

cout<<"Nama \t= "<<nama<<endl;

cout<<"Nilai \t= "<<nilai<<endl;

}

};

int main() {

mahasiswa budi(135, "Budi Kubudi", 86.7);

mahasiswa albert {"Albert Insting", 234, 101.5};

mahasiswa anton = {123, 98.5, "Antonia uvava osas"};

budi.printData();

anton.printData();

albert.printData();

return 0;

}

Pada contoh di atas dicontohkan dari penggunaan insialisasi dengan constructor menggunakan 3 cara yang disebutkan di atas. Sekaligus memperkenalkan anda mengenai overloading constructor, tidak perlu bingung mengenai overloading dan constructor jika anda belum mengetahui atau memahaminya karena akan penulis jelaskan mengenai Overloading dan constructor lebih dalam di artikel selanjutnya.

pada (C++ 11) Ada satu hal yang harus anda ingat, ketika anda menggunakan cara ke satu (Non-static Member Initialization), anda tidak bisa menggunakan cara kedua (Uniform Initialization) bersamaan, Begitu pula sebaliknya. inisialisasi adalah tentang hal memberi nilai awal. jika cara ke satu sudah dilakukan, melakukan cara kedua bukan lagi sebuah inisialisasi. Tapi pada (C++ 14) anda bisa menggunakan kedua cara secara bersamaan.

Dan jika anda sudah menggunakan constructor maka uniform inisialisasi akan dianggap sebagai argument untuk constructor tersebut.

Contoh program

#include <iostream>

using namespace std;

class mahasiswa{

public: //akses Specifier

//Members / Anggota

int nim;

string nama;

float nilai;

mahasiswa(int pNim, string pNama, float pNilai); //constructor untuk inisialisasi

void printData(); //Method

};

//Scope Resolution Operator

mahasiswa::mahasiswa(int pNim, string pNama, float pNilai){

nim = pNim;

nama = pNama;

nilai = pNilai;

}

void mahasiswa::printData(){

cout<<"NIM \t= "<<nim<<endl;

cout<<"Nama \t= "<<nama<<endl;

cout<<"Nilai \t= "<<nilai<<endl;

}

int main() {

//inisialisasi

mahasiswa budi(135, "Budi Kubudi", 86.7);

//Mengakses

budi.nim = 1234567;

budi.nama = "Budi Frost";

budi.nilai = 100;

budi.printData();

return 0;

}