Inisialisasi (Memberi Nilai Awal) pada sebuah object memiliki banyak cara, salah satunya adalah menggunakan constructor. Constructor memberi kemudahan dan fleksibilitas untuk melakukan inisialisasi pada member-member object. Seperti pada artikel sebelumnya yang menjelaskan tentang constructor penulis sebelumnya memberi contoh inisialisasi constructor menggunakan assignment operator (Operasi penugasan), yang ditulis di dalam definisi constructor.

class siswa{

private:

string nama;

float nilai;

public:

siswa(string pNama, int pNilai){ //Constructor

nama = pNama; //Assignment Operator

nilai = pNilai; //Assignment Operator

}

};

Kebenarannya adalah cara di atas bukan inisialisasi. Tapi cara tersebut masih diperbolehkan digunakan untuk inisialisasi jika itu masih bisa. Kita tau bahwa, pada dasarnya member diciptakan setelah class tersebut tercipta, setelah member diciptakan saat itulah constructor di jalankan untuk melakukan persiapan pada object yang sudah di atur salah satunya memberi nilai kepada member object. jika anda membuat perintah Inisialisasi didalam definisi constructor itu sama seperti kita melakukan hal seperti ini.

string nama

float nilai

nama = “belajarcpp”

nilai = 90

di atas adalah potongan program dari contoh yang pertama hanya saja penulis menyingkirkan semua hal tentang class. Cara kerjanya mirip seperti yang dijelaskan di atas bahwa, variabel akan diciptakan dan setelah itu diberi nilai sesuai operator penugasan di bawah. Tapi definisi inisialisasi adalah menciptakan variabel/member dengan nilai yang ditentukan (nilai langsung diberikan setelah tercipta). Dibawah adalh bukti kuat bahwa cara di atas bukan sebuah inisialisasi.

class siswa{

private:

const string nama;

const float nilai;

public:

siswa(string pNama, float pNilai){

nama = pNama;

nilai = pNilai;

}

};

Di atas adalah contoh potongan program yang masih sama seperti contoh potongan program yang pertama, penulis hanya mengubah member “nama” dan “nilai” menjadi member/variabel konstanta. Disaat kita mencoa untuk menjalankan program di atas akan menghasilkan pesan error yaitu, bahwa variabel / member konstanta tidak bisa didirikan tanpa di inisialisasi dan kedua nilai dari variabel / member konstanta tidak bisa di ubah, itu yang terjadi pada perintah assignment operator di dalam definisi constructor.

class siswa{

private:

string nama = “belajarCpp”;

const float nilai = 90;

};

Di atas merupakan Cara Non-static Member Initialization (C++11) yang dapat menjadi salah satu solusi kita untuk menginisialisasi member-member dalam class termasuk member konstanta. Tapi cara itu hanya akan berlaku pada class, yang berarti semua object yang menggunakan class tersebut akan memiliki nilai sama saat pembuatan, kita tidak bisa menentukan nilai dari member-member pada masing-masing object saat pembuatan object yang kemungkian object tersebut membutuhkan nilai yang berbeda. Dan juga cara tersebut sama sekali tidak mengikut sertakan constructor.

Untuk melakukan inisialisasi member-member object, Dalam C++ tersedia metode inisialisasi yang bernama “Member Initialization List” atau juga bisa disebut sebagai “Initializer List”

**Member Initialization List**

Member Initialization List adalah cara yang disediakan dalam bahasa pemrograman c++ untuk mempermudah kita dalam insisialisasi member variabel pada setiap object, yang kemungkinan kita membutuhkan nilai yang berbeda pada setiap object yang kita buat.

Member Initialization List memang dikhususkan untuk inisialisasi member-member variabel pada object, tidak seperti yang kita lakukan sebelumnya (menggunakan assignment Operator untuk inisialisasi), dengan menggunakan Member Initialization List kita dapat menginisialisasi variabel constanta melalui constrctor.

Untuk melakukan Member Initialization List kita dapat menaruh inisialisasi tepat setelah deklarasi constructor, kita tambahkan tanda titik dua “ : ”setelah tanda “ ) ” / ( Tepat di antara deklarasi constructor dengan Member Initialization List ).

Setelah tanda titik dua kita bebas untuk inisialisasi member variabel sesuai yang kita butuhkan, gunakan cara inisialsasi Direct Initialization atau biasanya disebut constructor initialization yang telah penulis jelaskan pada artikel sebelumnya. Jika kita melakukan lebih dari satu inisialisasi kita dapat tambahkan tanda koma ( , ) diantara setiap inisialisasi. Untuk mengakhiri, lanjutkan dengan definisi dari constructor tersebut “ ( //definisi } “

Bentuk Umum penulisan

class nama\_class{

public:

nama\_class(parameters):member\_var(nilai),member\_var2(nilai){

//Definisi Constructor

}

};

Contoh program

#include <iostream>

#include <string>

using namespace std;

class siswa{

private:

const int id;

string nama;

float nilai;

public:

siswa(int pId, string pNama, float pNilai):id(pId),nama(pNama),nilai(pNilai){ //Member Initialization List

//Definisi

}

~siswa(){

cout<<"Id = "<<id<<endl;

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

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

}

};

int main(){

siswa andi(12,"Budi Andik", 90.5);

return 0;

}

Di atas adalah contoh program yang sama seperti contoh-contoh sebelumnya di artikel ini, penulis hanya menambahkan destructor untuk mencetak saat program berakhir. Dan member variabel bersifat konstanta. Kita dapat melihat bahwa inisialisasi menggunakan Member Initialization List bekerja dengan baik, terutama dalam inisialisasi konstanta.

Dalam penulisan Member Initialization List kita juga dapit melakukan penulisan sebagai berikut.

//Bentuk 1

siswa(int pId, string pNama, float pNilai):id(pId),nama(pNama),nilai(pNilai){

//Definisi

}

//Bentuk 2

siswa(int pId, string pNama, float pNilai)

:id(pId),nama(pNama),nilai(pNilai){

//Definisi

}

Bentuk 3

siswa(int pId, string pNama, float pNilai):

id(pId),

nama(pNama),

nilai(pNilai)

{

//Definisi

}

**Constructor Delegation**

Constructor Delegation atau biasanya disebut Constructor Chaining adalah metode yang memungkinkan kita untuk membuat constructor yang dapat memanggil constructor lain. Hal ini biasanya dimanfaat oleh programmer untuk mengurangi jumlah pengulangan kode yang sama terutama pada overloading constructor. Untuk melakukanya adalah dengan cara memanggil constructor lainya dari dalam Member Initiaziation List (Bukan Definisi Constructor).

Contoh penulisan

class siswa{

public:

siswa(){

cout<<"Default Constructor"<<endl;

}

siswa(string nama):siswa(){

cout<<nama<<endl;

}

};

Di atas adalah contoh penulisan yang menunjukan bagaimana cara penulisan Delegating Constructor. Dimana constructor memanggil default constructor di dalam Member Initialization List. Dan di bawah ini adalah Contoh program dari Constructor Delegation untuk mengurangin pengulangan kode pada Overloading Contstrucor.

Contoh program

#include <iostream>

#include <string>

using namespace std;

class siswa{

private:

const int id;

string nama;

float nilai;

public:

siswa(int pId=0, string pNama="", float pNilai=0):id(pId),nama(pNama),nilai(pNilai){ //Member Initialization List

//Definisi

}

siswa(string pNama):siswa(0,pNama,0){}

siswa(float pNilai):siswa(0,"",pNilai){}

siswa(string pNama, float pNilai):siswa(0,pNama,pNilai){}

~siswa(){

cout<<"Id = "<<id<<endl;

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

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

cout<<endl;

}

};

int main(){

siswa a(1);

siswa b(2,"B",9.8f);

siswa c("C", 8.6f);

siswa d(99.9f);

siswa e;

return 0;

}

Di atas adalah contoh dari Delegatio Constructor beserta cara untuk menimalisir jmlah pengulangan kode. Pada constructor pertama, didirikan default constructor, tidak seperti default constructor sebelumnya yang dicontohkn tidak menggunakan sama sekali parameter, tapi pada Constructor di atas adalah default constructor karena, semua parameter pada constructor itu dibuat menjadi default parameter. Jadi ketika kita tidak memberikan argument saat pemanggilan, parameter akan tetap mempunyai argument.

Dan tiga constructor di bawah default constructor adalah Delegation Constructor. Bandingkan jika kita menginisialisasi member variabel tidak menggunakan Delegation Constructor (ditulis perintah satu persatu) pasti akan memakan banyak baris lagi.

Pada class di atas tidak mempunyai constructor dengan tanda atau parameter yang hanya memilki parameter untuk menginisialisasi id, karena pada default Constructor kita menggunakan default parameter, sifat default parameter hanya bisa di masukan nilai dari kiri ke kanan dan tidak bisa meloncati parameter sebelumnya, itu berarti jika kita hanya memberikan satu argument, argument itu akan diberikan kepada parameter yang ada di kiri, maka parameter yang dikanannya masih mempunyai nilai meskipun tidak diberikan argument saat pemanggilan, itu artinya kita tidak membutuhkan constructor yang hanya memiliki parameter pId.