



Modul Pelatihan

Dasar Pemrograman

Pengenalan Singkat C++

C++ merupakan bahasa pemrograman tingkat menengah yang merupakan pengembangan dari bahasa C, yang dibuat oleh Bjarne Stroustrup pada awal tahun 1980-an. Awalnya, bahasa ini dikenal dengan nama "C dengan Kelas" karena fitur-fitur baru yang diperkenalkan, yaitu penambahan konsep kelas dan objek dari paradigma pemrograman berorientasi objek (OOP) ke dalam bahasa C yang sudah ada.

C++ dikategorikan sebagai middle-level language (bahasa pemrograman tingkat menengah) karena memiliki karakteristik yang menggabungkan fitur dari high-level language (bahasa pemrograman tingkat tinggi) dan low-level language (bahasa pemrograman tingkat rendah). Sebagai contoh, C++ mendukung konsep pemrograman berorientasi objek seperti kelas, objek, pewarisan, polimorfisme, dan enkapsulasi, yang memungkinkan pengembang untuk menulis kode yang lebih abstrak dan mendekati cara berpikir manusia, dibandingkan dengan memikirkan bagaimana mesin akan mengeksekusi instruksi. Selain itu, C++ juga menyediakan pustaka untuk struktur data yang lebih kompleks, seperti vector, list, map, dan lain-lain.

Di sisi lain, C++ juga menawarkan fitur-fitur yang dimiliki oleh bahasa pemrograman tingkat rendah, seperti kemampuan untuk mengakses serta mengontrol memori komputer secara langsung, dan mengakses register pada CPU, yang biasanya tidak tersedia pada bahasa pemrograman tingkat tinggi. Inilah yang menjadikan C++ sering digunakan dalam pengembangan perangkat lunak yang membutuhkan performa tinggi.

Versi pertama dari C++ dirilis pada tahun 1985, dan sejak itu bahasa ini terus berkembang serta mendapatkan fitur-fitur baru, seperti template, exception handling, dan Standard Template Library (STL). Pada tahun 1998, C++ distandarisasi oleh ISO (International Organization for Standardization), yang menjadikannya bahasa pemrograman yang diakui secara internasional.

Pengembangan lebih lanjut dari C++ telah menghasilkan beberapa versi standar, seperti C++11, C++14, C++17, dan C++20, masing-masing dengan tambahan fitur baru dan peningkatan kinerja. Saat ini, C++ tetap menjadi salah satu bahasa pemrograman yang paling populer dan digunakan secara luas di berbagai bidang teknologi.

Tipe Data

Tipe data pada C++ berfungsi untuk menentukan jenis nilai yang dapat disimpan, operasi apa yang dapat dilakukan, serta menentukan ukuran penyimpanan yang diperlukan terhadap data tersebut dalam sebuah program. Tipe Data sendiri terbagi kedalam 2 macam yaitu tipe data sederhana dan tipe data kompleks.

1. Tipe data sederhana (*primitive data types*)

Tipe data sederhana yang didukung secara bawaan C++ itu sendiri dan dapat digunakan secara langsung pada C++. Jenis tipe data yang dimaksud yaitu sebagai berikut:

Tipe Data	Ukuran	Jangkauan Nilai
bool	1 bits	1 atau 0 (True atau False)
char	1 bytes (8 bits)	-128 sampai 128
unsigned char	1 bytes	0 sampai 255
int	4 bytes (32 bits)	-2147483648 to 2147483647
unsigned int	4 bytes	0 sampai 4294967295
short int	2 bytes (16 bits)	-32768 sampai 32767
float	4 bytes (32 bits)	+/- 3.4e +/- 38 (~7 digits)
double	8 bytes	+/- 1.7e +/- 308 (~15 digits)
long double	8 bytes	+/- 1.7e +/- 308 (~15 digits)

unsigned pada tabel diatas berfungsi untuk memberitahu compiler bahwa kita hanya menggunakan area bilangan bulat positif saja, sehingga dengan ukuran yang sama data tersebut hanya memiliki jangkauan nilai pada rentang positif saja.

2. Tipe Data Kompleks

Berikut contoh dari tipe data kompleks

- a. array
yaitu tipe data yang merupakan kumpulan dari beberapa data dengan tipe data yang sama. untuk lebih lanjut akan dibahas pada bab [Array](#)
- b. struct
yaitu tipe data yang merupakan gabungan dari beberapa tipe data yang tergabung menjadi 1 tipe data. untuk lebih lanjutnya akan dibahas pada bab [Struct](#)
- c. pointer
yaitu tipe data yang berfungsi untuk menunjuk pengalamatan suatu data suatu variabel pada memori. untuk lebih lanjut akan dibahas pada bab [Operator](#)
- d. string
yaitu tipe data yang berfungsi untuk menyimpan kumpulan karakter (*char*) secara berurutan. Pada bahasa c++ tradisional *string* juga dapat direpresentasikan sebagai array terdiri dari kumpulan karakter.

Berikut adalah contoh penggunaan tipe data pada bahasa c++

```
#include <iostream>

int main() {
    // tipe data sederhana

    // deklarasi tipe data
    char karakter;
    int angka;
    float angka_desimal;
    char * kalimat;

    // meng-asign nilai ke dalam sebuah variabel
    karakter = 'c';
    angka = 69;
    angka_desimal = 3.14f;
    kalimat = "hello world";

    // mencetak nilai suatu variabel pada terminal
    std::cout << "karakter : " << karakter << std::endl;
    std::cout << "angka : " << angka << std::endl;
    std::cout << "angka_desimal : " << angka_desimal << std::endl;
    std::cout << "kalimat : " << kalimat << std::endl;
}
```

Selain tipe data bawaan di atas , kita juga dapat mendefinisikan tipe data kita sendiri dari tipe data yang sudah ada menggunakan keyword *typedef* seperti berikut.

```
#include <iostream>

int main() {
    // deklarasi tipe data custom
    typedef char karakter;
    typedef karakter * kalimat;

    // deklarasi tipe data
    karakter ini_karakter;
    kalimat ini_kalimat;

    // meng-asign nilai ke dalam sebuah variabel
    ini_karakter = 'c';
    ini_kalimat = "hello world";

    // mencetak nilai suatu variabel pada terminal
    std::cout << "ini karakter : " << ini_karakter << std::endl;
    std::cout << "ini kalimat : " << ini_kalimat << std::endl;
}
```

EMBEDDED &
CYBER-PHYSICAL
SYSTEM LABORATORY

Operator

Secara sederhana pada C++, **Operator merupakan simbol**. Simbol yang dimaksud adalah lambang-lambang yang digunakan untuk melakukan operasi matematika maupun manipulasi logika pada sebuah variabel maupun nilai. Seperti Contoh, lambang `" + "` merupakan sebuah operator yang digunakan untuk melakukan penjumlahan, sedangkan `" - "` adalah operator yang digunakan untuk melakukan pengurangan.

Misalnya, kita ingin menjumlahkan nilai dari variabel a dan b menggunakan operator `" + "` maka penjumlahan akan menjadi:

$$C = a + b$$

Dari bentuk persamaan diatas kita dapat melihat bahwa `" + "` merupakan operator sedangkan `'a'` dan `'b'` merupakan variabel atau nilai yang ingin dioperasikan (Operand). Dalam Operasi operand di C++ pada dasarnya terbagi menjadi 3, yaitu unary, binary, dan ternary. Unary ketika melibatkan 1 buah operand, binary melibatkan 2 buah operand, sedangkan ternary ketika melibatkan 3 buah operand. Untuk operator dalam C++ sendiri dapat diklasifikasikan menjadi 6 tipe. Berikut adalah penjelasan tiap tipe operator:

1. Operator Aritmatika

Operator Aritmatika merupakan operator yang digunakan untuk melakukan perhitungan matematika pada operand. Untuk simbol-simbol aritmatika pada C++ adalah sebagai berikut:

Operator	Fungsi Aritmatika	Unary/Binary	Contoh
+	Penjumlahan	Binary	$3 + 2 = 5$
-	Pengurangan	Binary	$5 - 2 = 3$
*	Perkalian	Binary	$3 * 4 = 12$
/	Pembagian	Binary	$4 / 2 = 2$
%	Operasi Modulo (Sisa pembagian)	Binary	$6 \% 2 = 4$
+	Nilai Positif	Unary	+205 (Nilai Positif)
-	Nilai Negatif	Unary	-205 (Nilai Negatif)
++	Increment Operator	Unary	c++
--	Decrement Operator	Unary	c--

Contoh penggunaan operator Aritmatika:

```
#include <iostream>
using namespace std;
int main(){

//Deklarasi Tipe Data//
bool boolean;
char character;
int integer;
float floating;

//Memberikan Nilai Data//
boolean = true;
character = '1';
integer = 11;
floating = 11.11;

//Menampilkan Data//
cout << "Tipe data bool : " << boolean << endl;
cout << "Tipe data char : " << character << endl;
cout << "Tipe data int : " << integer << endl;
cout << "Tipe data float : " << floating << endl;
}
```

2. Operator Pembandingan

Operator Pembandingan merupakan operator yang digunakan untuk membandingkan dua nilai. Konsep yang sama digunakan seperti pada matematika untuk menentukan suatu variabel memiliki nilai lebih besar atau lebih kecil dan sebagainya. Hasil dari operasi tersebut merupakan nilai (1) untuk nilai Benar, atau (0) untuk nilai salah. Nilai pembandingan terdiri dari:

Operator	Operasi
!=	Tidak Sama dengan
==	Sama Dengan
<	Kurang Dari
>	Lebih Dari
<=	Kurang Dari Sama Dengan
>=	Lebih Dari Sama Dengan

Contoh penggunaan operator Pembanding:

```
#include <iostream>
using namespace std;

int main(){
    int a, b;

    cout << "Nilai a : ";
    cin >> a;
    cout << "Nilai b : ";
    cin >> b;

    // menggunakan operator pembanding
    cout << "a > b = " << (a > b) << endl;
    cout << "a < b = " << (a < b) << endl;
    cout << "a >= b = " << (a >= b) << endl;
    cout << "a <= b = " << (a <= b) << endl;
    cout << "a == b = " << (a == b) << endl;
    cout << "a != b = " << (a != b) << endl;

    return 0;
}
```

3. Operator Penugasan

Operator penugasan merupakan operator yang digunakan untuk memberikan nilai pada variabel. Dimana pada operator penugasan sisi kiri merupakan variabel dan sisi kanannya adalah nilai. Nilai pada sisi kanan harus sesuai dengan tipe data variabel sisi kiri. Berikut merupakan operator penugasan yang digunakan pada pemrograman C++.

Operat or	Keterangan	Contoh
=	Untuk menugaskan nilai pada variabel	a = 10
+=	Gabungan dari '+' dan '='. Digunakan untuk menambahkan nilai dari variabel kiri ke nilai kanan lalu hasilnya ditugaskan pada variabel kiri	a += b sama dengan a = a + b

-=	Gabungan dari '-' dan '='. Digunakan untuk mengurangi nilai pada variabel kiri dengan variabel	a -= b sama dengan a = a - b
=	Gabungan dari '' dan '='. Digunakan untuk mengalikan nilai variabel kiri dengan nilai variabel kanan lalu hasilnya ditugaskan pada variabel kiri	a *= b sama dengan a = a * b
/=	Gabungan dari '/' dan '='. Digunakan untuk membagi nilai variabel kiri dengan nilai variabel kanan lalu hasilnya di tugas pada variabel k.	a /= b sama dengan a = a / b

Contoh sederhana penggunaan Operator Penugasan:

```

#include <iostream>
using namespace std;
int main(){
//Deklarasi variabel//
int a;

//Contoh-Contoh operator//

//1. operator =
a = 10;
cout << "Nilai a : " << a << endl;
//2. operator +=
cout << "Nilai a += 10 : " << (a += 10) << endl;
//3. operator -=
cout << "Nilai a -= 10 : " << (a -= 10) << endl;
//4. operator *=
cout << "Nilai a *= 10 : " << (a *= 10) << endl;
//5. operator /=
cout << "Nilai a /= 10 : " << (a /= 10) << endl;

system("pause");

```

4. Operator Logika

Operator logika digunakan untuk mengolah nilai atau ekspresi dengan tipe data boolean. Operator digunakan untuk memeriksa kesamaan nilai dari dua data atau lebih dan menghasilkan nilai true (1) atau false (0). Macam operator logika pada C++ adalah:

Operator	Operasi
&&	AND Logic
	OR Logic
!	NOT Logic

Berikut merupakan tabel kebenaran dari operator logika:

AND			OR			NOT	
Input		Output	Input		Output	Input	Output
A	B	Y	A	B	Y	X	Y
0	0	0	0	0	0	0	1
0	1	0	0	1	1	1	0
1	0	0	1	0	1	-	-
1	1	1	1	1	1	-	-

Contoh penggunaan Operator Logika:

```
#include <iostream>
using namespace std;
int main(){
//Deklarasi variabel//
bool A,B;

//Memasukkan variabel//
cout << "Nilai A : ";
cin >> A;
cout << "Nilai B : ";
cin >> B;

//mengeluarkan nilai//
cout << "A : " << A << endl;
cout << "B : " << B << endl << endl;
cout << "A & B : " << (A&&B) << endl;
cout << "A | B : " << (A||B) << endl;
cout << "!A : " << (!A) << endl;
cout << "!B : " << (!B) << endl;
system("pause");

return 0;
}
```

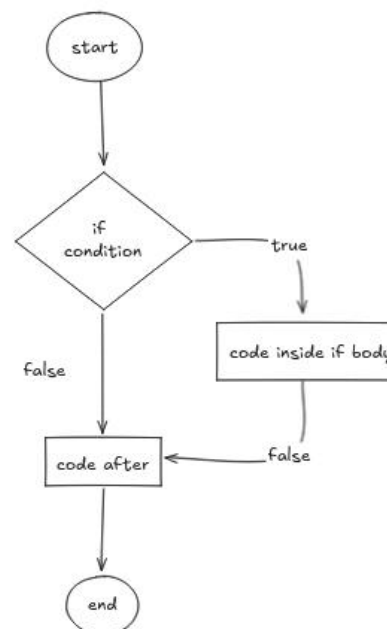
Pengkondisian dan Percabangan

Pengkondisian yaitu suatu cara untuk mengatur alur atau arah dari eksekusi program yang kita buat menggunakan ekspresi boolean dimana ketika benar akan bernilai 1 dan ketika salah akan bernilai 0.

Sedangkan percabangan yaitu mirip seperti pengkondisian hanya saja statement yang dilakukan pengecekan lebih dari 1.

Pengkondisian If

digunakan untuk mengambil keputusan statement tersebut akan dieksekusi atau tidak. Jika kondisi terpenuhi, maka statement tersebut akan dieksekusi, atau jika sebaliknya statement tidak akan dieksekusi. Berikut adalah alur dari if statement



Berikut contoh kode untuk penggunaan if

```

#include <iostream>

int main() {
    int a = 5;
    if (a == 5) {
        // ini akan dieksekusi ketika a bernilai 5
        std::cout << "nilai a adalah 5" << std::endl;
    }
}
  
```

Pengkondisian Else

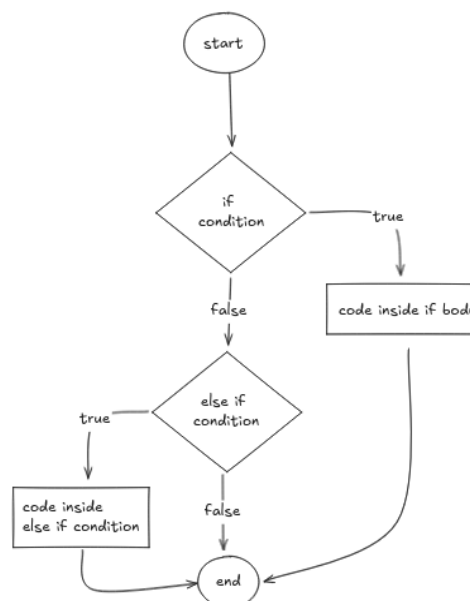
pengkondisian ini mirip dengan pengkondisian If sebelumnya, hanya saja kita dapat menentukan output apa yang dikeluarkan ketika statement pada pengkondisian tidak terpenuhi atau bernilai *false*. berikut adalah alur dari pengkondisian *Else*. Berikut adalah contoh kode program menggunakan pengkondisian *Else*

```
#include <iostream>

int main() {
    int a = 5;
    if (a == 5) {
        // ini akan dieksekusi ketika a bernilai 5
        std::cout << "nilai a adalah 5" << std::endl;
    } else {
        std::cout << "nilai a tidak sama dengan 5" << std::endl;
    }
}
```

Pengkondisian Else-If

pengkondisian ini mirip dengan pengkondisian if hanya saja statement pada blok *Else If* akan dieksekusi ketika statement pertama tidak terpenuhi dan statement kedua telah terpenuhi. Berikut adalah alur dari pengkondisian *Else-If*



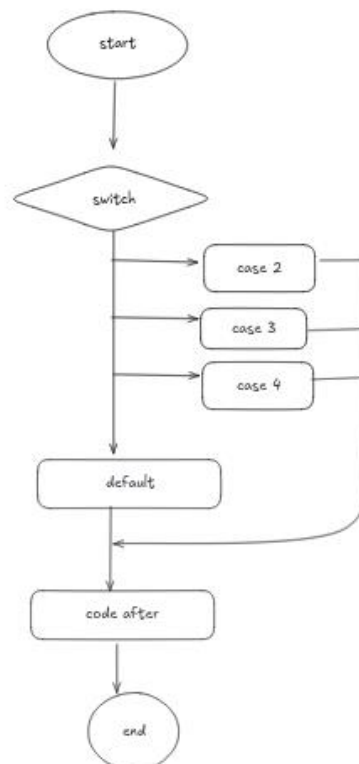
Berikut adalah contoh kode program untuk pengkondisian *Else If*

```
#include <iostream>

int main() {
    int a = 5;
    if (a < 5) {
        std::cout << "nilai a kurang dari 5" << std::endl;
    } else if (a % 2 == 0){
        std::cout << "nilai a bernilai genap namun lebih dari 5" << std::endl;
    }
}
```

Percabangan Switch

switch yaitu jenis percabangan untuk mengkondisikan lebih dari 1 macam state , berikut adalah alur dari percabangan switch.



Berikut adalah contoh kode program untuk percabangan switch

```
#include <iostream>

int main() {
    int a = 0;
    std::cout << "masukkan nilai a : ";
    std::cin >> a;

    switch (a) {
        case 2:
            std::cout << "nilai a bernilai 2" << std::endl;
        case 5:
            std::cout << "nilai a tidak bernilai 2 namun bernilai 5" << std::endl;
        default:
            std::cout << "nilai a tidak memenuhi semua kondisi percabangan yang ada" << std::endl;
    }

    return 0;
}
```



Array

Array merupakan kumpulan data sama yang tersimpan pada lokasi memori yang berdekatan dan elemennya dapat diakses secara acak menggunakan indeks dari array. Array dapat digunakan untuk menyimpan kumpulan tipe data primitif seperti int, float, double, char, dan lainnya. Selain itu, array juga dapat menyimpan tipe data turunan seperti structure, pointers, dan sebagainya. Berikut merupakan representasi dari array.

Indeks	0	1	2	3	4	5	6	7	8
Value	8	10	6	-2	11	7	1	100	15

Array memiliki indeks dan nilai data itu sendiri. Sedangkan jarak antar elemen pada array disesuaikan dengan lebar data untuk masing-masing tipe data array. Misalnya pada tipe data integer, maka jarak antar elemennya bernilai 2 s/d 4 byte. Indeks array dimulai dari indeks 0, dan seterusnya. Berikut merupakan beberapa contoh deklarasi array:

char karakter[20] :	Meminta 20 tempat di memori komputer dengan indeks dari 0-19, dimana semua elemennya memiliki tipe data karakter
int angka[5] :	Meminta 5 tempat di memori komputer dengan indeks dari 0-4 dimana elemennya berupa integer
int bil[5] :	Meminta 5 tempat di memori komputer dengan indeks 0-4 dimana semua elemennya bertipe data integer dengan indeks 0 bernilai 2, indeks 1 bernilai 0, indeks 2 bernilai 5, dan 2 sisa indeks kosong sehingga bernilai 0
char dinamis[] :	Mendeklarasikan array dengan ukuran maksimum array tidak diketahui namun ukuran tersebut diketahui berdasarkan inisialisasi yaitu sebanyak 4 elemen yang berisi 'e', '2', '0', dan '5'

Saat pemanggilan array, parameter di dalam tanda [] berupa indeks dari array yang ingin dipanggil. Misalnya "kondisi[0]" berarti indeks yang ke nol. Array yang sudah dipesan, misalnya 10 tempat tidak harus diisi semuanya, bisa saja hanya diisi 5 elemen saja, baik berurutan maupun tidak. Namun pada kondisi yang tidak penuh tersebut tempat pemesanan di memori tetap 10 tempat, jadi tempat yang tidak terisi tetap akan terpesan dalam memori dan

dibiarkan kosong. Ketika array dideklarasikan tetapi dikosongkan semua saat pendeklarasian, maka memori yang sudah dipesan akan memasukkan nilai yang sifatnya random.

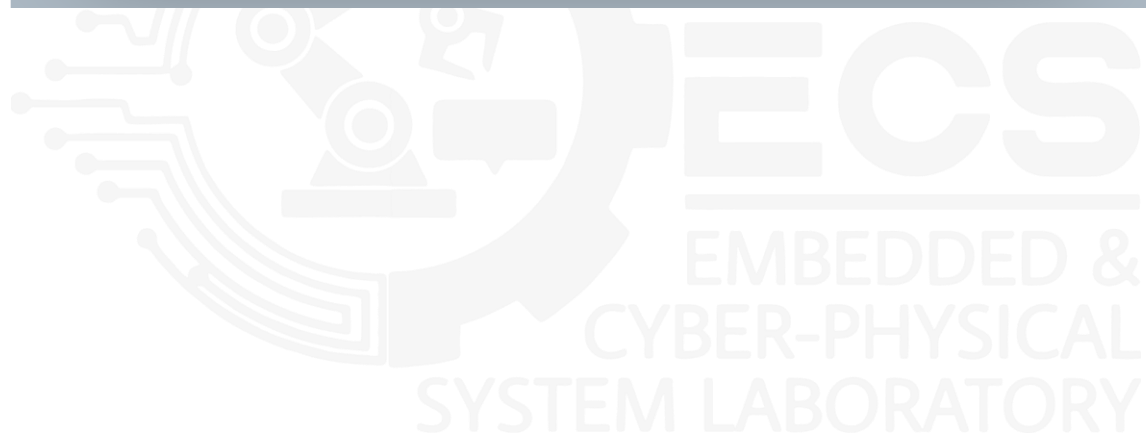
Contoh menggunakan Array:

```
#include <iostream>
using namespace std;
int main(){

    char huruf[3] = {'A','B','C'};
    int nilai [3] = {95,40,0};

    cout << "Nilai " << nilai[0] << " mendapat predikat " << huruf[0] << endl;
    cout << "Nilai " << nilai[1] << " mendapat predikat " << huruf[1] << endl;
    cout << "Nilai " << nilai[2] << " mendapat predikat " << huruf[2] << endl;

    system("pause");
    return 0;
}
```

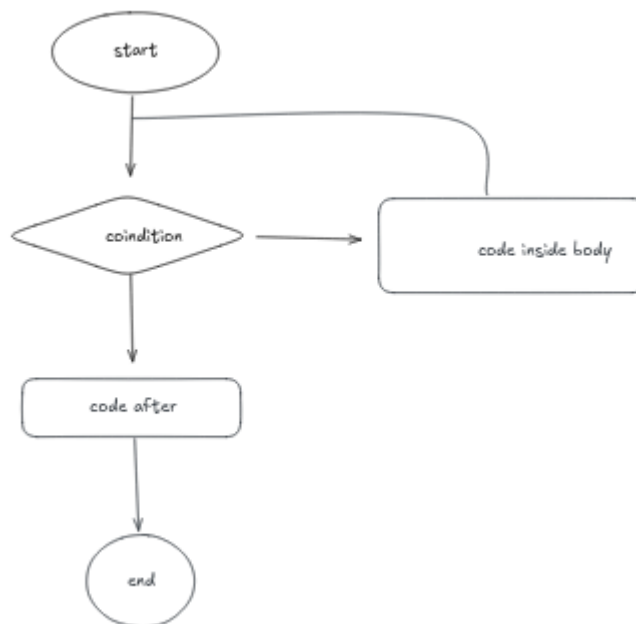


Perulangan

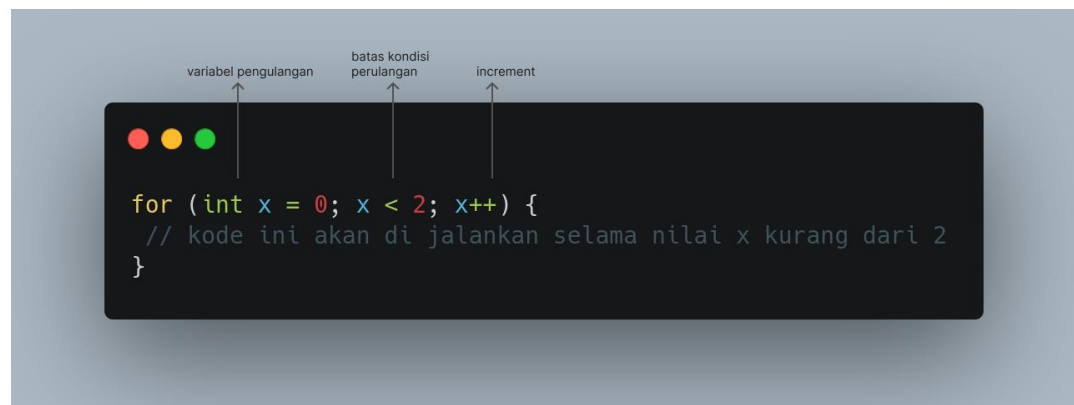
perulangan yaitu fitur pada bahasa pemrograman untuk melakukan suatu pengulangan sampai terpenuhi suatu nilai dan kerap digunakan untuk melakukan suatu iterasi pada suatu kumpulan data. pada c++ terdapat tiga macam jenis perulangan yaitu sebagai berikut.

1. For Loop

Jenis Perulangan ini digunakan untuk mengulangi perintah dengan jumlah pengulangan yang sudah diketahui. Pada statement for ini perlu dituliskan suatu kondisi untuk diuji berupa expresi boolean, nilai awal dan perintah yang dipakai untuk menghitung (counter). Berikut adalah alur dari for loop



Berikut cara penggunaan for loop pada C++



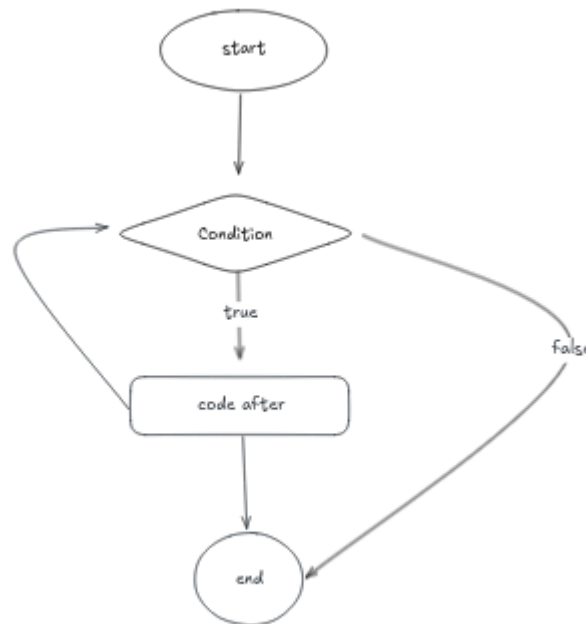
```

for (int x = 0; x < 2; x++) {
    // kode ini akan di jalankan selama nilai x kurang dari 2
}
  
```

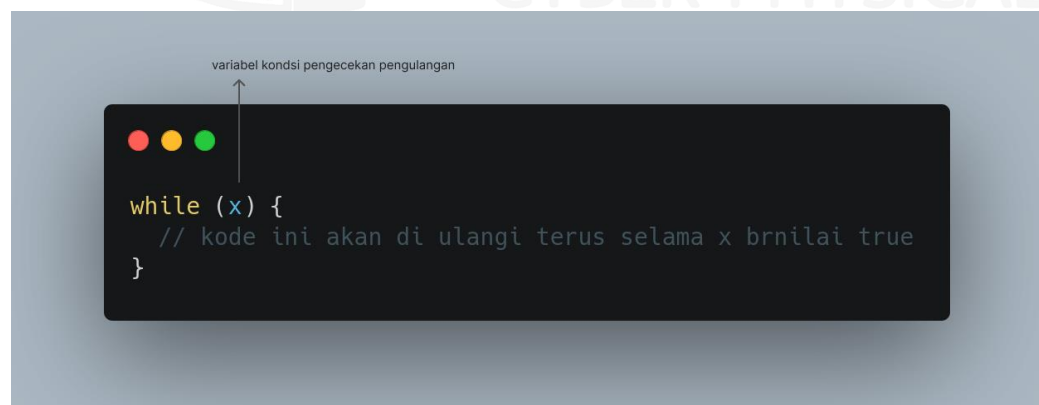
2. While Loop

Jenis Perulangan ini digunakan untuk mengulangi perintah dengan kondisi tertentu. pengulangan akan terus berjalan selama kondisi tersebut bernilai benar. While Loop digunakan jika pengguna belum mengetahui jumlah pengulangan yang dibutuhkan

Berikut adalah alur dari perulangan while loop



Berikut adalah cara penggunaan while loop pada c++



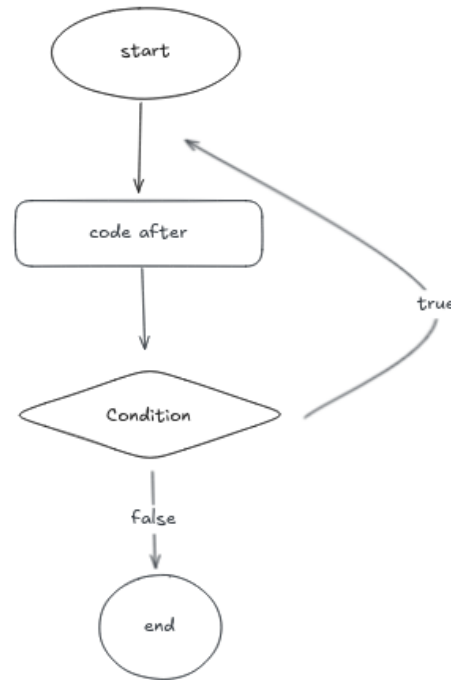
```

while (x) {
    // kode ini akan di ulangi terus selama x bernilai true
}
  
```

variabel kondisi pengecekan pengulangan

3. Do While Loop

Jenis Perulangan ini sama seperti while loop, hanya saja sebelum dilakukan pengecekan kondisi akan dijalankan perintah pada blok kode do terlebih dahulu.



Berikut adalah cara penggunaan do while loop pada c++

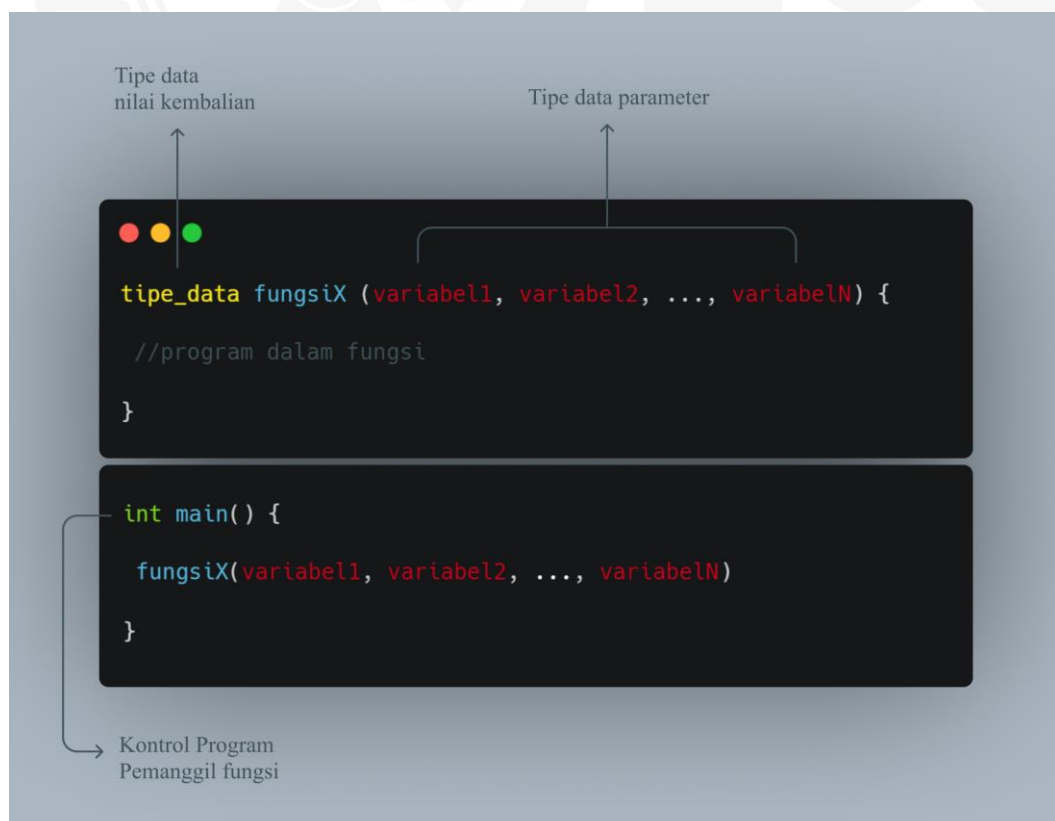


Fungsi

Fungsi (Function) adalah sekumpulan program yang diberi nama, sehingga dengan demikian jika program itu diperlukan dapat dipanggil kembali. Hal ini memudahkan agar tidak perlu menulis kembali program yang memiliki algoritma yang sama. Walaupun Pemrograman Berorientasi Objek telah menggeser perhatian dari fungsi ini, namun fungsi tetap saja merupakan bagian paling inti dalam suatu program. Fungsi global bisa berada di luar kelas maupun objek.

Fungsi dapat melakukan manipulasi terhadap data dan dapat mengembalikan suatu nilai. Semua program yang ditulis dengan bahasa C++ paling tidak mempunyai sebuah fungsi, yaitu *main()*, fungsi ini akan dipanggil pertama kali ketika program dieksekusi, sedangkan fungsi yang lain baru akan bekerja ketika fungsi tersebut dipanggil. Karena fungsi ini bukan merupakan bagian dari objek, maka fungsi ini dipanggil secara global, dapat diakses dari manapun dalam program yang ditulis. Setiap fungsi diberi nama, dan saat fungsi dipanggil dalam program utama, maka eksekusi program akan dialihkan ke tubuh (isi) fungsi tersebut. Setelah selesai, yaitu ditandai dengan statemen *return* atau tanda kurung kurawal tutup, maka akan kembali ke program utama melanjutkan ke baris program berikutnya. Peristiwa ini dinamakan pemanggilan fungsi.

Fungsi yang baik mengerjakan sebuah pekerjaan yang spesifik, mudah dipahami dan mudah dikenali berdasarkan nama fungsi tersebut. Pekerjaan yang kompleks lebih baik dipecah pecah menjadi beberapa fungsi yang nantinya dapat dipanggil ketika diperlukan. Fungsi terdiri dari 2 macam, yaitu fungsi yang dibuat sendiri (user-defined) dan fungsi standard (built-in). Fungsi standard merupakan bagian dari paket kompiler yang kita pakai yang sudah tersedia untuk digunakan, sedangkan fungsi yang dibuat sendiri adalah fungsi yang kita tulis sebelum dapat dipergunakan. Berikut adalah contoh bentuk dasar script fungsi yang mengembalikan nilai



1. Fungsi yang mengembalikan nilai

```
#include <iostream>
using namespace std;

// Menginisiasikan parameter Fungsi
double perhitungandaya (float voltage, float hambatan) {
    double daya;
    daya = (voltage*voltage)/hambatan;
    return daya;
}

// Kode program dari fungsi
int main() {
    float tegangan = 12;
    float resistansi = 4;
    double dayakeluar = perhitungandaya(tegangan,resistansi);
    cout << "Daya yang dihasilkan : " << dayakeluar << "W" << endl;

    system("pause");
    return 0;
}
```

2. Fungsi yang tidak mengembalikan nilai

```
#include <iostream>
using namespace std;

// Menginisiasikan parameter Fungsi
void perhitunganDaya (float voltage, float hambatan) {
    double daya;
    daya = (voltage*voltage)/hambatan;
    cout << "Daya yang dihasilkan : " << daya << "W" << endl;
}

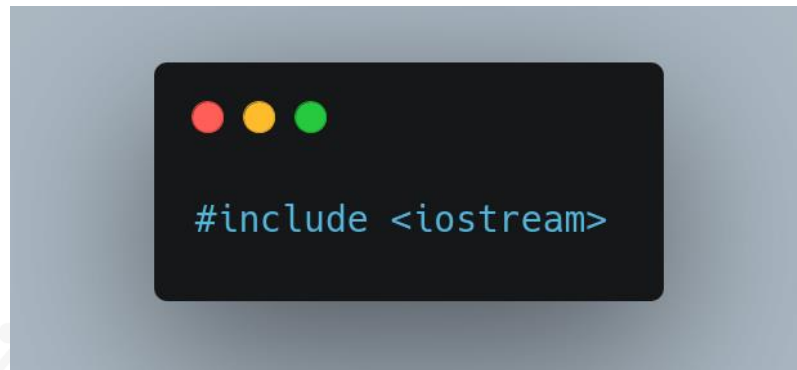
// Kode program dari fungsi
int main() {
    float tegangan = 12;
    float resistansi = 4;
    perhitunganDaya(tegangan,resistansi);

    system("pause");
    return 0;
}
```

Header

header merupakan fitur pada bahasa pemrograman C++ yang berfungsi untuk memanggil library yang sudah ada sehingga suatu fungsi dapat digunakan secara baik dan benar. untuk mendeklarasikan file header pada C++ digunakan keyword **#include**. Misalnya sebagai contoh penggunaan Iostream pada baris kode sebelumnya yang merupakan standard library untuk input output pada C++.

Berikut adalah contoh kode program menggunakan header.



Berikut adalah contoh library yang sering digunakan pada bahasa pemrograman C++.

1. #include <iostream>

iostream atau kependekan dari input output stream yang merupakan standar library yang berhubungan dengan input output, seperti baca tulis ke/dari terminal dan baca tulis ke/dari file. Berikut adalah contoh beberapa fungsi yang terdapat pada library iostream

1) std::cin

Fungsi cin pada C++ digunakan untuk membaca input dari pengguna melalui console. cin adalah singkatan dari "character input" Fungsi ini beroperasi dengan mengambil data input yang diberikan oleh pengguna dan menyimpannya ke dalam variabel.

2) std::cout

`std::cout` pada C++ adalah fungsi output standar yang digunakan untuk menampilkan data ke layar (console). Fungsi ini digunakan bersamaan dengan operator `<<`, yang disebut **insertion operator**

Untuk list standard library lainnya dapat dilihat pada dokumentasi resmi bahasa C++ pada link berikut: <https://cplusplus.com/reference/>

Struct

Struct merupakan fitur pada bahasa C++ yang berfungsi untuk menyimpan data yang terdiri dari kumpulan tipe data yang berbeda-beda. secara pendeklarasian , struct sangat berbeda dengan array yang hanya memiliki 1 buah tipe data untuk setiap kumpulanya.

Berikut adalah contoh kode program untuk membuat struct.



Berikut contoh kode program untuk menggunakan struct yang telah dibuat



Pointer

Pointer adalah suatu jenis variabel dalam bahasa pemrograman yang menyimpan pengalamatan memori dari variabel lain. Alih-alih menyimpan nilai data itu sendiri, pointer menyimpan lokasi di mana data tersebut disimpan pada memori. Dengan menggunakan pointer, program dapat mengakses dan memanipulasi data secara tidak langsung, yang memungkinkan efisiensi memori dan kontrol yang lebih fleksibel terhadap struktur data seperti array.

Untuk cara pendefinisian pointer yaitu sebagai berikut



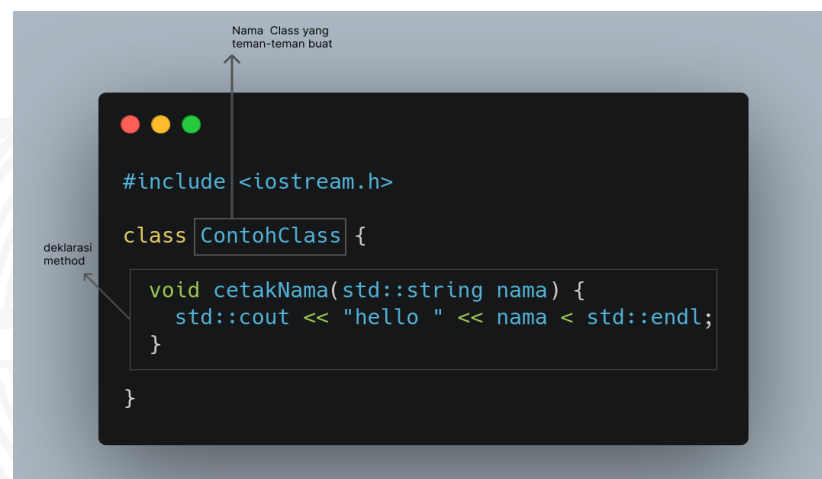
Maksud dari baris kode diatas yaitu variabel b mengarah ke alamat memori yang sama dengan variabel a, sehingga ketika **variabel b** diubah nilainya juga akan berpengaruh terhadap **variabel a**;

Pemrograman Berorientasi Objek

Object Oriented Programming atau biasa disingkat OOP yaitu suatu paradigma atau pendekatan dalam pemrograman yang berfokus pada penggunaan objek sebagai representasi data dalam program. Dalam OOP, data dan metode yang memanipulasi data dikelompokkan ke dalam satu entitas yang disebut kelas. OOP mendukung prinsip-prinsip dasar seperti enkapsulasi, pewarisan, polimorfisme, dan abstraksi, yang memungkinkan pengorganisasian kode yang lebih modular, dan reusable.

Class dan Method

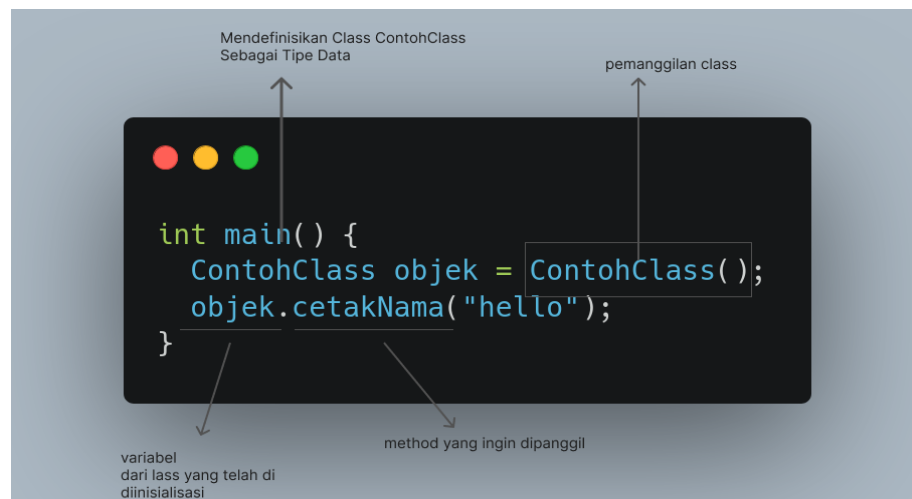
Class yaitu suatu tipe data yang dapat kita definisikan sendiri dan membungkus kumpulan fungsi dan variabel. Sedangkan metode yaitu suatu istilah yang merujuk pada fungsi yang terdapat di dalam class. Berikut adalah penulisan untuk mendefinisikan class pada bahasa pemrograman c++



Untuk membuat suatu instansi menggunakan class yang telah kita definisikan, terdapat 2 macam cara yaitu menggunakan class sebagai objek biasa dan sebagai pointer.

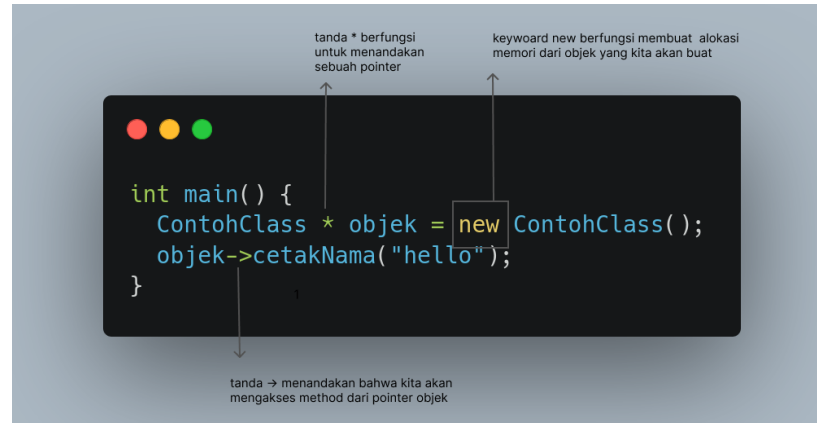
1. Definisi class sebagai objek biasa

dengan menginisialisasi class dengan cara seperti ini maka objek ini akan disimpan di memori **stack**.



2. Definisi class sebagai pointer

Dengan menginisialisasi class dengan cara seperti ini maka objek ini akan disimpan di memori **heap**.



Pewarisan Sifat

Pewarisan sifat, atau inheritance, adalah suatu teknik dalam Pemrograman Berorientasi Objek (OOP) untuk menurunkan sifat berupa fungsi dan variabel dengan akses level public. Sebagai contoh, apabila terdapat sebuah kelas Kendaraan yang memiliki metode gas() dan rem(), kemudian terdapat kelas untuk kendaraan spesifik jenis tertentu seperti **Tesla** dan **Jeep**, yang keduanya termasuk ke dalam jenis mobil, kita dapat mewarisi kelas Mobil pada kedua kelas tersebut. Dengan demikian, kedua kelas ini akan memiliki metode gas() dan rem() tanpa perlu mendefinisikan ulang kedua metode tersebut pada masing-masing kelas

Namun, perlu diingat hanya method dengan level akses protected dan public saja yang dapat diakses oleh class turunannya. Berikut adalah contoh untuk penulisan kodenya

```
#include <iostream>

class Mobil {
protected:

    void gas() {
        std::print("kendaraan digas");
    }

    void rem() {
        std::print("kendaraan direm");
    }
}
```

```
class Tesla : protected Mobil {  
public:  
  
    Tesla() {  
        gas();  
    }  
  
    ~Tesla() {  
        rem();  
    }  
}
```

```
class Tesla : protected Mobil {  
public:  
  
    Tesla() {  
        gas();  
    }  
  
    ~Tesla() {  
        rem();  
    }  
}
```

Enkapsulasi

Enkapsulasi merupakan salah satu konsep dasar dalam pemrograman berorientasi objek, dimana memiliki kegunaan untuk menjaga suatu proses program agar tidak dapat diakses secara sembarangan atau intervensi oleh program lain. Untuk dapat memenuhi kebutuhan tersebut maka variabel class perlu dideklarasikan peran penentu akses dalam Enkapsulasi menjadi beberapa jenis:

1. **Private**, merupakan penentu akses yang membatasi akses data dan fungsi hanya di dalam kelas. Data dan fungsi yang dideklarasikan private hanya dapat diakses oleh anggota fungsi dari kelas yang sama. Dengan tujuan agar tidak dapat di akses dari variabel class lainnya pada satu program yang sama.
2. **Protected**, penentu akses terproteksi mirip dengan private, tetapi perbedaannya adalah penentu akses proteksi mengizinkan kelas turunan untuk mengakses data-data dalam class tersebut. Untuk dapat mengakses data tersebut perlu menggunakan fungsi anggota **derived class**.
3. **Public**, variable class yang telah di private atau di protected masih dapat dikontrol, diakses dan dimodifikasi dalam program, dengan menggunakan metode **get** and **set** yang dideklarasikan pada variabel kelas public untuk dapat melakukan kontrol akses dan modifikasi pada class private.

```
class Employee {  
    private:  
        // Variabel Private  
        int salary;  
  
    public:  
        // Metode Set  
        void setSalary(int s) {  
            salary = s;  
        }  
        // Metode Get  
        int getSalary() {  
            return salary;  
        }  
};
```

Polimorfisme

Polimorfisme yaitu suatu fungsi yang memiliki lebih dari satu makna atau pendefinisian yang berbeda sehingga dalam satu fungsi yang sama dapat digunakan lebih dari satu pendefinisian parameter yang berbeda-beda.

Berikut adalah contoh dari polymorphisme pada C++

```
#include <iostream>

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

int jumlah(int a, int b, int c)
{
    return a+b+c
}

int main() {
    return 0;

    std::cout << "jumlah dengan 2 parameter : " << jumlah(1, 2) << std::end;
    std::cout << "jumlah dengan 3 parameter : " << jumlah(1, 2, 4) << std::end;
}
```

Namespace

Namespace dalam C++ adalah fitur yang membantu dalam mengatur kode dan menyediakan container untuk menghindari konflik penanaman, dan juga sebagai lokasi untuk menampung hal hal seperti fungsi, kelas, konstanta sebagai cara untuk mengelompokkan mereka secara logis. Bayangkan terdapat dua library, dimana kedua fungsi diberi nama contoh_fungsi(). Dimana tanpa namespace, dapat terjadi conflict. Seperti contoh kode di bawah, dapat menyebabkan error, maka Namespace dapat membantu dalam menghindari masalah tersebut

```
void contoh_fungsi() {  
    // kode untuk library pertama  
}  
  
void contoh_fungsi() {  
    // kode untuk library pertama  
}
```

Untuk memudahkan pemahaman, definisi namespace dapat menggunakan struktur seperti pada gambar dibawah, dimana untuk dapat mengakses variabel dalam namespace, digunakan scope resolution operator (::).

```
namespace myNamespace {  
    // code inside the namespace  
}
```

Struktur
namespace

```
myNamespace::functionName();
```

Scope resolution
operator (::)

Dengan definisi diatas, berikut adalah contoh dari penggunaan namespace:

```
#include <iostream>
using namespace std;

namespace ns1 {
    void greet() {
        cout << "Hello from ns1!" << endl;
    }
}

namespace ns2 {
    void greet() {
        cout << "Hello from ns2!" << endl;
    }
}

int main() {
    ns1::greet(); // Output: Hello dari ns1!
    ns2::greet(); // Output: Hello dari ns2!
    return 0;
}
```

Jika dalam proses membuat suatu program dengan menggunakan banyak namespace, untuk menulis namespaceName::functionName setiap kali ingin memanggil variabel namespace tersebut. Dengan mudah dapat menggunakan kata kunci “using”, dapat membantu membawa namespace kedalam skope, maka akan dengan mudah untuk mengawali semua dengan namespaceName. Berikut adalah contoh penggunaan:

```
#include <iostream>
using namespace std;

namespace ns1 {
    void greet() {
        cout << "Hello from ns1!" << endl;
    }
}

int main() {
    using namespace ns1; // Maka fungsi ns1's dapat dipakai tanpa ns1::
    greet(); // Output: Hello from ns1!
    return 0;
}
```

Namespace dapat kita nested atau class dideklarasikan dalam scope class lainnya. hal ini membantu dalam mengorganisir hasil kode yang diperlukan. Sebagai contoh, dibawah ini

```
#include <iostream>
using namespace std;

namespace outer {
    namespace inner {
        void greet() {
            cout << "Hello from inner namespace!" << endl;
        }
    }
}

int main() {
    outer::inner::greet(); // Output: Hello from inner namespace!
    return 0;
}
```

Ada metode lain juga yaitu, yaitu unnamed (anonymous) namespace adalah namespace tanpa nama. Kode dalam ruang nama ini hanya dapat diakses di dalam file yang dideklarasikan. Hal ini sangat berguna ketika ingin membatasi cakupan fungsi dan variabel pada satu file. Berikut adalah contoh pemakaian:

```
#include <iostream>
using namespace std;

namespace {
    void greet() {
        cout << "Hello from anonymous namespace!" << endl;
    }
}

int main() {
    greet(); // Output: Hello from anonymous namespace!
    return 0;
}
```

Notes:

Pada sebagian besar program C++, kita biasanya menulis menggunakan namespace std. namespace std ini ada di dalam file header iostream. Std adalah singkatan dari standard. File header <iostream> berisi banyak anggota yang umum digunakan seperti cout, cin, endl, dll. Jadi untuk menghindari penulisan std::cout atau std::endl berulang-ulang, kita biasanya menambahkan directive using, menggunakan namespace std dalam program kita.

Daripada menggunakan direktif using (using namespace std), kita bisa menggunakan deklarasi using (using std::cout) hanya untuk mengimpor anggota file iostream yang umum digunakan. Hal ini dapat membantu kita menghindari konflik penamaan karena kita dapat membuat fungsi atau kelas dalam program kita dengan nama yang sudah ada dalam namespace std.



Konstruktor dan Destruktor

konstruktor yaitu suatu fungsi yang pertama kali dijalankan ketika suatu class pertama kali diinisialisasi. sedangkan Destruktor yaitu kebalikan dari konstruktor yaitu akan dijalankan ketika objek keluar dari lingkungannya atau dihancurkan secara eksplisit menggunakan keyboard **delete**. Kita juga dapat menambahkan parameter yang akan diisi saat kita menginisialisasi class dengan menambahkan parameter pada constructor.

Berikut adalah kode program untuk pembuatan konstruktor pada c++.

inialisasi construct dengan membuat fungsi dengan nama yang sama dengan nama class tanpa perlu memberikan tipe data di depannya

```
#include <iostream>

class Mobil {
public:
    Mobil(const char * nama) {
        std::cout << "ini mobil " << nama << std::endl;
    }
}
```

Berikut adalah kode program untuk pembuatan destruktur pada c++.

```
#include <iostream>

class Mobil {
public:
    ~Mobil() {
        std::cout << "ini destruktur dari kelas mobil " << std::endl;
    }
}
```

Cara membuat destruktur mirip dengan pembuatan constructor, hanya saja dengan menambahkan tanda tilde (~) pada awalnya, dan destruktur tidak dapat menerima parameter seperti konstruktor.

Abstraksi

Abstraksi yaitu suatu teknik pada OOP untuk menyembunyikan detail implementasi dan hanya memberikan informasi mengenai method apa saja yang terdapat pada class tersebut, dengan tujuan agar dapat dibuat beberapa implementasi dari class tersebut dengan pendefinisian berbeda namun dengan nama method yang sama.

```
#include <iostream>
#include <string>
using namespace std;

// Class Abstraksi untuk "mobil"
class Car {
public:
    // Pure virtual function untuk memulai mobil
    virtual void start() = 0;

    // Pure virtual function untuk stop mobil
    virtual void stop() = 0;

    // Virtual destructor (Penting untuk abstract class)
    virtual ~Car() {}
};

// Derived class Sedan.
class Sedan : public Car {
public:
    void start() override {
        cout << "Sedan starting..." << endl;
    }

    void stop() override {
        cout << "Sedan stopping..." << endl;
    }
};

// Derived class SUV.
class SUV : public Car {
public:
    void start() override {
        cout << "SUV starting..." << endl;
    }

    void stop() override {
        cout << "SUV stopping..." << endl;
    }
};

int main() {
    // Gunakan pointer untuk base class Car
    Car* sedan = new Sedan();
    Car* suv = new SUV();

    sedan->start();
    sedan->stop();

    suv->start();
    suv->stop();

    return 0;
}
```

Overloading

overloading yaitu suatu teknik pada OOP untuk meng-override atau mengganti isi atau definisi dari method pada suatu kelas di kelas turunannya dengan menambahkan keyword **override** pada awal pendefinisian sebuah method atau sebelum tipe data.

Berikut adalah contoh penggunaan overloading pada C++.

```
#include <iostream>

class kelasA {
    void hello() {
        std::cout << "hallo ini di cetak pada kelas a" << std::endl;
    }
}

class kelasB : public kelasA {
    override void hello() {
        std::cout << "hallo ini di cetak pada kelas B" << std::endl;
    }
}

int main() {
    return 0;

    KelasA instansiA = KelasA();
    instansiA.hello();

    KelasB instansiB = KelasB();
    instansiB.hello();
}
```

Referensi

<https://en.cppreference.com/w/>

<https://cplusplus.com/doc/>

