Struktur Program & Operasi pada Bahasa C++

Asprak TI UNESA ‘24

August 07, 2025

Struktur Program & Operasi pada Bahasa C++

Tujuan

Peserta praktikum, diharapkan dapat memahami struktur penulisan program C++, memahami konsep tipe data bilangan bulat (integer), bilangan real (float), beserta tipe data karakter (char) dan melakukan suatu perhitungan matematika sederhana menggunakan bahasa pemrograman C++.

Kajian Teori

Struktur Bahasa Pemrograman C++

Perhatikan kode berikut ini.

```
#include <iostream>
using namespace std;
int main(){
  cout << “Pucuk ubi pucuk kangkung, banyak bunyi pecah muncung” << endl;
  return 0;
}
```

Penjelasan :

- Baris 1 : `#include <iostream>` , Pada baris pertama terdapat “`#include`” yang merupakan instruksi kepada *compiler* untuk memasukkan *file*/pustaka (*library*) yang akan digunakan pada program. File ini sendiri merupakan sebuah file *pre-written* yang memiliki beberapa fungsi dan kelas yang sudah ditulis untuk menjalankan tugas tertentu. [Lihat lebih detail]
- Baris 2 : `using namespace std;` Merupakan sebuah pernyataan yang dimana ketika kita ingin menggunakan “cin” ataupun “cout” kita tidak perlu menambahkan “std” seperti “std::cout << “Pagi” << endl;”. [Lihat lebih detail]
- Baris 3 : `int main()` diikuti dengan kurung kurawal “{}”. Merupakan fungsi utama dan tempat dimana program mulai dijalankan. di dalam kurung kurawal `int main()` terdapat beberapa *syntax*/instruksi yang akan dijalankan. [Lihat lebih detail]
- Baris 4 : `cout` (diucapkan "see-out") adalah perintah yang digunakan bersama dengan operator penyisipan (<<) untuk mengeluarkan/mencetak teks. Dalam contoh diatas, kita akan menghasilkan "Pucuk ubi pucuk kangkung, banyak bunyi pecah muncung".
- Baris 5 : `return 0;` mengakhiri fungsi utama.
- Perlu diketahui pada setiap menulis instruksi pada program c++ perlu diakhiri dengan tanda *semicolon* “;”

Keyword & Identifier

Perhatikan kode dibawah ini :

```
#include <iostream>
using namespace std;
int main(){
  int jumlah;
  string namaBarang;
  float harga;
  bool isSold;
}
```

Keyword

Keyword merupakan kata yang sudah ditentukan sebelumnya oleh suatu bahasa pemrograman, Kata ini mempunyai makna khusus bagi *compiler* dan tidak bisa didefinisikan ulang. Pada program diatas, yang termasuk keyword adalah: `using namespace`, `int`, `main`, `string`, `float`, dan `bool`.

Identifier

Identifier adalah nama yang diberikan ke fungsi, variabel, dan entitas lainnya. Pada program di atas contoh *Identifier* adalah `jumlah`, `namaBarang`, `harga`, dan `isSold`. Ada beberapa aturan untuk penamaan *identifier* yaitu:

1. Karakter tidak boleh mengandung simbol selain *underscore* (_).
2. Tidak memiliki nama yang sama dengan salah satu *keyword* C++, namun boleh digunakan apabila digabung dengan kata lain, seperti (intA1, bool1).
3. Tidak menggunakan operator aritmatika (* + - /)
4. Tidak menggunakan spasi.
5. *Identifier* bersifat *case-sensitive* artinya (A) akan dibedakan dengan (a).
6. *Identifier* tidak boleh dimulai dengan angka, harus dengan huruf ((a-z) atau (A-Z)) atau *underscore* (_).

Variabel

Variabel adalah suatu tempat yang digunakan untuk menampung data atau nilai pada memori yang mempunyai nilai yang dapat berubah–ubah selama proses program. Dalam bahasa C++, variabel menyimpan data/nilai dengan tipe data tertentu. Seperti halnya variabel yang menyimpan suatu angka yang termasuk bilangan bulat (integer).

Deklarasi Variabel

Pada bahasa C++, variabel harus dideklarasikan terlebih dahulu sebelum bisa digunakan.

Untuk mendeklarasikan variabel dapat dilakukan dengan *syntax* berikut ini.

```
//format
tipe_data identifier;

//Contoh
int jumlah;
float harga;
string nama;
```

Pada contoh di atas, `int`, `float`, dan `string` merupakan tipe data. sedangkan `jumlah`, `harga`, dan `nama` merupakan identifier.

Jika kita ingin mendeklarasikan variabel dengan tipe yang sama, maka kita bisa menggunakan tanda koma (,).

```
//format
<tipe_data> <variabel1>, <variabel2>,... dst;

//Contoh
int x,y,z;
```

Pengisian Variabel

Setelah variabel dideklarasikan, maka variabel dapat diisi dengan sebuah nilai sesuai dengan tipe datanya.

```
//format
<variabel1> = <value>;

//Contoh
jumlah = 10;
harga = 2.5;
nama = “Kak Gem”;
```

Pengisian variabel juga dapat dilakukan ketika variabel dideklarasikan.

```
//format
<tipe_data> <identifier> = <value>;

//Contoh
int jumlah = 10;
float harga = 2.5;
string nama = “Kak Gem”;
```

Konstanta dan Literal

Literal merupakan definisi dari nilai itu sendiri, contohnya adalah 2,5 dan “Kak Gem” merupakan literal. Sedangkan konstanta merupakan variabel konstan yang mewakili literal yang dimana variabel ini tidak mungkin nilainya berubah.

Variabel konstan dibuat dengan menambahkan keyword *const* saat pendefinisian variabel.

Perlu diperhatikan bahwa definisi variabel konstan harus disertai inisialisasinya.

```
//format
const <tipe_data> <nama_var> = <value>;

//Contoh
3,14 ← ini adalah literal
const float = 3.14; ← Variabel yang bersifat konstan dan memiliki literal didalamnya
```

Segala bentuk perubahan yang dilakukan terhadap variabel konstan akan menghasilkan error.

```
//Contoh
const float = 3.14;
float = 2; //Error
```

Selain menggunakan keyword *const*, pendefinisian variabel dapat menggunakan #define. Penggunaan #define diterapkan luar fungsi utama [int main()]

```
//format
#define <nama> <value>

//Contoh

#include <bits\stdc++.h>
using namespace std;
#define konstFloat 3,14

int main(){
    float a = konstFloat;
}
```

Konvensi Penamaan C++ (Naming Convention)

Dalam C++, konvensi penamaan sering menggunakan prefiks atau sufiks tertentu, membedakan antara huruf besar dan huruf kecil untuk variabel, memulai nama kelas dengan huruf kapital dan banyak lagi, Tujuan dari konvensi ini adalah untuk menjaga konsistensi kode, mempermudah *maintenance* kode, serta meningkatkan keterbacaan, sehingga *programmer* lain dapat dengan mudah memahami dan menavigasi kode tersebut. Berikut adalah beberapa konvensi umum yang sering digunakan.

1. Nama variabel harus menggambarkan/menjelaskan apa yang disimpan variabel tersebut

```
string nama = "Pak Vincent"; //mewakili nama
int jumlahOrang = 2;  //mewakili jumlah orang
bool is_alive = true; //mewakili status hidup seseorang
```

2. Nama variabel boolean biasanya ditulis dengan awalan 'is'

```
bool isSigma = true;
bool is_alive = false;
bool IsEmpty = false;
```

3. Nama variabel menggunakan salah satu dari beberapa konvensi yang sudah ditentukan

```
// Camel Case
int totalItemsInCart = 10;

// Snake Case
int total_items_in_cart = 20;
```

```
// Pascal Case
int TotalItemsInCart = 30;
```

4. variabel konstan ditulis menggunakan uppercase dan underscore (_)

```
const float PI = 3.14;
const int SIZE_LIMIT = 2000;
```

Tipe Data

Tipe data adalah kategori yang mendefinisikan jenis nilai yang bisa disimpan dalam variabel dan operasi apa yang akan dilakukan oleh variabel tersebut. secara sederhana tipe data dibagi menjadi 2 yaitu tipe data primitif dan tipe data kompleks

Tipe Data Primitif

1. Bilangan bulat

   Variabel dengan tipe data ini digunakan untuk menyimpan bilangan positif hingga negatif tanpa koma, contoh tipenya adalah `int`, `long`, `unsigned int`

```
int kotak = 5;
long saldoTabungan = 10000000;
unsigned int jumlahBarang = 10;
```

2. Bilangan real

   Variabel dengan tipe data ini digunakan untuk menyimpan bilangan yang mempunyai koma, tipe yang biasa digunakan adalah `float`, `double`

```
float IPK = 3.8;
double Phi = 3.14159265;

// double memiliki rentang nilai(range), penggunaan memori, dan presisi
lebih besar daripada float
```

3. Karakter

   Variabel dengan tipe data ini digunakan untuk menyimpan satu huruf atau karakter. tipenya adalah `char`

```
char golonganDarah = A;
```

4. Boolean

   Variabel boolean digunakan untuk menyimpan value true(benar) / false(salah), tipenya adalah `bool`

```
bool IsCodingFun = true;
bool IsCheatingAllowed = false;
```

Tipe Data Kompleks

Tipe data kompleks adalah tipe data yang dibangun dari tipe data primitif untuk menyimpan informasi yang lebih kompleks

1. Array

   Sekumpulan elemen dengan tipe data yang sama, disimpan bersebelahan pada memori dan diakses dengan menggunakan index

```
int[4] = {9, 11, 69, 420);
```

2. Pointer

   Variabel yang menyimpan alamat dari variabel lain, memungkinkan data untuk dimanipulasi langsung di memori, pointer biasa dilambangkan dengan (*)

```
int a = 69;
int *aPtr = &a;
```

3. Struktur

   Tipe data yang dapat menyimpan beberapa nilai dari berbagai tipe data dalam satu variabel

```
//deklarasi struct
struct Student{
    string name;
    int age;
};

//inisiasi dan penggunaan
Student student1;
student1.name = “Onikata Kayoko”;
student1.age = 18;
```

Operasi Input & Output Dasar

Dalam membuat program di bahasa C++ terkadang kita ingin membuat sebuah program yang mana data atau variabelnya bisa sesuai dengan kemauan penggunanya sehingga oleh karena itu diperlukan operasi input dalam program, dan untuk mengeluarkan hasilnya pula diperlukan operasi output. Untuk menggunakan operasi ini kita perlu memasukkan library *iostream* ke dalam program.

Input

Pada operasi input ini kita akan membuat sebuah program yang meminta input/masukan dari penggunanya. Agar bisa berfungsi seperti yang dijelaskan sebelumnya kita memerlukan library dari iostream agar bisa berfungsi dan kemudian baru bisa meminta input dari user. Dalam bahasa C++ perintah yang digunakan untuk input yaitu cin. Perlu diperhatikan bahwa bits/stdc++.h merupakan library dasar dan penting agar cin dapat digunakan untuk meminta input.

```
#include <bits\stdc++.h>
using namespace std;

int main(){
   int x;
   cin >> x;
}
```

Output

Operasi output dipergunakan di dalam sebuah program untuk memberikan keluaran (output) sesuai dari program yang dibuat. Sama seperti input, agar *output* ini bisa berjalan maka diperlukan *library* iostream yang merupakan *library* dasar. Perlu diperhatikan bahwa endl merupakan manipulator spesial yang digunakan dalam cout agar baris outputnya hanya berhenti pada baris tersebut sehingga output selanjutnya akan terbentuk di baris setelahnya.

```
//contoh 1
#include <iostream>
using namespace std;

int main(){
   string nama = "Pak Vincent";
   cout << nama << endl;
   return 0;
}
```

```
//contoh 2
#include <iostream>
using namespace std;

int main(){
   string nama;
   cin >> nama;
   cout << "Nama saya adalah " << nama << endl;
   return 0;
}
```

Operator

Operator dapat diartikan sebagai sebuah simbol yang digunakan untuk melakukan operasi tertentu. Misalnya jika kita ingin menjumlahkan variabel a dan b, maka kita akan menggunakan operator penjumlahan (+).

Operator Assignment

Operator *Assignment* sesuai namanya yaitu untuk *assign* (memberi) sebuah *value* atau nilai ke dalam sebuah variabel. Simbol yang biasa digunakan adalah tanda sama dengan (=). Contohnya:

```
int x = 5; // Nilai 5 diberikan ke variabel x
float modus = 23,55; // Nilai 23,55 diberikan ke variabel modus
char tidak = G; //Sebuah character G diberikan ke variabel tidak
```

Operator Aritmatika

Operator Aritmatika adalah jenis operator yang digunakan untuk melakukan operasi matematika pada angka yang telah dimasukkan ke sebuah variabel sebelumnya. Beberapa operator menggunakan simbol yang sama pada matematika (penjumlahan dengan simbol '+', pengurangan dengan '-', dst.). Contoh:

```
int a = 10;
int b = 3;
int jumlah = a + b // Variabel jumlah berisi dari penjumlahan variabel a
dan b yang sebelumnya telah diisi
int kali = a * b // Variabel kali berisi dari perkalian variabel a dan b
yang sebelumnya telah diisi
int bagi = a / b // Variabel bagi berisi dari pembagian variabel a dan b
```

```
yang sebelumnya telah disi
```

Operator - operator aritmatika pada bahasa C++ adalah sebagai berikut.

Simbol	Operasi	Contoh
+	Penjumlahan	a + b
-	Pengurangan	a - b
*	Perkalian	a * b
/	Pembagian	a / b
%	Sisa Bagi (Modulo)	a % b

Operator Increment dan Decrement

Operator *increment* dan *decrement* digunakan untuk menambahkan dan mengurangkan nilai suatu variabel. Berikut contoh penggunaannya

```
int a = 69;
a++;
cout << a; //70

int b = 420;
b--;
cout << b; //419
```

Operator ini mempunyai dua versi yaitu prefix dan postfix.

Prefix increment/decrement

Nilai variabel akan diubah terlebih dahulu baru digunakan ke sebuah ekspresi

```
int a = 10;
int b = ++a; //pada contoh ini a akan diubah menjadi 11 baru b menjadi 11
```

Postfix increment/decrement

Nilai variabel akan digunakan dalam sebuah ekspresi, baru nilainya diubah

```
int a = 10;
int b = a++; //pada contoh ini b akan menjadi a (10), baru a ditambah 1
```

```
menjadi 11
```

Bisa diingat bahwa perbedaan antara postfix dan prefix hanya akan terjadi apabila digunakan dalam sebuah ekspresi

```
int a = 90;
a++;
```

```
int a = 90;
++a;
```

Kedua cuplikan kode diatas akan menghasilkan yang sama yaitu a = 91.

Operator Relasional

Operator yang digunakan untuk membandingkan dua variabel, output dari perbandingan ini nantinya akan berbentuk boolean(true/false). contoh:

```
int a = 69;
int b = 420;


bool result1 = a == b; //karena a tidak sama dengan b maka operasi ini akan menghasilkan output false;

bool result2 = a < b; //karena a kurang dari b maka operasi ini akan menghasilkan output true
```

Berikut ini operator relasional pada C++.

Operator	Simbol	Keterangan	Contoh
Sama dengan	==	Digunakan untuk memeriksa apakah kedua variabel memiliki nilai yang sama.	3 == 2 (FALSE) 3 == 3 (TRUE)
Tidak sama dengan	!=	Digunakan untuk memeriksa apakah kedua variabel memiliki nilai yang	3 != 2 (TRUE) 3 != 3 (FALSE)

		tidak sama.	
Lebih besar	>	Digunakan untuk membandingkan apakah variabel pertama lebih besar nilainya dari operan kedua.	3 > 1 (TRUE) 1 > 5 (FALSE) 2 > 4 (FALSE)
Lebih kecil	<	Digunakan untuk membandingkan apakah variabel pertama lebih kecil nilainya dari operan kedua.	7 < 5 (FALSE) 4 < 2 (FALSE) 1 < 6 (TRUE)
Lebih besar sama dengan	>=	Digunakan untuk membandingkan apakah variabel pertama lebih besar atau sama nilainya dari operan kedua.	7 >= 3 (TRUE) 6 >= 6 (TRUE) 2 >= 5 (FALSE)
Lebih kecil sama dengan	<=	Digunakan untuk membandingkan apakah variabel pertama lebih kecil atau sama nilainya dari operan kedua.	3 <= 1 (FALSE) 6 <= 6 (TRUE) 3 <= 6 (TRUE)

Operator Logika

Operator ini digunakan untuk melakukan operasi logika, yaitu operasi yang melibatkan boolean. Operasi ini biasa digunakan pada pernyataan ‘if’, dan ‘while’ (akan dibahas dimateri selanjutnya)

```
bool a = true;
bool b = false;


//Operasi OR
//Operasi OR akan menghasilkan output ‘true’ jika salah satu atau kedua
operand( disini a dan b) memiliki value ‘true’.

bool orResult = a || b; // menghasilkan true


//operasi AND
//operasi AND akan menghasilkan output ‘true’ jika kedua operand memiliki
value ‘true’.

bool andResult = a && b; // menghasilkan false


//operasi NOT
//menghasilkan kebalikan dari nilai operand, jika operand mempunyai nilai
‘true’ maka menghasilkan ‘false’ dan sebaliknya.
```

```
bool notResult1 = !a; // karena a awalnya ‘true’, operasi ini akan
menghasilkan ‘false’.

bool notResult2 = !(andResult); // karena resultAnd awalnya ‘false’,
operasi ini akan menghasilkan ‘true’.
```

Operator Bitwise

Operator bitwise digunakan untuk memanipulasi data dalam level bit. Saat melakukan operasi ini, bilangan bulat dianggap sebagai urutan digit biner.

```
int a = 5; //0101 dalam biner
int b = 6; //0110 dalam biner


//bitwise AND operator
//Membandingkan masing-masing bit dari dua operand. Hasilnya adalah ‘1’
jika kedua bit tersebut ‘1’, Dan ‘0’ jika tidak.

int andResult = a & b; //hasilnya adalah 4
/*
0101 (a)
0110 (b)
----
0100 (andResult)
*/


//bitwise OR operator
//Membandingkan masing-masing bit dari dua operand. Hasilnya adalah ‘1’
jika salah satu atau kedua bit tersebut ‘1’.

int orResult = a | b; //hasilnya adalah 7
/*
0101 (a)
0110 (b)
----
0111 (orResult)
*/

//bitwise XOR operator
//Membandingkan masing-masing bit dari dua operand. Hasilnya adalah ‘1’
jika salah satu dari kedua bit tersebut ‘1’, namun tidak keduanya

int xorResult = a ^ b; //hasilnya adalah 3
/*
0101 (a)
```

```
0110 (b)
----
0011 (xorResult)
*/


//bitwise NOT operator
//Membalikkan semua bit operand. Bit ‘1’ akan menjadi ‘0’ dan sebaliknya.

int notResult= ~a; //hasilnya adalah -6
/*
dalam 8-bit integer
bit 5 mempunyai biner
0000 0101

oleh karena itu saat dibalik menjadi
1111 1010
*/


//Bitwise left shift operator
//Menggeser bit ke kiri sebanyak ‘1’ posisi.

int leftShiftResult = a << 1; // hasilnya adalah 10
//int leftShift = a << n; //menggeser bit ke kiri sebanyak ‘n’ posisi
/*
0101 (a)
----
1010 (c)
*/


//Bitwise right shift operator
//Menggeser bit ke kanan sebanyak ‘1’ posisi.

int rightShiftResult = a >> 1; // hasilnya adalah 2
//int rightShift = a >> n; //menggeser bit ke kanan sebanyak ‘n’ posisi
/*
0101 (a)
----
0010 (c)
*/
```

Operator Gabungan

Operator gabungan berisi gabungan dari beberapa operator lain dalam satu baris kode, perhatikan program dibawah

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

int main(){
    int num = 10;
    if(num > 0 && num < 15){
        cout <<  “Kondisi Terpenuhi”;
    }
}
```

Dalam kode diatas bisa dilihat bahwa num = 10; dan pada statement if kondisi yang di check adalah apabila num lebih dari 0 dan num kurang dari 15. Dalam program diatas terdapat gabungan dua operator yaitu **Operator Relasional** dan **Operator Logika**, operator relasional adalah ((num > 0) (num < 15)) sementara operator logika adalah (&&) yang menjadi penghubung antara dua operator relasional.

Rangkuman Variable

```
int myNum = 5;              // Integer (whole number without decimals)
double myFloatNum = 5.99;   // Floating point number (with decimals)
char myLetter = 'D';        // Character
string myText = "Hello";    // String (text)
bool myBoolean = true;      // Boolean (true or false)
```

Rangkuman Operator

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
&=	x &= 3	x = x & 3