1. Sejarah C++

Tahun 1978, Brian W. Kerninghan & Dennis M. Ritchie dari AT & T Laboratories mengembangkan bahasa B menjadi bahasa C. Bahasa B yang diciptakan oleh Ken Thompson sebenarnya merupakan pengembangan dari bahasa BCPL (Basic Combined Programming Language) yang diciptakan oleh Martin Richard.

Sejak tahun 1980, bahasa C banyak digunakan pemrogram di Eropa yang sebelumnya menggunakan bahasa B dan BCPL. Dalam perkembangannya, bahasa C menjadi bahasa paling populer diantara bahasa lainnya, seperti PASCAL, BASIC, FORTRAN.

Tahun 1989, dunia pemrograman C mengalami peristiwa penting dengan dikeluarkannya standar bahasa C oleh *American National Standards Institute* (ANSI). Bahasa C yang diciptakan Kerninghan & Ritchie kemudian dikenal dengan nama ANSI C.

Mulai awal tahun 1980, Bjarne Stroustrup dari AT & T Bell Laboratories mulai mengembangkan bahasa C. Pada tahun 1985, lahirlah secara resmi bahasa baru hasil pengembangan C yang dikenal dengan nama C++. Sebenarnya bahasa C++ mengalami dua tahap evolusi. C++ yang pertama, dirilis oleh AT&T Laboratories, dinamakan **cfront**. C++ versi kuno ini hanya berupa kompiler yang menterjemahkan C++ menjadi bahasa C.

Pada evolusi selanjutnya, Borland International Inc. mengembangkan kompiler C++ menjadi sebuah kompiler yang mampu mengubah C++ langsung menjadi bahasa mesin (assembly). Sejak evolusi ini, mulai tahun 1990 C++ menjadi bahasa berorientasi obyek yang digunakan oleh sebagian besar pemrogram professional.

2. Struktur Bahasa C++

Berikut adalah sebuah contoh listing program dalam Bahasa C++:

Contoh 2.1 :

```
// my first program in C++
#include <iostream>
int main ()
{
  std::cout << "Hello World!";
}
```

Hasil :

```
Hello World!
```

Sisi kiri merupakan *source code*, yang dapat diberi nama *hiworld*.cpp dan sisi kanan adalah hasilnya setelah di-kompile dan di-eksekusi.

Program diatas merupakan salah satu program paling sederhana dalam C++, tetapi dalam program tersebut mengandung komponen dasar yang selalu ada pada setiap pemrograman C++. Jika dilihat satu persatu :

// my first program in C++

Baris ini adalah komentar. semua baris yang diawali dengan dua garis miring (**//**) akan dianggap sebagai komentar dan tidak akan berpengaruh terhadap program. Dapat digunakan oleh programmer untuk menyertakan penjelasan singkat atau observasi yang terkait dengan program tersebut.

#include <iostream>

Kalimat yang diawali dengan tanda (#) adalah directives read dan diinterpreterkan oleh *preprocessor*. Bukan merupakan baris kode yang dieksekusi, tetapi indikasi untuk kompiler. Dalam kasus ini kalimat **#include <iostream>** memberitahukan preprocessor kompiler untuk menyertakan header file standard **iostream**. File spesifik ini juga termasuk library deklarasi standard I/O pada C++ dan file ini disertakan karena fungsi-fungsinya akan digunakan nanti dalam program.

int main()

Baris ini mencocokan pada awal dari deklarasi fungsi **main**. fungsi **main** merupakan titik awal dimana seluruh program C++ akan mulai dieksekusi. Diletakan diawal, ditengah atau diakhir program, isi dari fungsi main akan selalu dieksekusi pertama kali. Pada dasarnya, seluruh program C++ memiliki fungsi **main**.

main diikuti oleh sepasang tanda kurung **()** karena merupakan fungsi. pada C++, semua fungsi diikuti oleh sepasang tanda kurung **()** dimana, dapat berisi argumen didalamnya. Isi dari fungsi **main** selanjutnya akan mengikuti,berupa deklarasi formal dan dituliskan diantara kurung kurawal (**{}**), seperti dalam contoh.

std::cout << "Hello World";

Baris ini adalah C++ *statement*. Baris statement terdiri dari tiga bagian. Pertama, std::cout, yang mengindentifikasi ***st**andard **c**haracter **out**put* device. Kedua, operator masukan (<<), yang mengindikasi kalimat sesudahnya dimasukkan ke std::cout. Bagian

terakhir, kalimat dalam tanda petik ("Hello World!"), adalah kalimat yang dimasukkan ke *standard output*.

Perhatikan setiap kalimat diakhiri dengan tanda semicolon (;). Karakter ini menandakan akhir dari instruksi dan harus disertakan pada setiap akhir instruksi pada program C++ manapun.

Tidak semua baris pada program ini melakukan aksi. Ada baris yang hanya berisi komentar (diawali //), baris yang berisi instruksi untuk preprocessor kompiler (Yang diawali **#**),kemudian baris yang merupakan inisialisasi sebuah fungsi (dalam kasus ini, fungsi **main**) dan baris yang berisi instruksi (seperti, **cout <<**), baris yang terakhir ini disertakan dalam blok yang dibatasi oleh kurung kurawal (**{}**) dari fungsi **main**.

Struktur program dapat dituliskan dalam bentuk yang lain agar lebih mudah dibaca, contoh :

```
int main ()
{
 std::cout << " Hello World ";
}
```

dapat juga dituliskan :

```
int main () { std::cout << " Hello World "; }
```

Dalam satu baris dan memiliki arti yang sama dengan program-program sebelumnya. pada C++ pembatas antar instruksi ditandai dengan semicolon (;) pada setiap akhir instruksi.

Contoh 2.2 :

```
// my second program in C++
#include <iostream>

int main ()
{
  std::cout << "Hello World! ";
  std::cout << "I'm a C++ program";
}
```

Hasil :

Hello World! I'm a C++ program

Dalam contoh 2, program memberikan dua masukan ke dalam std::cout dalam dua statement yang berbeda.

Komentar

Komentar adalah bagian dari program yang diabaikan oleh kompiler. Tidak melaksanakan aksi apapun. Mereka berguna untuk memungkinkan para programmer untuk memasukan catatan atau deskripsi tambahan mengenai program tersebut. C++ memiliki dua cara untuk menuliskan komentar :

```
//      Komentar baris
/*      Komentar Blok  */
```

Komentar baris, akan mengabaikan apapun mulai dari tanda (//) sampai akhir dari baris yang sama. Komentar Blok, akan mengabaikan apapun yang berada diantara tanda /* dan */.

Menggunakan *namespace std*

Sebagian besar kode program C++ hanya menggunakan cout, tanpa std. Penggunaan dari kedua bentuk tersebut memberikan hasil yang sama. Penggunaan std::cout langsung memasukkan cout ke dalam namespace std. cout adalah bagian dari *standard library*, dan semua elemen pada *standard library* C++ dinyatakan dalam apa yang disebut *namespace*: the namespace std.

Untuk merujuk pada elemen dalam std namespace, dapat dengan menggunakan:

```
Using namespace std:
```

Contoh 2.3 :

```
// my second program in C++
#include <iostream>
Using namespace std;

int main ()
{
  cout << "Hello World! ";
  cout << "I'm a C++ program";
}
```

Hasil :

Hello World! I'm a C++ program

3. Variabel, tipe data, konstanta

Untuk dapat menulis program yang dapat membantu menjalankan tugas-tugas kita, kita harus mengenal konsep dari **variabel**. Sebagai ilustrasi, ingat 2 buah angka, angka pertama adalah 5 dan angka kedua adalah 2. Selanjutnya tambahkan 1 pada angka pertama kemudian hasilnya dikurangi angka kedua (dimana hasil akhirnya adalah 4).

Seluruh proses ini dapat diekspresikan dalam C++ dengan serangkaian instruksi sbb :

```
a = 5;
b = 2;
a = a + 1;
result = a - b;
```

Jelas ini merupakan satu contoh yang sangat sederhana karena kita hanya menggunakan 2 nilai integer yang kecil, tetapi komputer dapat menyimpan jutaan angka dalam waktu yang bersamaan dan dapat melakukan operasi matematika yang rumit.

Karena itu, kita dapat mendefinisikan variable sebagai bagian dari memory untuk menyimpan nilai yang telah ditentukan. Setiap variable memerlukan **identifier** yang dapat membedakannya dari variable yang lain, sebagai contoh dari kode diatas identifier variabelnya adalah **a**, **b** dan **result**, tetapi kita dapat membuat nama untuk variabel selama masih merupakan identifier yang benar.

3.1. Identifiers

Identifier adalah untaian satu atau lebih huruf, angka, atau garis bawah (_). Panjang dari identifier, tidak terbatas, walaupun untuk beberapa kompiler hanya 32 karakter pertama saja yang dibaca sebagai identifier (sisanya diabaikan). Identifier harus selalu diawali dengan huruf atau garis bawah (_).

Ketentuan lainnya yang harus diperhatikan dalam menentukan identifier adalah tidak boleh menggunakan **key word** dari bahasa C++. Diawah ini adalah **key** word dalam C++ :

alignas	**Alignof**	**and**	**and_eq**	**asm**
auto	**Bitand**	**bool**	**break**	**case**
catch	**Char**	**class**	**const**	**const_cast**
continue	**Default**	**delete**	**do**	**double**
dynamic_cast	**Else**	**enum**	**explicit**	**extern**
false	**Float**	**for**	**friend**	**goto**
if	**Inline**	**int**	**long**	**mutable**
namespace	**New**	**operator**	**private**	**protected**
public	**Register**	**reinterpret_cast**	**return**	**short**
signed	**Sizeof**	**static**	**static_cast**	**struct**
switch	**template**	**this**	**throw**	**true**
try	**Typedef**	**typeid**	**typename**	**union**
unsigned	**Using**	**virtual**	**void**	**volatile**
wchar_t	**While**	**xor**	**xor_eq**	

Sebagai tambahan, represetasi alternatif dari operator, tidak dapat digunakan sebagai identifier. Contoh :

```
and, and_eq, bitand, bitor, compl, not, not_eq, or, or_eq,
xor, xor_eq
```

catatan: Bahasa C++ adalah bahasa yang "case sensitive", ini berarti identifier yang dituliskan dengan huruf kapital akan dianggap berbeda dengan identifier yang sama tetapi dituliskan dengan huruf kecil, sabagai contoh : variabel **RESULT** tidak sama dengan variable **result** ataupun variabel **Result**.

3.2. Tipe Data

Nilai-nilai variable disimpan di suatu tempat di lokasi yang tidak ditentukan dalam memori komputer direpresentasikan sebagai nol dan satu. Dalam program tidak perlu mengetahui lokasi yang tepat di mana variabel disimpan; cukup dengan memanggil nama variable.
Tipe data yang ada pada C++, berikut nilai kisaran yang dapat direpresentasikan :

DATA TYPES

Name	Bytes*	Description	Range*
char	1	character or integer 8 bits length.	**signed:** -128 to 127 **unsigned:** 0 to 255
short	2	integer 16 bits length.	**signed:** -32768 to 32767 **unsigned:** 0 to 65535
long	4	integer 32 bits length.	**signed:**-2147483648 to 2147483647 **unsigned:** 0 to 4294967295
int	*	Integer. Its length traditionally depends on the length of the system's Word **type**, thus in MSDOS it is 16 bits long, whereas in 32 bit systems (like Windows 9x/2000/NT and systems that work under protected mode in x86 systems) it is 32 bits long (4 bytes).	See **short**, **long**
float	4	floating point number.	3.4e + / - 38 (7 digits)
double	8	double precision floating point number.	1.7e + / - 308 (15 digits)
long double	10	long double precision floating point number.	1.2e + / - 4932 (19 digits)
bool	1	Boolean value. It can take one of two values: true or false NOTE: this is a type recently added by the ANSI-C++ standard. Not all	**true** or **false**

Name	Bytes*	Description	Range*
		compilers support it. Consult section bool type for compatibility information.	
wchar_t	2	Wide character. It is designed as a type to store international characters of a two-byte character set. NOTE: this is a type recently added by the ANSI-C++ standard. Not all compilers support it.	wide characters

3.3. Deklarasi variabel

Untuk menggunakan variabel pada C++, kita harus mendeklarasikan tipe data yang akan digunakan. Sintaks penulisan deklarasi variabel adalah dengan menuliskan tipe data yang akan digunakan diikuti dengan identifier yang benar, contoh :

```
int a;
float mynumber;
```

Jika akan menggunakan tipe data yang sama untuk beberapa identifier maka dapat dituliskan dengan menggunakan tanda koma, contoh :

```
int a, b, c;
```

Contoh 3.1 :

```
// operating with variables

#include <iostream>
using namespace std;

int main ()
{
  // declaring variables:
  int a, b;
  int result;

  // process:
  a = 5;
  b = 2;
  a = a + 1;
  result = a - b;

  // print out the result:
  cout << result;

  // terminate the program:
  return 0;
}
```

Hasil:

```
4
```

3.4. Inisialisasi Variabel

Pada pemrograman C++ ada tiga cara untuk menginisialisasi variable. Cara pertama adalah C-like initialization (karena cara ini diturunkan dari bahasa C). Terdiri dari tanda "=" diikuti oleh nilai yang diinisialisasi oleh variable. Penulisannya sbb:

```
type identifier = initial_value ;
```

Misalkan kita akan mendeklarasikan variabel **int** dengan nama **a** yang bernilai **0**, maka dapat dituliskan :

```
int a = 0;
```

Cara yang kedua, dikenal dengan constructor initialization (digunakan oleh bahasa C++), dengan cara menyertakan nilai yang akan diberikan dalam tanda **()**:

```
type identifier (initial_value) ;
```

Contoh :

```
int a (0);
```

Cara yang ketiga, dikenal dengan uniform initialization, mirip dengan cara kedua, tetapi menggunakan kurung kurawal { }. Cara ini mulai digunakan pada C++ standar 2011.

```
type identifier {initial_value"} ;
```

Contoh:

```
int a {0};
```

Contoh 3.2:

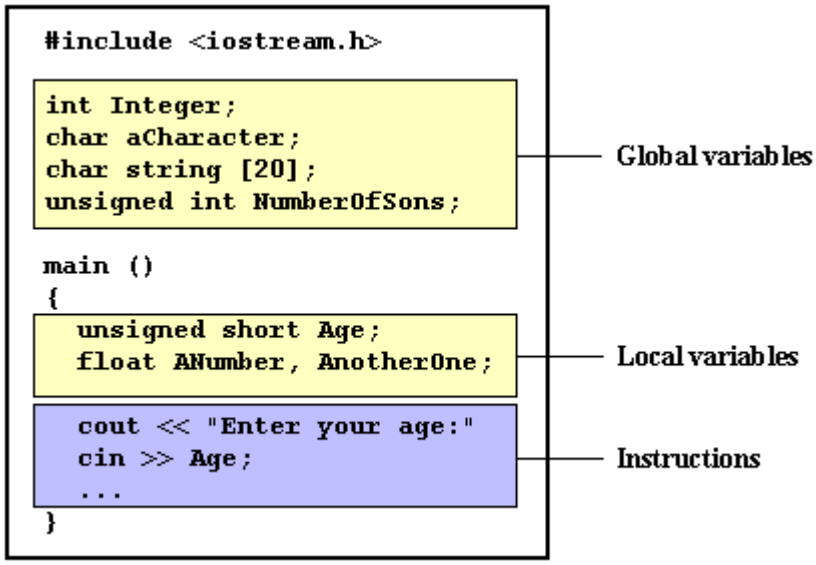
```
// initialization of variables

#include <iostream>
using namespace std;

int main ()
{
  int a=5;               // initial value: 5
  int b(3);              // initial value: 3
  int c{2};              // initial value: 2
  int result;            // initial value
undetermined

  a = a + b;
  result = a - c;
  cout << result;

  return 0;
}
```

Hasil:

```
6
```

Lingkup Variabel

Pada C++, kita dapat mendeklarasikan variable dibagian mana saja dari program, bahkan diantara 2 kalimat perintah.



variabel Global dapat digunakan untuk setiap bagian dari program, maupun fungsi, walaupun dideklarasikan diakhir program.

Lingkup dari **variable local** terbatas. Hanya berlaku dimana variable tersebut dideklarasikan. Jika dideklarasikan diawal fungsi (seperti dalam **main**) maka lingkup dari variable tersebut adalah untuk seluruh fungsi **main**. Seperti contoh diatas, jika terdapat fungsi lain yang ditambahkan pada main(), maka variable local yang dideklarasikan dalam **main** tidak dapat digunakan pada fungsi lainnya dan sebaliknya.

Pada C++, lingkup variable local ditandai dengan blok dimana variable tersebut dideklarasikan (blok tersebut adalah sekumpulan instruksi dalam kurung kurawal {}). Jika dideklarasikan dalam fungsi tersebut, maka akan berlaku sebagai variable dalam fungsi tersebut, jika dideklarasikan dalam sebuah perulangan, maka hanya berlaku dalam perulangan tersebut, dan seterusnya.

3.5. Pengenalan String

Untuk menggunakan tipe data string, program perlu disertakan header <string>

Contoh 3.3:

```
// my first string
#include <iostream>
#include <string>
using namespace std;

int main ()
{
  string mystring;
  mystring = "This is a string";
  cout << mystring;
  return 0;
}
```

Hasil:

```
This is a string
```

Seperti terlihat dari contoh 3.3, string dapat diinisialisasi dengan string literal apapun yang valid, seperti halnya tipe numerik. Variable dapat diinisialisasi menggunakan tiga tipe inisialisasi yang dimiliki oleh tipe numerik:

```
string mystring = "This is a string";
string mystring ("This is a string");
string mystring {"This is a string"};
```

String juga dapat melakukan operasi dasar yang dapat dilakukan oleh tipe data dasar, seperti dinyatakan tanpa nilai awal atau mengubah nilai pada saat eksekusi dilakukan.

Contoh 3.4:

```
// my first string
#include <iostream>
#include <string>
using namespace std;

int main ()
{
  string mystring;
  mystring = "This is the initial string content";
  cout << mystring << endl;
  mystring = "This is a different string content";
  cout << mystring << endl;
  return 0;
}
```

```
This is the initial string
content
This is a different string
content
```

Catatan: perintah endl adalah manipulator ends the line (mencetak ke baris baru).

3.6. Konstanta : Literals.

Konstanta adalah ekspresi dengan nilai yang tetap. Literal adalah jenis yang jelas dari konstanta. Mereka digunakan untuk mengekspresikan nilai-nilai tertentu dalam kode sumber dari program. Misalnya:

```
a = 5;
```

Nilai "5" pada potongan kode di atas adalah literal konstan.

Literal konstan terbagi dalam Nilai Integer, Nilai Floating-Point, Karakter, String, Boolean, pointers, dan user-defined literal.

3.7. Nilai Integer

Merupakan nilai konstanta numerik yang meng-identifikasikan nilai integer decimal. Karena merupakan nilai numeric, maka tidak memerlukan tanda kutip (") maupun karakter khusus lainnya. Contoh :

```
1776
707
-273
```

C++ memungkinkan kita untuk mempergunakan nilai oktal (base 8) dan heksadesimal (base 16). Jika menggunakan octal maka harus diawali dengan karakter **0** (karakter nol), dan untuk heksadesimal diawali dengan karakter **0x** (nol, x). Contoh :

```
75          // decimal
0113        // octal
0x4b        // hexadecimal
```

Dari contoh diatas, seluruhnya merepresentasikan nilai yang sama : 75.

3.8. Nilai Floating Point

Merepresentasikan nilai desimal dan/atau eksponen, termasuk titik desimal dan karakter **e** (Yang merepresentasikan "dikali 10 pangkat n" , dimana n merupakan nilai integer) atau keduanya. Contoh :

```
3.14159     // 3.14159
6.02e23     // 6.02 x 10^23
1.6e-19     // 1.6 x 10^-19
3.0         // 3.0
```

3.9. Karakter dan String

Karakter dan string menggunakan tanda kutip :

```
'z'
'p'
"Hello world"
"How do you do?"
```

Untuk karakter tunggal dituliskan diantara kutip tunggal (') dan untuk untaian beberapa karakter, dituliskan diantara kutip ganda (").

Konstanta karakter dan string memiliki beberapa hal khusus, seperti **escape codes**.

\n	Newline
\r	carriage return
\t	Tabulation
\v	vertical tabulation
\b	Backspace
\f	page feed
\a	alert (beep)
\'	single quotes (')
\"	double quotes (")
\?	question (?)
****	inverted slash (\)

Contoh :

```
'\n'
'\t'
"Left \t Right"
"one\ntwo\nthree"
```

Sebagai tambahan, kita dapat menuliskan karakter apapun dengan menuliskan yang diikuti dengan kode ASCII, mengekspresikan sebagai octal (contoh, **\23** atau **\40**) maupun heksadesimal (contoh, **\x20** atau **\x4A**).

3.10 Typed constant expression

Terkadang, kita perlu untuk memberikan nama ke nilai konstan. Contoh:

```
const double pi = 3.1415926;
const char tab = '\t';
```

Dengan prefix **const** kita dapat mendeklarasikan konstanta dengan tipe yang spesifik seperti yang kita inginkan. contoh :

```
const int width = 100;
const char tab = '\t';
const zip = 12440;
```

Jika tipe data tidak disebutkan, maka kompiler akan meng-asumsikan sebagai **int**.

Kita dapat menggunakan nama yang telah diberikan tersebut untuk memanggil nilai konstan.

Contoh 3.5:

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

const double pi = 3.14159;
const char newline = '\n';

int main ()
{
  double r=5.0;                // radius
  double circle;

  circle = 2 * pi * r;
  cout << circle;
  cout << newline;
}
```

Hasil:

```
31.4159
```

3.11. Preprocessor definitions (*#define*)

Kita dapat mendefinisikan sendiri nama untuk konstanta yang akan kita pergunakan, dengan menggunakan preprocessor directive **#define**. Dengan format :

```
#define identifier replacement
```

Dengan menggunakan #define, setiap pemanggilan *identifier* pada program direpresentasikan sebagai *replacement* yang bernilai sampai dengan akhr baris. Perubahan ini dilakukan oleh preprocessor, dan dilakukan sebelum program di compile.

Contoh 3.6:

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

#define PI 3.14159
#define NEWLINE '\n'

int main ()
{
  double r=5.0;                // radius
  double circle;

  circle = 2 * PI * r;
  cout << circle;
  cout << NEWLINE;

}
```

Hasil:

```
31.4159
```

Pada dasarnya, yang dilakukan oleh kompiler ketika membaca **#define** adalah menggantikan literal yang ada (dalam contoh, **PI** dan **NEWLINE**) dengan nilai yang telah ditetapkan (**3.14159265** dan **'\n'**). #define bukan merupakan instruksi, oleh sebab itu tidak diakhiri dengan tanda semicolon (;).

3.12. Operator

Operator-operator yang disediakan C++ berupa *keyword* atau karakter khusus. Operator-operator ini cukup penting untuk diketahui karena merupakan salah satu dasar bahasa C++.

Assignation (=).

Operator *assignation* digunakan untuk memberikan nilai ke suatu variable.

```
a = 5;
```

Memberikan nilai integer **5** ke variabel **a**. Sisi kiri dari operator disebut */value* (left value) dan sisi kanan disebut *rvalue* (right value). */value* harus selalu berupa variabeldan sisi kanan dapat berupa konstanta, variabel, hasil dari suatu operasi atau kombinasi dari semuanya.

Contoh 3.7.:

```
// assignment operator
#include <iostream>
using namespace std;

int main ()
{
  int a, b;         // a:?,  b:?
```

Hasil:

```
a:4 b:7
```

```
  a = 10;            // a:10, b:?
  b = 4;             // a:10, b:4
  a = b;             // a:4,  b:4
  b = 7;             // a:4,  b:7

  cout << "a:";
  cout << a;
  cout << " b:";
  cout << b;
}
```

Contoh :

```
a = 2 + (b = 5);
```

equivalen dengan :

```
b = 5;
a = 2 + b;
```

Hasil akhir adalah a bernilai 7.

Ekspresi berikut juga valid di C++:

```
a = b = c = 5;
```

ekspresi di atas memberikan nilai 5 ke 3 variabel a, b dan c.

*Arithmetic operators (+, -, *, /, %)*

+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulo

Operasi penambahan, pengurangan, perkalian dan pembagian menggunakan symbol yang umum seperti symbol operator biasa. Operasi modulo menggunakan symbol persentase (%), menampilkan sisa hasil bagi dari pembagian. Contoh:

```
a = 11 % 3;
```

hasil akhir adalah variable a bernilai 2.

Compound assignation operators
(+=, -=, *=, /=, %=, >>=, <<=, &=, ^=, |=)

Operator penugasan gabungan mengubah nilai variable saat ini dengan melakukan operasi terhadapnya. Mereka setara dengan menugaskan hasil operasi untuk operan pertama:

expression	equivalent to...
y += x;	y = y + x;
x -= 5;	x = x - 5;
x /= y;	x = x / y;
price *= units + 1;	price = price * (units+1);

Contoh 3.8:

```
// compound assignment operators
#include <iostream>
using namespace std;

int main ()
{
  int a, b=3;
  a = b;
  a+=2;              // equivalent to a=a+2
  cout << a;
}
```

Hasil:

```
5
```

Increment (++) and decrement (--).

Beberapa ekspresi dapat disederhanakan seperti ekspresi untuk menambah dan mengurangkan.
Contoh :

```
a++;
a+=1;
a=a+1;
```

Contoh diatas adalah equivalen secara fungsional. Nilai a ditambah 1.

Operator Increase dan Decrease dapat digunakan sebagai *prefix* atau *suffix*. Dengan kata lain dapat dituliskan sebelum identifier variabel (**++a**) atau sesudahnya (**a++**). operator increase yang digunakan sebagai *prefix* (**++a**), Perbedaannya terlihat pada tabel dibawah ini :

Example 1

```
B=3;
A=++B;
// A is 4, B is 4
```

Example 2

```
B=3;
A=B++;
// A is 3, B is 4
```

Pada contoh 1, **B** ditambahkan sebelum nilainya diberikan ke **A**. Sedangkan contoh 2, Nilai **B** diberikan terlebih dahulu ke **A** dan **B** ditambahkan kemudian.

Contoh 3.9:

```
// compound assignment operators
#include <iostream>
using namespace std;

int main ()
{
  int a, b=3;

  a = b++;
  cout << a << endl;
  cout << b << endl;

  a = ++b;
  cout << a << endl;
  cout << b << endl;
}
```

Hasil:

```
3
4
5
5
```

Relational operators (==, !=, >, <, >=, <=)

Untuk mengevaluasi antara 2 ekspresi, dapat digunakan operator Relasional. Hasil dari operator ini adalah nilai **bool** yaitu hanya berupa **true** atau **false**, atau dapat juga dalam nilai `int`, 0 untuk mereprensentasikan **"false"** dan 1 untuk merepresentasikan **"true"**. Operator-operator relasional pada C++ :

operator	Description
==	Equal to
!=	Not equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to

Contoh :

(7 == 5) would return **false.**
(5 > 4) would return **true.**
(3 != 2) would return **true.**
(6 >= 6) would return **true.**
(5 < 5) would return **false.**

Contoh, misalkan **a=2**, **b=3** dan **c=6** :

(a == 5) would return **false.**

(a*b >= c) would return **true** since (2*3 >= 6) is it.

(b+4 > a*c) would return **false** since (3+4 > 2*6) is it.

((b=2) == a) would return **true**.

Logic operators (!, &&, ||).

Operator ! equivalen dengan operasi boolean NOT, hanya mempunyai 1 operand, berguna untuk membalikkan nilai dari operand yang bersangkutan. Contoh :

!(5 == 5) returns **false** because the expression at its right (5 == 5) would be **true**.

!(6 <= 4) returns **true** because (6 <= 4) would be **false**.

!true returns **false**.

!false returns **true**.

operator Logika **&&** dan **||** digunakan untuk mengevaluasi 2 ekspresi dan menghasilkan 1 nilai akhir. mempunyai arti yang sama dengan operator logika Boolean *AND* dan *OR*. Contoh :

First Operand **a**	Second Operand **b**	result **a && b**	result **a || b**
true	true	**true**	**true**
true	false	**false**	**true**
false	true	**false**	**true**
false	false	**false**	**false**

Contoh :

((5 == 5) && (3 > 6)) returns **false** (*true* && *false*).

((5 == 5) || (3 > 6)) returns **true** (*true* || *false*).

Conditional operator (?).

operator kondisional mengevaluasi ekspresi dan memberikan hasil tergantung dari hasil evaluasi (*true* atau *false*). Sintaks :

```
condition ? result1 : result2
```

Jika kondisi **true** maka akan menghasilkan *result1*, jika tidak akan menghasilkan *result2*.

7==5 ? 4 : 3 returns 3 since 7 is not equal to 5.

7==5+2 ? 4 : 3 returns 4 since 7 is equal to 5+2.

5>3 ? a : b	returns a, since 5 is greater than 3.
a>b ? a : b	returns the greater one, a or b.

Contoh 3.10:

```
// conditional operator
#include <iostream>
using namespace std;

int main ()
{
  int a,b,c;

  a=2;
  b=7;
  c = (a>b)? a : b;

  cout << c << '\n';
}
```

Hasil:

```
7
```

Pada contoh diatas, a bernilai 2 dan b bernilai 7. Ekspresi pertama yang di proses (a>b) adalah false, sehingga nilai yang diambil adalah nilai kedua setelah tanda tanya (*result2*), yaitu b yang bernilai 7.

Comma operator (,)

Operator koma digunakan untuk memisahkan dua atau lebih ekspresi yang termasuk, di mana hanya satu ekspresi yang diharapkan. Ketika set ekspresi harus dievaluasi untuk nilai, hanya ekspresi paling kanan dianggap.

Contoh:

```
a = (b=3, b+2);
```

Kode diatas akan memasukkan nilai 3 ke variable b, dan memasukkan b+2 ke variable a. Maka, pada akhirnya, variable a akan bernilai 5 dan variable bernilai 3.

Contoh 3.11:

```
// Comma operator
#include <iostream>
using namespace std;

int main ()
{
  int a,b;

  a=(b=3, b+2);

  cout << a << '\n';
  cout << b << endl;
}
```

Hasil:

```
5
3
```

Bitwise Operators (&, |, ^, ~, <<, >>).

Operator Bitwise memodifikasi variabel menurut bit yang merepresentasikan nilai yang disimpan, atau dengan kata lain dalam representasi binary.

op	asm	Description
&	**AND**	Logical AND
|	**OR**	Logical OR
^	**XOR**	Logical exclusive OR
~	**NOT**	Complement to one (bit inversion)
<<	**SHL**	Shift Left
>>	**SHR**	Shift Right

Explicit type casting operators

Type casting operators memungkinkan untuk mengkonversikan tipe data yang sudah diberikan ke tipe data yang lain. Ada beberapa cara yang dapat dilakukan dalam C++, yang paling popular yaitu tipe baru dituliskan dalam tanda kurung **()** contoh:

```
int i;
float f = 3.14;
i = (int) f;
```

Contoh diatas, mengkonversikan nilai **3.14** menjadi nilai integer (**3**). Type casting operator yang digunakan **(int)**. Cara lainnya :

```
i = int ( f );
```

sizeof()

Operator ini menerma 1 parameter, dapat berupa type variabel atau variabel itu sendiri dan mengembalikan ukurannya type atau object tersebut dalam bytes :

```
a = sizeof (char);
```

Contoh diatas akan memberikan nilai 1 ke **a** karena **char** adalah tipe data dengan panjang 1 byte. Nilai yang diberikan oleh **sizeof** bersifat konstsn constant.

Prioritas pada operator

Contoh :

```
a = 5 + 7 % 2
```

Jawaban dari contoh diatas adalah 6. Tabel berikut ini adalah prioritas operator dari tinggi ke rendah :

Level	Precedence group	Operator	Description	Grouping
1	Scope	`::`	scope qualifier	Left-to-right
2	Postfix (unary)	`++ --`	postfix increment / decrement	Left-to-right
		`()`	functional forms	
		`[]`	subscript	
		`. ->`	member access	
3	Prefix (unary)	`++ --`	prefix increment / decrement	Right-to-left
		`~ !`	bitwise NOT / logical NOT	
		`+ -`	unary prefix	
		`& *`	reference / dereference	
		`new delete`	allocation / deallocation	
		`sizeof`	parameter pack	
		(*type*)	C-style type-casting	
4	Pointer-to-member	`.* ->*`	access pointer	Left-to-right
5	Arithmetic: scaling	`* / %`	multiply, divide, modulo	Left-to-right
6	Arithmetic: addition	`+ -`	addition, subtraction	Left-to-right
7	Bitwise shift	`<< >>`	shift left, shift right	Left-to-right
8	Relational	`< > <= >=`	comparison operators	Left-to-right
9	Equality	`== !=`	equality / inequality	Left-to-right
10	And	`&`	bitwise AND	Left-to-right
11	Exclusive or	`^`	bitwise XOR	Left-to-right
12	Inclusive or	`|`	bitwise OR	Left-to-right
13	Conjunction	`&&`	logical AND	Left-to-right
14	Disjunction	`||`	logical OR	Left-to-right
15	Assignment-level expressions	`= *= /= %= += -= >>= <<= &= ^= |=`	assignment / compound assignment	Right-to-left
		`?:`	conditional operator	
16	Sequencing	`,`	comma separator	Left-to-right

4. Basic Input/Output

Contoh program dari bagian sebelumnya memberikan sedikit interaksi dengan pengguna. Mereka hanya dicetak secara sederhana di layar, tetapi standar *library* memberikan banyak cara untuk berinteraksi dengan pengguna melalui fitur input / output. Bagian ini akan menyajikan pengenalan singkat ke beberapa fitur input/output yang paling berguna.

C ++ menggunakan abstraksi yang nyaman disebut *stream* untuk melakukan operasi input dan output dalam sekuensial media seperti layar, keyboard atau file. Sebuah *stream* adalah sebuah entitas di mana sebuah program dapat menyisipkan atau ekstrak karakter ke / dari. Tidak perlu untuk mengetahui rincian tentang media yang berhubungan dengan aliran atau spesifikasi internal. Semua yang perlu kita ketahui adalah bahwa aliran adalah sumber / tujuan dari karakter, dan bahwa karakter ini disediakan / diterima secara berurutan (yaitu, satu per satu).

Standar *library* mendefinisikan beberapa objek stream yang dapat digunakan untuk mengakses apa yang dianggap sebagai sumber standar dan tujuan dari karakter dengan lingkungan di mana program berjalan:

stream	Description
cin	standard input stream
cout	standard output stream
cerr	standard error (output) stream
clog	standard logging (output) stream

Kita akan melihat lebih detail hanya cout dan cin (standar output dan input *stream*); cerr dan clog juga termasuk output stream, sehingga mereka pada dasarnya bekerja seperti cout, dengan satu-satunya perbedaan adalah bahwa mereka mengidentifikasi *stream* untuk tujuan tertentu: pesan kesalahan dan logging; yang, dalam banyak kasus, di sebagian besar lingkungan setup, mereka benar-benar melakukan hal yang sama: mereka mencetak pada layar, meskipun mereka juga dapat secara individual diarahkan.

Standard output (cout)
Pada kebanyakan lingkungan program, secara default standard keluaran adalah layar monitor, dan C++ menngunakan cout untuk perintah tersebut.

Untuk format operasi output, cout digunakan bersama dengan operasi insert (<<).

Contoh 4.1:

```
cout << "Output sentence"; // prints Output sentence on screen
cout << 120;               // prints number 120 on screen
cout << x;                 // prints the value of x on screen
```

Pada contoh diatas, contoh baris pertama memasukkan kalimat "Output sentence", baris kedua memasukkan angka 120 dan baris ketiga memasukkan nilai yang dimiliki oleh variable x ke dalam cout.

Contoh 4.2:

```
cout << "Hello";  // prints Hello
cout << Hello;    // prints the content of variable Hello
```

Pada contoh 2, nilai yang dihasilkan oleh kedua baris tersebut berbeda. Pada baris pertama, menampilkan kata Hello, sedangkan baris kedua menghasilkan nilai yang dimiliki oleh variable Hello.

Operasi insert (<<) dapat dirangkai dalam satu statemen:

```
cout << "This " << " is a " << "single C++ statement";
```

Statemen diatas akan mencetak teks This is a single C++ statement.

Rangkaian operasi insert ini sangat berguna untuk menggabungkan literal dan variable dalam satu pernyataan:

```
 cout << "I am " << age << " years old and my zipcode is " << zipcode;
```

Perintah cout tidak secara otomatis memindahkan hasil output ke baris baru, meskipun penulisan cout di baris baru pada program. Untuk memindahkan ke baris baru, dapat menggunakan perintah "\n".

Contoh 4.3:

```
cout << "First sentence.\n";
cout << "Second sentence.\nThird sentence.";
```

Hasil dari kode di atas adalah:

```
First sentence.
Second sentence.
Third sentence.
```

Cara lain adalah menggunakan "endl"

```
cout << "First sentence." << endl;
cout << "Second sentence." << endl;
```

Hasil dari kode di atas adalah:

```
First sentence.
Second sentence.
```

4.1.Standard Input (cin)

Untuk format operasi input, cin digunakan bersama-sama dengan operator ekstraksi, yang ditulis sebagai >> (yaitu, dua tanda "lebih besar dari"). Operator ini kemudian diikuti oleh variabel dimana data yang akan disimpan.

```
int age;
cin >> age;
```

Pernyataan pertama menyatakan variabel age bertipe int, dan pernyataan kedua mengekstrak dari cin nilai yang akan disimpan di dalamnya. Operasi ini membuat program menunggu masukan dari cin; umumnya, ini berarti bahwa program akan menunggu pengguna untuk memasukkan beberapa nilai dengan keyboard. Dalam hal ini, perhatikan bahwa karakter diperkenalkan menggunakan keyboard hanya dikirimkan ke program ketika tombol ENTER (atau RETURN) ditekan. Setelah pernyataan dengan operasi ekstraksi pada cin tercapai, program akan menunggu selama diperlukan sampai beberapa nilai dimasukkan.

Contoh 4.4:

```
// i/o example

#include <iostream>
using namespace std;

int main ()
{
  int i;
  cout << "Please enter an integer value: ";
  cin >> i;
  cout << "The value you entered is " << i;
  cout << " and its double is " << i*2 << ".\n";
  return 0;
}
```

Hasil:

```
Please enter an integer
value: 702
The value you entered is
702 and its double is 1404.
```

Ekstraksi pada cin juga dapat dirangkai untuk meminta lebih dari satu masukan dalam sebuah pernyataan tunggal:

Contoh 4.5:

```
cin >> a >> b;
```

Contoh 4.6:

```
cin >> a;
cin >> b;
```

Contoh 5 memiliki arti seperti pada contoh 6.

4.2. cin dan strings

Operator ekstraksi dapat digunakan pada cin untuk mendapatkan string karakter dengan cara yang sama seperti dengan jenis data fundamental:

Contoh 4.7:

```
string mystring;
cin >> mystring;
```

Untuk mendapatkan seluruh baris dari cin, terdapat fungsi, yang disebut getline, yang mengambil *stream* (cin) sebagai argumen pertama, dan variabel string sebagai kedua.

Contoh 4.8:

```
// cin with strings
#include <iostream>
#include <string>
using namespace std;

int main ()
{
  string mystr;
  cout << "What's your name? ";
  getline (cin, mystr);
  cout << "Hello " << mystr << ".\n";
  cout << "What is your favorite team? ";
  getline (cin, mystr);
  cout << "I like " << mystr << " too!\n";
  return 0;
}
```

Hasil

```
What's your name? Homer Simpson
Hello Homer Simpson.
What is your favorite team? The
Isotopes
I like The Isotopes too!
```

4.3. Stringstream

Header standar <sstream> mendefinisikan sebuah tipe yang disebut stringstream yang memungkinkan string untuk diperlakukan sebagai stream, sehingga memungkinkan operasi ekstraksi atau penyisipan dari / ke string dengan cara yang sama seperti yang dilakukan pada cin dan cout. Fitur ini sangat berguna untuk mengkonversi string ke nilai numerik dan sebaliknya. Misalnya, untuk mengekstrak integer dari string kita bisa menulis:

```
string mystr ("1204");
int myint;
stringstream(mystr) >> myint;
```

Kode di atas menyatakan mystr dimasukkan nilai "1204". Variabel myint bertipe integer. Kemudian, baris ketiga menggunakan stringstream untuk mengekstrak variabel mystr. Potongan kode ini menyimpan nilai numerik 1204 dalam variabel yang disebut Myint.

Contoh 9:

```
// stringstreams
#include <iostream>
#include <string>
#include <sstream>
using namespace std;

int main ()
{
  string mystr;
  float price=0;
  int quantity=0;

  cout << "Enter price: ";
  getline (cin,mystr);
  stringstream(mystr) >> price;
  cout << "Enter quantity: ";
  getline (cin,mystr);
  stringstream(mystr) >> quantity;
  cout << "Total price: " << price*quantity << endl;
  return 0;
}
```

```
Enter price: 22.25
Enter quantity: 7
Total price: 155.75
```

5.Struktur Kontrol

Sebuah program biasanya tidak terbatas hanya pada intruksi yang terurut saja, tetapi juga memungkinkan terjadinya percabangan, perulangan dan pengambilan keputusan. Untuk mengatasi kebutuhan itu C++ menyediakan struktur kontrol yang dapat menangani hal-hal tersebut.

Untuk membahas hal tersebut diatas, akan ditemui istilah ***block of instructions***. Blok instruksi adalah sekumpulan instruksi yang dibatasi dengan tanda semicolon (;) tetapi dikelompokan dalam satu blok yang dibatasi dengan kurung kurawal **{ }**.

5.1.Struktur Kondisional : *if* and *else*

Digunakan untuk mengeksekusi sebuah atau satu blok instruksi jika kondisi terpenuhi, sintaks:

```
if (condition) statement
```

condition merupakan ekspresi yang dievaluasi. Jika kondisi bernilai **true**, maka ***statement*** akan dijalankan. Jika **false**, maka ***statement*** akan diabaikan dan program menjalankan instruksi selanjutnya.

Contoh, Akan tercetak **x is 100** jika nilai yang disimpan pada variable **x** adalah 100:

```
if (x == 100)
  cout << "x is 100";
```

Jika ada lebih dari satu instruksi yang akan dijalankan maka harus dibuat dalam blok instruksi dengan menggunakan tanda kurung kurawal { }:

```
if (x == 100)
 {
  cout << "x is ";
  cout << x;
 }
```

Dapat juga menggunakan keyword *else,* jika kondisi tidak terpenuhi. Penulisannya digabungkan dengan **if** :

```
if (condition) statement1 else statement2
```

Contoh 5.1:

```
if (x == 100)
  cout << "x is 100";
else
  cout << "x is not 100";
```

Akan tercetak **x is 100** jika nilai x adalah 100, jika tidak akan tercetak **x is not 100**.

Contoh 5.2:

```
//structure control
#include <iostream>
using namespace std;

int main()
{
    int x;
    cout << "Masukkan nilai x: ";
    cin >> x;

    if (x == 100)
        cout << "x is 100";
    else
    {
        cout << "x is not 100" << endl;
        cout << "x = " << x;
    }
}
```

Hasil:

```
Masukkan nilai x: 107
X is not 100
X = 107
```

Contoh 5.3:

```
if (x > 0)
  cout << "x is positive";
else if (x < 0)
  cout << "x is negative";
else
  cout << "x is 0";
```

5.2.Struktur perulangan (loops)

Loops merupakan perulangan *statement* dengan jumlah tertentu jika kondisi terpenuhi.

5.2.1.The *while* loop.

Sintaks:

while (*expression***)** *statement*

Fungsi dari statement diatas adalah mengulang *statement* jika *expression* bernilai *true*.

Contoh 5.4:

```
// custom countdown using while
#include <iostream>
using namespace std;

int main ()
{
  int n = 10;

  while (n>0) {
    cout << n << ", ";
    --n;
  }

  cout << "liftoff!\n";
}
```

```
10, 9, 8, 7, 6, 5, 4, 3, 2, 1, liftoff!
```

Algoritma program dimulai dari **main** :

1. Nilai n adalah 10

2. Instruksi while mengevaluasi apakah **(n>0)**. Ada dua kemungkinan :

true: meng-eksekusi *statement* (step **3,**)

false: melompati *statement*. lanjut ke step **5.**.

3.Mengeksekusi *statement* : cout << n << ", ";
--n;

(Menampilkan **n** di layar dan mengurangi **n** dengan 1).

4. Akhir dari blok. kembali ke step **2.**

5. lanjut menuju program setelah blok. Cetak : **liftoff!** dan program berakhir.

5.2.2.The *do-while* loop.

Format:

```
do statement while (condition);
```

Secara fungsional, hampir sama dengan *while* loop, hanya saja *condition* dalam *do-while* dievaluasi setelah eksekusi *statement* , dengan kata lain, sedikitnya satu kali eksekusi *statement* walaupun kondisi tidak terpenuhi.

Contoh 5.5:

```
// echo machine
#include <iostream>
#include <string>
using namespace std;

int main ()
{
  string str;
  do {
    cout << "Enter text: ";
    getline (cin,str);
    cout << "You entered: " << str << '\n';
  } while (str != "goodbye");
}
```

Hasil:

```
Enter text: hello
You entered: hello
Enter text: who's there?
You entered: who's there?
Enter text: goodbye
You entered: goodbye
```

Contoh 5.6:

```
// number echoer
#include <iostream>
int main ()
{
  unsigned long n;
  do {
    cout << "Enter number (0 to end): ";
    cin >> n;
    cout << "You entered: " << n << "\n";
  } while (n != 0);
  return 0;
}
```

Hasil:

```
Enter number (0 to end): 123
You entered: 123
Enter number (0 to end): 160277
You entered: 160277
Enter number (0 to end): 0
You entered: 0
```

5.3.The *for* loop.

Format :

for (*initialization*; *condition*; *increase*) *statement*;

Fungsinya akan mengulang *statement* jika *condition* bernilai benar. Sama seperti *while* loop., hanya saja **for** memungkinkan untuk memberikan instruksi *initialization* dan intruksi *increase,* sehingga dapat menampilkan loop dengan counter.

Algoritma perulangan *for* :

1. *initialization,* digunakan untuk memberikan nilai awal untuk variable counter. Dieksekusi hanya sekali.
2. *condition,* dievaluasi, jika bernilai **true** maka loop berlanjut, sebaliknya loop berhenti dan *statement* diabaikan

3. *statement,* dieksekusi, bisa berupa instruksi tunggal maupun blok instruksi (dalam tanda **{ }**).
4. *increase,* dieksekusi kemudian algoritma kembali ke step 2.

Contoh 5.7:

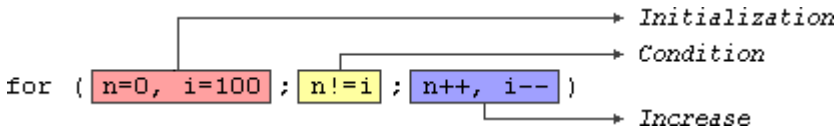
```
// countdown using a for loop
#include <iostream>
using namespace std;

int main ()
{
  for (int n=10; n>0; n--) {
    cout << n << ", ";
  }
  cout << "liftoff!\n";
}
```

```
10, 9, 8, 7, 6, 5, 4, 3, 2, 1,
liftoff!
```

Initialization dan *increase* bersifat optional. Sehingga dapat dituliskan : **for (;n<10;)** untuk for tanpa *initialization* dan *increase*; atau **for (;n<10;n++)** untuk for dengan *increase* tetapi tanpa *initialization*. Dengan operator koma (,) kita dapat mendeklarasikan lebih dari satu instruksi pada bagian manapun termasuk dalam loop **for**, contoh :

```
for ( n=0, i=100 ; n!=i ; n++, i-- )
      {
        // whatever here...
      }
```

Loop diatas akan meng-eksekusi sebanyak 50 kali :



nilai awal **n** = **0** dan **i** = **100**, dengan kondisi **(n!=i)** (yaitu **n** tidak sama dengan **i**). Karena **n** mengalami penambahan 1 dan **i** mengalami pengurangan 1, maka kondisi loop akan salah setelah loop yang ke-50, yaitu ketika **n** dan **i** bernilai 50.

5.4. Kontrol Percabangan (Bifurcation) dan Lompatan (jumps)

5.4.1. Instruksi *break*

Dengan menggunakan instruksi *break*, program akan keluar dari loop walaupun kondisi untuk berakhirnya loop belum terpenuhi. Dapat digunakan untuk mengakhiri *infinite loop*, atau untuk menyebabkan loop selesai sebelum saatnya.

Contoh 5.8:

```
// BREAK LOOP EXAMPLE
#include <iostream>
using namespace std;

int main ()
{
        int n;
        for (n=10; n>0; n--) {
                cout << n << ", ";
                if (n==3)
                {
                        cout << "countdown aborted!";
                        break;
                }
        }
}
```

Hasil:

```
10, 9, 8, 7, 6, 5, 4, 3, countdown
aborted!
```

5.4.2.Instruksi *continue*

Instruksi *continue* menyebabkan program akan melewati instruksi selanjutnya hingga akhir blok dalam loop. Atau dengan kata lain langsung melompat ke iterasi selanjutnya. Contoh berikut akan melewati angka 5 dalam hitungan mundur :

Contoh 5.8:

```
// continue loop example
#include <iostream>
using namespace std;

int main ()
{
  for (int n=10; n>0; n--) {
    if (n==5) continue;
    cout << n << ", ";
  }
  cout << "liftoff!\n";
}
```

Hasil:

```
10, 9, 8, 7, 6, 4, 3, 2, 1, liftoff!
```

5.4.3. Instruksi *goto*

Menyebabkan lompatan dalam program. Tujuan dari lompatan diidentifikasikan dengan label, yang berisikan argumen-argumen. penulisan label diikuti dengan tanda colon (:). Contoh :

Contoh 5.8:

```
// goto loop example
#include <iostream>
using namespace std;

int main ()
{
  int n=10;
mylabel:
  cout << n << ", ";
```

Hasil:

```
10, 9, 8, 7, 6, 5, 4, 3, 2, 1, liftoff!
```

```
  n--;
  if (n>0) goto mylabel;
  cout << "liftoff!\n";
}
```

5.4.4. Struktur Seleksi : *switch*.

Instruksi *switch* digunakan untuk membandingkan beberapa nilai konstan yang mungkin untuk sebuah ekspresi, hampir sama dengan *if* dan *else if*. Bentuk umumnya :

```
switch (expression) {
  case constant1:
    block of instructions 1
    break;
  case constant2:
    block of instructions 2
    break;
  .
  .
  .
  default:
    default block of instructions
  }
```

switch meng-evaluasi *expression* dan memeriksa apakah equivalen dengan *constant1*, jika ya, maka akan meng-eksekusi *block of instructions 1* sampai terbaca keyword **break**, kemudian program akan lompat ke akhir dari stuktur selektif *switch*.

Jika *expression* tidak sama dengan *constant1*, maka akan diperiksa apakah *expression* equivalen dengan *constant2*. jika ya, maka akan dieksekusi *block of instructions 2* sampai terbaca **break**. Begitu seterusnya, jika tidak ada satupun konstanta yang sesuai maka akan mengeksekusi **default:**

contoh :

***switch* example**

```
switch (x) {
  case 1:
    cout << "x is 1";
    break;
  case 2:
    cout << "x is 2";
    break;
  default:
    cout << "value of x
unknown";
 }
```

***if-else* equivalent**

```
if (x == 1) {
  cout << "x is 1";
  }
else if (x == 2) {
  cout << "x is 2";
  }
else {
  cout << "value of x unknown";
 }
```

6.Function

6.1. Pengertian Fungsi

- Fungsi/function adalah suatu kumpulan instruksi/perintah/program yang dikelompokkan menjadi satu, letaknya terpisah dari program yang menggunakan fungsi tersebut, memiliki nama tertentu yang unik, dan digunakan untuk mengerjakan suatu tujuan tertentu. Dalam bahasa pemrograman lain fungsi dapat disebut sebagai subrutin (basic, VB) atau procedure (pascal, Delphi).

- Function adalah satu blok instruksi yang dieksekusi ketika dipanggil dari bagian lain dalam suatu program.

6.2. Kelebihan Fungsi

1. Dapat melakukan pendekatan top-down dan divide-and conquer: *Top-down*: penelusuran program mudah, *Divide-and-conquer*: program besar dapat dipisah menjadi program-program kecil.
2. Kode program menjadi lebih pendek, mudah dibaca, dan mudah dipahami.
3. Program dapat dikerjakan oleh beberapa orang sehingga program cepat selesai dengan koordinasi yang mudah.
4. Mudah dalam mencari kesalahan-kesalahan karena alur logika jelas dan sederhana.
5. Kesalahan dapat dilokalisasi dalam suatu modul tertentu saja.
6. Modifikasi program dapat dilakukan pada suatu modul tertentu saja tanpa mengganggu program keseluruhan.
7. Fungsi – fungsi menjadikan program mempunyai struktur yang jelas.
8. Dengan memisahkan langkah – langkah detail ke satu atau lebih fungsi – fungsi, maka fungsi utama (main) akan menjadi lebih pendek, jelas dan mudah dimengerti.
9. Fungsi –fungsi digunakan untuk menghindari penulisan program yang sama yang ditulis secara berulang – ulang.
10. Langkah – langkah tersebut dapat dituliskan sekali saja secara terpisah dalam bentuk fungsi. Selanjutnya bagian program yang membutuhkan langkah – langkah ini tidak perlu selalu menuliskannya, cukup memanggil fungsi tersebut.
11. Mempermudah dokumentasi.
12. Reusability: Suatu fungsi dapat digunakan kembali oleh program atau fungsi lain.

6.3. Jenis - Jenis Fungsi

1. Standard Library Function
   Yaitu fungsi-fungsi yang telah disediakan oleh *C* dalam file-file header atau librarynya. Untuk function ini kita harus mendeklarasikan terlebih dahulu library

yang akan digunakan, yaitu dengan menggunakan preprosesor direktif. Misalnya: #include <string>

2. Programmer-Defined Function
   Adalah function yang dibuat oleh programmer sendiri. Function ini memiliki nama tertentu yang unik dalam program, letaknya terpisah dari program utama, dan bisa dijadikan satu ke dalam suatu library buatan programmer itu sendiri yang kemudian juga di-includekan jika ingin menggunakannya.

Karena prinsip kerja program C sekuensial, maka :

- Jika bagian dari program yang menggunakan fungsi diletakkan sebelum definisi dari fungsi, maka deklarasi dari fungsi diperlukan.
- Akan tetapi jika bagian dari program yang menggunakan fungsi terletak setelah definisi dari fungsi, maka deklarasi dari fungsi dapat tidak dituliskan.

6.4. Format dari function

```
type name ( argument1, argument2, ...) statement
```

Dimana :

- ***type***, adalah tipe dari data yang akan dikembalikan/dihasilkan oleh *function*.
- ***name***, adalah nama yang memungkinkan kita memanggil function.
- ***arguments*** (dispesifikasikan sesuai kebutuhan). Setiap argumen terdiri dari tipe data diikuti identifier, seperti deklarasi variable (contoh, int x) dan berfungsi dalam function seperti variable lainnya. Juga dapat melakukan *passing parameters* ke function itu ketika dipanggil. Parameter yang berbeda dipisahkan dengan koma.
- ***statement***, merupakan bagian badan suatu *function*. Dapat berupa instruksi tunggal maupun satu blok instruksi yang dituliskan diantara kurung kurawal **{}**.

6.5. Penggunaan Programmer-Defined Function

- Adalah function(fungsi) yang dibuat oleh programmer sendiri.
- Function(fungsi) ini memiliki nama tertentu yang unik dalam program, letaknya terpisah dari program utama, dan bisa dijadikan satu ke dalam suatu library buatan programmer itu sendiri yang kemudian juga di-includekan jika ingin menggunakannya.

6.5.1. Jenis fungsi Programmer-Defined Function di C++ :

1. Fungsi yang Mengembalikan Nilai (nonvoid)

- Disebut non-void karena mengembalikan nilai kembalian yang berasal dari keluaran hasil proses function tersebut.
- Memiliki nilai kembalian.

- Dapat dianalogikan sebagai suatu variabel yang memiliki tipe data tertentu sehingga dapat langsung ditampilkan hasilnya.
- Ciri - ciri :
  - Ada keyword return.
  - Ada tipe data yang mengawali deklarasi fungsi.
  - Tidak ada keyword void.

Contoh 6.1

```
// function example
#include <iostream>
using namespace std;

int addition (int a, int b)
{
  int r;
  r=a+b;
  return r;
}

int main ()
{
  int z;
  z = addition (5,3);
  cout << "The result is " << z;
}
```

Hasil:

```
The result is 8
```

Program diatas, ketika dieksekusi akan mulai dari fungsi **main**. **main** function memulai dengan deklarasi variabel **z** dengan tipe **int**. Setelah itu instruksi pemanggilan fungsi **addition**. Jika diperhatikan, ada kesamaan antara sruktur pemanggilan dengan deklarasi fungsi itu sendiri, perhatikan contoh dibawah ini :

```
int addition (int a, int b)
               ↑      ↑
z = addition (  5   ,   3  );
```

Instruksi pemanggilan dalam fungsi **main** untuk fungsi **addition**, memberikan 2 nilai : **5** dan **3** mengacu ke parameter **int a** dan **int b** yang dideklarasikan untuk fungsi **addition**.

Saat fungsi dipanggil dari **main**, kontrol program beralih dari fungsi **main** ke fungsi **addition**. Nilai dari kedua parameter yang diberikan (**5** dan **3**) di-*copy* ke variable local ; **int a** dan **int b**.
Fungsi **addition** mendeklarasikan variable baru (**int r;**), kemudian ekspresi **r=a+b;**, yang berarti **r** merupakan hasil penjumlahan dari **a** dan **b**, dimana **a** dan **b** bernilai **5** dan **3** sehingga hasil akhirnya **8**. perintah selanjutnya adalah :

```
return (r);
```

Merupakan akhir dari fungsi **addition**, dan mengembalikan kontrol pada fungsi **main**. Statement **return** dikuti dengan variabel **r** (**return (r);**, sehingga nilai dari **r** yaitu 8 akan dikembalikan :

```
int addition (int a, int b)
  8
z = addition (  5  ,  3  );
```

Dengan kata lain pemanggilan fungsi (**addition (5,3)**) adalah menggantikan dengan nilai yang akan dikembalikan (**8**).

Dalam sebuah program, sebuah fungsi dapat dipanggil beberapa kali.

Contoh 6.2

```
// function example
#include <iostream>
using namespace std;

int subtraction (int a, int b)
{
  int r;
  r=a-b;
  return r;
}

int main ()
{
  int x=5, y=3, z;
  z = subtraction (7,2);
  cout << "The first result is " << z << '\n';
  cout << "The second result is " << subtraction (7,2) << '\n';
  cout << "The third result is " << subtraction (x,y) << '\n';
  z= 4 + subtraction (x,y);
  cout << "The fourth result is " << z << '\n';
}
```

Hasil:

```
The first result is 5
The second result is 5
The third result is 2
The fourth result is 6
```

Fungsi diatas melakukan pengurangan dan mengembalikan hasilnya. Jika diperhatikan dalam fungsi **main**, dapat dilihat beberapa cara pemanggilan fungsi yang berbeda.

Perhatikan penulisan pemanggilan *function,* format penulisan pada dasarnya sama.
Fungsi 1 :

```
z = subtraction (7,2);
cout << "The first result is " << z;
```

Fungsi 2 :

```
cout << "The second result is " << subtraction (7,2);
```

Fungsi 3 :

```
cout << "The third result is " << subtraction (x,y);
```

Hal lain dari contoh diatas, parameter yang digunakan adalah variable, bukan konstanta. Contoh diatas memberikan nilai dari **x** dan **y**, yaitu **5** dan **3**, hasilnya **2**.

Fungsi 4 :

```
z = 4 + subtraction (x,y);
```

Atau dapat dituliskan :

```
z = subtraction (x,y) + 4;
```

Akan memberikan hasil akhir yang sama. Perhatikan, pada setiap akhir ekspresi selalu diberi tanda semicolon (;).

Contoh 6.3

```
//function hitung luas segitiga
#include <iostream>
using namespace std;

int luas (int a, int t)
{return (0.5*a*t);}

main(){
   int al,tg;
   cout<<"Alas   = ";cin>>al;
   cout<<"Tinggi = ";cin>>tg;
   cout<<"Luas Segitiga = "<<luas(al,tg);
}
```

```
Alas = 5
Tinggi = 10
Luas Segitiga = 25
```

2. Fungsi yang Tidak Mengembalikan Nilai (void)

- Fungsi yang void sering disebut juga prosedur
- Disebut void karena fungsi tersebut tidak mengembalikan suatu nilai keluaran yang didapat dari hasil proses fungsi tersebut.
- Tidak dapat langsung ditampilkan hasilnya.
- Tidak memiliki nilai kembalian fungsi.
- Keyword void juga digunakan jika suatu function tidak mengandung suatu parameter apapun.
- Ciri - ciri :
  - Tidak adanya keyword return.
  - Tidak adanya tipe data di dalam deklarasi fungsi.
  - Menggunakan keyword void.

Deklarasi fungsi umumnya dimulai dengan menentukan tipe. Bentuk fungsi ini adalah fungsi yang mengembalikan nilai. Tapi bagaimana jika fungsi tidak perlu mengembalikan nilai? Dalam hal ini, jenis yang akan digunakan adalah void, yang merupakan tipe khusus untuk mewakili tidak adanya nilai. Sebagai contoh, sebuah fungsi yang hanya mencetak pesan mungkin tidak perlu mengembalikan nilai apapun:

Contoh 6.4:

```
// void function example
#include <iostream>
using namespace std;

void printmessage ()
{
  cout << "I'm a function!";
}

int main ()
{
  printmessage ();
}
```

Hasil:

```
I'm a function!
```

Argument passed by value dan by reference.

Parameter yang diberikan ke fungsi masih merupakan *passed by value*. Berarti, ketika memanggil sebuah fungsi, yang diberikan ke fungsi adalah nilainya, tidak pernah men-spesifikasikan variablenya. Sebagai Contoh, pemanggilan fungsi addition, menggunakan perintah berikut:

```
int x=5, y=3, z;
z = addition ( x , y );
```

Yang berarti memanggil fungsi **addition** dengan memberikan nilai dari **x** dan **y**, yaitu **5** dan **3,** bukan variabelnya.

```
int addition (int a, int b)
                 5    3
z = addition (  x  ,  y  );
```

Tetapi, dapat juga memanipulasi dari dalam fungsi, nilai dari variable external. Untuk hal itu, digunakan argument *passed by reference*.

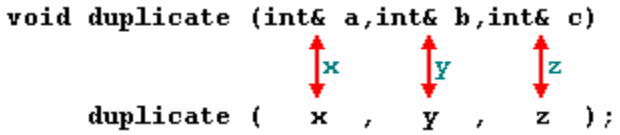
Contoh 6.5:

```
// passing parameters by reference
#include <iostream>
using namespace std;

void duplicate (int& a, int& b, int& c)
{
  a*=2;
  b*=2;
  c*=2;
}

int main ()
{
  int x=1, y=3, z=7;
  duplicate (x, y, z);
  cout << "x=" << x << ", y=" << y << ", z="
<< z;
  return 0;
}
```

Hasil:

```
x=2, y=6, z=14
```

Perhatikan deklarasi **duplicate,** tipe pada setiap argumen diakhiri dengan tanda *ampersand* (**&**), yang menandakan bahwa variable tersebut biasanya akan *passed by reference* dari pada *by value*.
Ketika mengirimkan variable *by reference,* yang dikirimkan adalah variabelnya dan perubahan apapun yang dilakukan dalam fungsi akan berpengaruh pada variable diluarnya.



Atau dengan kata lain, parameter yang telah ditetapkan adalah **a**, **b** dan **c** dan parameter yang digunakan saat pemanggilan adalah **x**, **y** dan **z**, maka perubahan pada **a** akan mempengaruhi nilai **x**, begitupun pada **b** akan mempengaruhi **y**, dan **c** mempengaruhi **z**.

Itu sebabnya mengapa hasil output dari program diatas adalah nilai variable dalam main dikalikan 2. jika deklarasi fungsi tidak diakhiri dengan tanda ampersand (&), maka variable tidak akan *passed by reference,* sehingga hasilnya akan tetap nilai dari **x**, **y** dan **z** tanpa mengalami perubahan.

Passing by reference merupakan cara efektif yang memungkinkan sebuah fungsi mengembalikan lebih dari satu nilai. Contoh, fungsi ini akan mengembalikan nilai sebelum dan sesudahnya dari nilai awal parameter :

Contoh 6.6:

```
// more than one returning value
#include <iostream>
using namespace std;

void prevnext (int x, int& prev, int& next)
{
  prev = x-1;
  next = x+1;
}

int main ()
{
  int x=100, y, z;
  prevnext (x, y, z);
  cout << "Previous=" << y << ", Next=" <<
z;
  return 0;
}
```

Hasil:

```
Previous=99, Next=101
```

Contoh 6.7:

```
//function hitung luas segitiga
#include <iostream>
using namespace std;

void luas(int& ls, int a, int t)
{ls = 0.5*a*t;}

main()
{
 int al,tg,hsl;
      cout<<"Alas    = ";cin>>al;
      cout<<"Tinggi = ";cin>>tg;
          luas(hsl,al,tg);
      cout<<"Luas Segitiga = "<<hsl;
}
```

Hasil:

```
Alas = 5
Tinggi = 10
Luas Segitiga = 25
```