ELEX MEDIA KOMPUTINDO

Cepat Menguasai C++

Ringkas dan tepat mempelajari C++
untuk programmer pemula

Jubilee Enterprise

Cepat Menguasai C++

Sanksi Pelanggaran Pasal 113
Undang-Undang Nomor 28 Tahun 2014
tentang Hak Cipta

1. Setiap Orang yang dengan tanpa hak melakukan pelanggaran hak ekonomi sebagaimana dimaksud dalam Pasal 9 ayat (1) huruf i untuk Penggunaan Secara Komersial dipidana dengan pidana penjara paling lama 1 (satu) tahun dan/atau pidana denda paling banyak Rp100.000.000 (seratus juta rupiah).

2. Setiap Orang yang dengan tanpa hak dan/atau tanpa izin Pencipta atau pemegang Hak Cipta melakukan pelanggaran hak ekonomi Pencipta sebagaimana dimaksud dalam Pasal 9 ayat (1) huruf c, huruf d, huruf f, dan/atau huruf h untuk Penggunaan Secara Komersial dipidana dengan pidana penjara paling lama 3 (tiga) tahun dan/atau pidana denda paling banyak Rp500.000.000,00 (lima ratus juta rupiah).

3. Setiap Orang yang dengan tanpa hak dan/atau tanpa izin Pencipta atau pemegang Hak Cipta melakukan pelanggaran hak ekonomi Pencipta sebagaimana dimaksud dalam Pasal 9 ayat (1) huruf a, huruf b, huruf e, dan/atau huruf g untuk Penggunaan Secara Komersial dipidana dengan pidana penjara paling lama 4 (empat) tahun dan/atau pidana denda paling banyak Rp1.000.000.000,00 (satu miliar rupiah).

4. Setiap Orang yang memenuhi unsur sebagaimana dimaksud pada ayat (3) yang dilakukan dalam bentuk pembajakan, dipidana dengan pidana penjara paling lama 10 (sepuluh) tahun dan/atau pidana denda paling banyak Rp4.000.000.000,00 (empat miliar rupiah).

Cepat Menguasai C++

Jubilee Enterprise

PENERBIT PT ELEX MEDIA KOMPUTINDO

Cepat Menguasai C++

Jubilee Enterprise

Diterbitkan pertama kali oleh
Penerbit PT Elex Media Komputindo
Kelompok Gramedia, Anggota IKAPI, Jakarta 2015

121150272
ISBN: 9786020258263

Dicetak oleh Percetakan PT Gramedia, Jakarta
Isi di luar tanggung jawab percetakan

Kata Pengantar

C++ dianggap sebagai bahasa pemrograman masa depan. Oleh karena itu, mempelajari C++ di saat sekarang bernilai investasi. Dari sudut pandang ekonomis inilah, kami berniat untuk menyuguhkan tema C++ kepada para pembaca sebagai salah satu alternatif pemrograman di tengah banyak tema programming lainnya yang gencar di dunia perbukuan.

Buku ini mengupas tentang C++ dan bagaimana Anda bisa mempelajari pemrograman ini dengan cepat dan ringkas. Diharapkan, setelah membaca buku ini, Anda akan mengenal bahasa pemrograman yang di masa mendatang akan dihargai sangat tinggi.

Kalau sudah begitu, kami berharap agar Anda bisa membaca buku ini dengan baik dan mempraktikkannya dalam latihan sehari-hari untuk membuat script yang praktis dan fungsional.

Yogyakarta, 10 Desember 2014

Gregorius Agung
Founder Jubilee Enterprise
"Information Technology is Our Passion and Book is Our Way"
Do you need top-notch IT Book? Just .... thinkjubilee.com

Daftar Isi

BAB 1

Sekilas Mengenai C++

Bahasa pemrograman C++ adalah bahasa pemrograman yang bersifat statis, compiled, general-purposed, case-sensitive, dan mendukung pemrograman prosedural, pemrograman berorientasi objek, maupun pemrograman generik.

C++ dikembangkan oleh Bjarne Stroustrup pada tahun 1979 di Bell Labs, Murray Hill, New Jersey sebagai bahasa pemrograman middle-level yang merupakan kombinasi bahasa high-level dan low-level.

Pemrograman Berorientasi Objek

C++ mendukung pemrograman berorientasi objek, termasuk empat pilar utama pemrograman berorientasi objek:

- Enkapsulasi.
- Data hiding.
- Pewarisan.
- Polimorfisme.

Library Standar

Library standar C++ terdiri dari tiga bagian berikut:

- Inti bahasa yang terdiri dari variabel, tipe data, literal, dan sebagainya.
- Library standar C++ berisi kumpulan fungsi untuk memanipulasi file, string, dan sebagainya.
- Standard Template Library (STL) berisi kumpulan method untuk memanipulasi struktur data, dan sebagainya.

Penggunaan C++

C++ digunakan oleh para programmer dalam berbagai domain aplikasi.

C++ banyak digunakan untuk membuat driver dan software lainnya yang secara langsung memanipulasi hardware.

Antarmuka sistem informasi Apple Macintosh dan Windows menggunakan C++ sebagai bahasa pemrograman utamanya.

BAB 2

Menyiapkan C++

Sebelum mulai menggunakan C++, Anda perlu menyiapkan software berikut dalam komputer Anda.

Editor Teks

Editor teks digunakan untuk menulis program Anda. Anda dapat menggunakan editor seperti: Windows Notepad, OS Edit command, Brief, Epsilon, EMACS, vim, atau vi.

File yang Anda buat dengan editor disebut source file, dan untuk C++ file biasanya diberi nama dengan ekstensi .cpp, .cp, atau .c.

Compiler C++

Compiler digunakan untuk melakukan proses compile kode program menjadi program yang dapat dijalankan.

Compiler yang sering digunakan adalah compiler GNU C/C++ yang dapat Anda gunakan secara gratis.

Anda juga dapat menggunakan compiler lain atau IDE seperti Eclipse, NetBeans, dan lainnya.

Menginstal Compiler GNU C/C++

Berikut ini langkah-langkah untuk menginstal GNU C/C++ (GCC) pada Linux, Mac OS, dan Windows.

UNIX/ Linux

Jika Anda menggunakan Liux atau UNIX, Anda dapat memeriksa apakah GCC sudah terinstal dengan menggunakan perintah berikut pada command line:

```
$ g++-v
```

Jika sudah terinstal, akan ditampilkan pesan seperti pada contoh berikut:

```
Using built-in specs.
Target: i386-redhat-linux
Configuredwith:../configure --prefix=/usr .......
Thread model: posix
gcc version 4.1.220080704(RedHat4.1.2-46)
```

Jika belum terinstal, Anda dapat menginstalnya dengan instruksi yang ada pada situs http://gcc.gnu.org/install/.

Mac OS

Jika Anda menggunakan Mac OS, Anda dapat mengunduh dan menginstal Xcode development environment pada situs Apple.

Anda dapat mengunduhnya dari situs developer.apple.com/ technologies/tools/.

Windows

Untuk menginstal GCC pada Windows, Anda perlu menginstal MinGW.

Untuk menginstal MinGW Anda pertama-tama dapat mengunduhnya pada situs ***www.mingw.org*** dan memilih versi terbaru dengan nama ***MinGW-<versi>.exe***.

Dalam menginstal MinGW, minimal Anda harus menginstal gcc-core, gcc-g++, binutils, dan MinGW runtime.

Setelah instalasi selesai, Anda kemudian dapat menjalankan gcc, g++, ar, ranlib, dlltool, dan tools GNU lainnya dari command line.

BAB 3

Sintaks Dasar C++

Sebuah program C++ dapat didefinisikan sebagai kumpulan objek yang saling berkomunikasi melalui method-method yang ada.

Berikut ini penjelasan singkat mengenai kelas, objek, method, dan variabel instance:

- Objek - sebuah objek memiliki keadaan dan perilaku. Contoh: Seekor anjing memiliki keadaan - warna, nama, jenis, dan memiliki perilaku - makan, berlari, menggonggong. Objek merupakan instance dari kelas.
- Kelas - kelas dapat didefinisikan sebagai sebuah template/ blueprint yang mendeskripsikan perilaku/keadaan yang didukung oleh tipe kelas tersebut.
- Method - method dasarnya merupakan perilaku. Sebuah kelas bisa memiliki beberapa method. Di dalam method, logika dituliskan, data dimanipulasi dan semua operasi dieksekusi.
- Variabel instance - setiap objek memiliki kumpulan variabel uniknya masing-masing. Keadaan suatu objek ditentukan oleh nilai-nilai yang dimasukkan dalam variabel instance yang ada.

Struktur Program C++

Berikut ini contoh kode sederhana yang menampilkan teks “Halo Dunia”.

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

// eksekusi program dimulai pada main()
int main()
{
 cout <<"Halo Dunia"; // menampilkan Halo Dunia
 return 0;
}
```

Berikut ini penjelasan bagian-bagian dari program tersebut:

- Bahasa pemrograman C++ mendefinisikan beberapa header, yang berisi informasi yang diperlukan atau informasi yang berguna bagi program. Pada contoh program di atas, header <iostream> diperlukan.
- Baris using namespace std; memberitahu compiler untuk menggunakan namespace std.
- Baris `//` eksekusi program dimulai pada main() merupakan komentar single-line. Komentar single-line diawali dengan tanda // dan berakhir pada akhir baris tersebut.
- Baris int main() adalah fungsi utama di mana eksekusi program mulai dijalankan.
- Baris cout <<"Halo Dunia"; menyebabkan pesan “Halo Dunia” ditampilkan pada layar monitor.
- Baris return 0; mengakhiri fungsi main() dan mengembalikan nilai 0 sebagai nilai balikan.

Compile dan Eksekusi Program C++

Berikut ini langkah-langkah untuk menyimpan file, compile dan menjalankan program.

- Buka editor teks dan tambahkan contoh kode di atas.
- Simpan file dengan nama: hello.cpp.
- Buka jendela command prompt dan buka direktori penyimpanan file.

- Ketikkan ‘g++ hello.cpp’ dan tekan enter untuk melakukan compile.
- Ketikkan ‘a.out’ untuk menjalankan program.
- Pesan ‘Halo Dunia’ akan ditampilkan.

```
$ g++ hello.cpp
$ ./a.out
Halo Dunia
```

Titik-koma dan Blok dalam C++

Dalam C++, titik koma merupakan terminator. Setiap statemen individual harus diakhiri dengan tanda titik koma.

Berikut contoh kode yang terdiri dari tiga statemen:

```
x = y;
y = y+1;
add(x, y);
```

Blok merupakan kumpulan statemen yang berhubungan logis yang dibungkus oleh tanda kurung kurawal buka dan tutup { }.

Berikut contohnya:

```
{
 cout <<"Halo Dunia"; // menampilkan Halo Dunia
 return 0;
}
```

Identifier C++

Identifier C++ merupakan nama yang digunakan untuk mengidentifikasi variabel, fungsi, kelas, modul, atau item lainnya yang dibuat oleh pengguna.

Sebuah identifier diawali dengan huruf A - Z, atau a - z atau underscore _ yang diikuti dengan huruf, underscore, dan angkah (0-9).

C++ bersifat case-sensitive sehingga Manpower dan manpower merupakan dua identifier yang berbeda.

Berikut ini beberapa contoh penulisan identifier:

```
mohd zara abc move_name a_123
myname50 _temp j a23b9 retVal
```

Kata Kunci C++

Berikut ini daftar kata kunci yang ada kalam C++. Kata-kata kunci ini tidak dapat digunakan sebagai nama identifier.

Asm	else	new	this
Auto	enum	operator	throw
Bool	explicit	private	true
Break	export	protected	try
Case	extern	public	typedef
Catch	false	register	typeid
Char	float	reinterpret_cast	typename
Class	for	return	union
Const	friend	short	unsigned
const_cast	goto	signed	using
continue	If	sizeof	virtual
Default	inline	Static	void
Delete	int	static_cast	volatile
Do	long	Struct	wchar_t
Double	mutable	Switch	while
dynamic_cast	namespace	Template	

Daftar kata kunci dalam C++

Spasi dalam C++

Baris yang berisi spasi atau berisi komentar disebut baris kosong dan akan diabaikan oleh compiler C++. Spasi dalam C++ digunakan untuk mendeskripsikan baris kosong, tab, baris baru dan komentar. Spasi memisahkan satu bagian statemen dengan bagian lainnya. Berikut contohnya:

```
int umur;
```

BAB 4

Komentar dalam C++

Komentar dalam suatu program merupakan statemen yang menjelaskan kode yang ada dalam program.

C++ mendukung penggunaan komentar single-line dan multi-line. Semua karakter yang ada dalam komentar akan diabaikan oleh compiler C++.

Berikut ini contoh komentar multi-line yang diawali dengan /* dan diakhiri dengan */:

```
/* Ini adalah komentar */

/* Ini adalah contoh
 * komentar multi-line
 * dalam C++
 */
```

Berikut ini contoh komentar single-line yang diawali dengan //:

```
#include<iostream>
usingnamespace std;

main()
{
 cout <<"Hello World"; // menampilkan Hello World

 return0;
}
```

Ketika kode tersebut di-compile, // menampilkan Hello World akan diabaikan dan kode tersebut akan menghasilkan output berikut:

```
Hello World
```

Dalam komentar /* dan */, karakter // tidak memiliki arti, dan sebaliknya dalam komentar //, karakter /* dan */ tidak memiliki arti.

Dengan demikian Ada dapat membuat komentar di dalam komentar lain:

```
/* Baris untuk menampilkan Hello World menjadi komentar:
 cout << "Hello World"; // menampilkan Hello World
*/
```

BAB 5

Tipe Data dalam C++

Tipe Data Primitif

C++ menyediakan beberapa tipe data primitif yang dapat Anda gunakan. Berikut ini daftar tipe data dasar dalam C++:

Tipe	Kata Kunci
Boolean	bool
Character	char
Integer	int
Floating point	float
Double floating point	double
Valueless	void
Wide character	wchar_t

Daftar tipe data primitif

Beberapa tipe data tersebut dapat dimodifikasi dengan modifier signed, unsigned, short, dan long.

Berikut ini tabel yang menjelaskan tipe variabel, lebar bit, dan nilai minimum dan maksimumnya.

Tipe	Lebar Bit	Jangkauan Nilai
char	1byte	-127 sampai 127 atau 0 sampai 255
unsigned char	1byte	0 sampai 255
signed char	1byte	-127 sampai 127
int	4bytes	-2147483648 sampai 2147483647
unsigned int	4bytes	0 sampai 4294967295
signed int	4bytes	-2147483648 sampai 2147483647
short int	2bytes	-32768 sampai 32767
unsigned short int	Range	0 sampai 65,535
signed short int	Range	-32768 sampai 32767
long int	4bytes	-2,147,483,647 sampai 2,147,483,647
signed long int	4bytes	sama dengan long int
unsigned long int	4bytes	0 sampai 4,294,967,295
float	4bytes	+/- 3.4e +/- 38 (~7 digits)
double	8bytes	+/- 1.7e +/- 308 (~15 digits)
long double	8bytes	+/- 1.7e +/- 308 (~15 digits)
wchar_t	2 atau 4 bytes	1 karakter

Daftar tipe variabel, lebar bit, dan jangkauannya

Ukuran variabel mungkin berbeda dengan tabel tersebut, tergantung compiler dan komputer yang Anda gunakan.

Berikut ini contoh kode untuk menampilkan ukuran tipe data pada komputer Anda:

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

int main()
{
 cout <<"Ukuran char : "<<sizeof(char)<< endl;
 cout <<"Ukuran int : "<<sizeof(int)<< endl;
 cout <<"Ukuran short int : "<<sizeof(shortint)<< endl;
 cout <<"Ukuran long int : "<<sizeof(longint)<< endl;
 cout <<"Ukuran float : "<<sizeof(float)<< endl;
 cout <<"Ukuran double : "<<sizeof(double)<< endl;
 cout <<"Ukuran wchar_t : "<<sizeof(wchar_t)<< endl;

 return0;
}
```

Kode tersebut menggunakan endl untuk memasukkan baris baru, operator << untuk menampilkan beberapa nilai, dan fungsi sizeof() untuk mendapatkan ukuran dari tipe data.

Kode tersebut akan menghasilkan output yang mungkin berbeda untuk masing-masing komputer:

```
Ukuran char:1
Ukuran int:4
Ukuran shortint:2
Ukuran longint:4
Ukuran float:4
Ukuran double:8
Ukuran wchar_t:4
```

Deklarasi typedef

Anda dapat membuat nama baru untuk tipe yang ada dengan menggunakan typedef.

Berikut sintaksnya:

```
typedef type namabaru;
```

Berikut contoh penggunaannya:

```
typedef int intbaru;

intbaru jarak;
```

Tipe Enum

Tipe enum mendeklarasikan nama tipe opsional dan menentukan beberapa identifier yang dapat digunakan sebagai nilai dari tipe tersebut. Setiap enumerator merupakan konstanta dengan tipe enum.

Berikut sintaksnya:

```
enum nama-enum { daftar nama } daftar-variabel;
```

Berikut ini contoh enum warna dan variabel w dengan tipe warna. Variabel w berisi nilai “biru”:

```
enum warna { merah, hijau, biru } w;
w = biru;
```

Secara default, nilai dari nama pertama adalah 0, nama kedua 1, nama ketiga 2, dan seterusnya. Tetapi Anda dapat mengatur nilai nama dengan menambah initializer.

Berikut contohnya, hijau memiliki nilai 5 dan biru akan memiliki nilai 6:

```
enum warna { merah, hijau=5, biru } w;
```

BAB 6

Tipe Variabel dalam C++

Variabel merupakan wadah penyimpanan yang memiliki nama/identifier yang dapat dimanipulasi oleh program. Setiap variabel dalam C++ memiliki tipe tertentu, yang menentukan ukuran dan layout dari memori variabel; jangkauan nilai yang dapat ditampung; dan operasi-operasi yang dapat dijalankan terhadap variabel.

Definisi Variabel dalam C++

Berikut sintaks untuk mendefinisikan variabel:

```
tipe daftar_variabel;
```

Berikut ini contoh dalam mendefinisikan variabel:

```
int i, j, k;
char c, ch;
float f, gaji;
double d;
```

Variabel dapat diinisialisasi (memasukkan nilai dalam variabel) pada saat dideklarasi. Inisializer terdiri dari tanda sama dengan yang diikuti dengan ekspresi konstan sebagai berikut:

```
tipe nama_variabel = nilai;
```

Berikut contohnya:

```
extern int d = 3, f = 5; // deklarasi d dan f.
int d = 3, f = 5; // definisi dan inisialisasi d dan f.
byte z = 22; // definisi dan inisialisasi z.
char x = 'x'; // variabel x memiliki nilai 'x'.
```

Deklarasi Variabel dalam C++

Berikut ini contoh deklarasi variabel di bagian awal, yang kemudian didefinisikan diinisialisasi dalam fungsi main():

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

// Definisi variabel:
extern int a, b;
extern int c;
extern float f;

int main ()
{
 // Definisi variabel:
 int a, b;
 int c;
 float f;

 // Inisialisasi aktual
 a =10;
 b =20;
 c = a + b;

 cout << c << endl ;

 f =70.0/3.0;
 cout << f << endl ;

 return0;
}
```

Kode tersebut akan menghasilkan output berikut:

```
30
23.3333
```

Konsep yang sama juga berlaku untuk deklarasi fungsi di mana Anda menentukan nama fungsi pada waktu deklarasi dan definisi aktualnya dapat dibuat di tempat lain. Berikut contohnya:

```
// deklarasi fungsi
int fung();

int main()
{
 // pemanggilan fungsi
 int i = fung();
}

// definisi fungsi
int fung()
{
 return 0;
}
```

BAB 7

Lingkup Variabel

Lingkup adalah bagian/area dalam program. Ada tiga area di mana variabel dapat dideklarasikan:

- Di dalam fungsi atau blok yang disebut variabel lokal.
- Pada definisi parameter fungsi yang disebut parameter formal.
- Di luar fungsi yang disebut variabel global.

Variabel Lokal

Variabel yang dideklarasikan di dalam suatu fungsi atau blok adalah variabel lokal. Berikut contoh penggunaannya:

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

int main ()
{
 // Deklarasi variabel lokal:
 int a, b;
 int c;

 // inisialisasi aktual
 a =10;
 b =20;
 c = a + b;

 cout << c;

 return0;
}
```

Variabel Global

Variabel global didefinisikan di luar fungsi, biasanya di bagian atas kode program.

Variabel global dapat diakses oleh semua fungsi dan dapat digunakan oleh program setelah dideklarasikan.

Berikut ini contoh penggunaan variabel lokal dan global:

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

// Deklarasi variabel global:
int g;

int main ()
{
 // Deklarasi variabel lokal:
 int a, b;

 // inisialisasi aktual
 a =10;
 b =20;
 g = a + b;

 cout << g;

 return0;
}
```

Inisialisasi Variabel Lokal dan Global

Ketika sebuah variabel didefinisikan, variabel tersebut tidak diinisialisasi oleh sistem. Anda harus menginisialisasinya secara manual.

Variabel global diinisialisasi secara otomatis oleh sistem jika Anda mendefinisikannya sebagai berikut:

Tipe Data	**Initializer**
Int	0
Char	'\0'
Float	0
Double	0
Pointer	NULL

BAB 8

Konstanta dalam C++

Konstanta merupakan referensi nilai tetap yang tidak dapat diubah. Konstanta berperan seperti variabel biasa, tetapi nilai konstanta bersifat tetap dan tidak dapat diubah setelah didefinisikan.

Mendefinisikan Konstanta

Konstanta biasanya didefinisikan dengan menggunakan huruf kapital. Ada dua cara untuk mendefinisikan konstanta dalam C++:

- Menggunakan preprocessor #define.
- Menggunakan kata kunci const.

Preprocessor #define

Berikut ini bentuk penggunaan preprocessor #define:

```
#define identifier nilai
```

Berikut contoh penggunaannya:

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

#define PANJANG 10
#define LEBAR 5
#define BARISBARU '\n'

int main()
```

```
{
 int luas;

 luas = PANJANG * LEBAR;

 cout << luas;
 cout << BARISBARU;

 return 0;
}
```

Kode tersebut akan menghasilkan output berikut:

```
50
```

Kata Kunci Const

Anda dapat menggunakan awalan const untuk mendeklarasikan konstanta dengan tipe tertentu.

Berikut sintaksnya:

```
const tipe nama_konstanta = nilai;
```

Berikut contoh penggunaannya:

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

int main()
{
 const int PANJANG =10;
 const int LEBAR =5;
 const char BARISBARU ='\n';

 int luas;

 luas = PANJANG * LEBAR;

 cout << luas;
 cout << BARISBARU;

 return 0;
}
```

Kode tersebut akan menghasilkan output berikut:

```
50
```

BAB 9

Operator dalam C++

Operator Aritmatika

Operator aritmatika digunakan pada ekspresi matematik seperti pada operasi aljabar. Berikut ini daftar operator aritmatika.

Diasumsikan bahwa variabel A bernilai 10 dan B bernilai 20, maka:

Operator	Deskripsi	Contoh
+	Penjumlahan - Menambahkan nilai-nilai yang ada di kedua sisi operator	A + B hasilnya 30
-	Pengurangan - Mengurangkan nilai operan di sebelah kiri dengan nilai operan di sebelah kanan	A - B hasilnya -10
*	Perkalian - Mengalikan nilai-nilai yang ada di kedua sisi operator	A * B hasilnya 200
/	Pembagian - Membagi nilai operan di sebelah kiri dengan nilai operan di sebelah kanan	B / A hasilnya 2

%	Modulus - Melakukan pembagian nilai operan di sebelah kiri dengan nilai operan di sebelah kanan dan mengembalikan sisa nilainya	B % A hasilnya 0
++	Peningkatan - Menambahkan 1 pada nilai operan	B++ hasilnya 21
--	Penurunan - Mengurangkan 1 dari nilai operan	B-- hasilnya 19

Operator aritmatika dalam C++

Berikut ini contoh penggunaan operator aritmatika:

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

main()
{
 int a =21;
 int b =10;
 int c;

 c = a + b;
 cout <<"Line 1 - Nilai c adalah :"<< c << endl ;

 c = a - b;
 cout <<"Line 2 - Nilai c adalah :"<< c << endl ;

 c = a * b;
 cout <<"Line 3 - Nilai c adalah :"<< c << endl ;

 c = a / b;
 cout <<"Line 4 - Nilai c adalah :"<< c << endl ;

 c = a % b;
 cout <<"Line 5 - Nilai c adalah :"<< c << endl ;

 c = a++;
 cout <<"Line 6 - Nilai c adalah :"<< c << endl ;

 c = a--;
 cout <<"Line 7 - Nilai c adalah :"<< c << endl ;

 return0;
}
```

Kode tersebut akan menghasilkan output berikut:

```
Line 1 - Nilai c adalah :31
Line 2 - Nilai c adalah :11
Line 3 - Nilai c adalah :210
```

```
Line 4 - Nilai c adalah :2
Line 5 - Nilai c adalah :1
Line 6 - Nilai c adalah :21
Line 7 - Nilai c adalah :22
```

Operator Relasional

Berikut ini operator relasional yang didukung dalam C++.

Diasumsikan bahwa variabel A bernilai 10 dan B bernilai 20, maka:

Operator	Deskripsi	Contoh
==	Memeriksa apakah nilai kedua operan sama atau tidak, jika sama maka kondisi bernilai benar.	(A == B) adalah tidak benar.
!=	Memeriksa apakah nilai kedua operan sama atau tidak, jika tidak sama maka kondisi bernilai benar.	(A != B) adalah benar.
>	Memeriksa apakah nilai operan di sebelah kiri lebih dari nilai operan di sebelah kanan, jika ya maka kondisi bernilai benar.	(A > B) adalah tidak benar.
<	Memeriksa apakah nilai operan di sebelah kiri kurang dari nilai operan di sebelah kanan, jika ya maka kondisi bernilai benar.	(A < B) adalah benar.
>=	Memeriksa apakah nilai operan di sebelah kiri lebih dari atau sama dengan nilai operan di sebelah kanan, jika ya maka kondisi bernilai benar.	(A >= B) adalah tidak benar.
<=	Memeriksa apakah nilai operan di sebelah kiri kurang dari atau sama dengan nilai operan di sebelah kanan, jika ya maka kondisi bernilai benar.	(A <= B) adalah benar.

Operator relasional dalam C++

Berikut ini contoh penggunaannya:

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

main()
{
 int a =21;
 int b =10;
 int c ;

 if( a == b )
 {
 cout <<"Line 1 - a sama dengan b"<< endl ;
 }
 else
 {
 cout <<"Line 1 - a tidak sama dengan b"<< endl ;
 }

 if( a < b )
 {
 cout <<"Line 2 - a kurang dari b"<< endl ;
 }
 else
 {
 cout <<"Line 2 - a tidak kurang dari b"<< endl ;
 }

 if( a > b )
 {
 cout <<"Line 3 - a lebih dari b"<< endl ;
 }
 else
 {
 cout <<"Line 3 - a tidak lebih dari b"<< endl ;
 }

 /* Mengubah nilai a dan b */
 a =5;
 b =20;

 if( a <= b )
 {
 cout <<"Line 4 - a kurang dari \
 atau sama dengan b"<< endl ;
 }

 if( b >= a )
 {
 cout <<"Line 5 - b lebih dari \
 atau sama dengan a"<< endl ;
 }

 return0;
}
```

Kode tersebut akan menghasilkan output berikut:

```
Line 1 - a tidak sama dengan b
Line 2 - a tidak kurang dari b
Line 3 - a lebih dari b
Line 4 - a kurang dari atau sama dengan b
Line 5 - b lebih dari atau sama dengan a
```

Operator Logika

Berikut ini daftar operator logika dalam C++.

Diasumsikan bahwa nilai variabel A adalah 1 dan B adalah 0, maka:

Operator	Deskripsi	Contoh
&&	Operator logika AND. Jika kedua operan bukan nol, maka kondisi bernilai benar.	(A && B) adalah tidak benar.
||	Operator logika OR. Jika ada operan yang bukan nol, maka kondisi bernilai benar.	(A || B) adalah benar.
!	Operator logika NOT. Digunakan untuk membalik keadaan logika dari operan. Jika kondisi bernilai benar maka operator NOT akan membuatnya menjadi tidak benar.	!(A && B) adalah benar.

Operator logika dalam C++

Berikut ini contoh penggunaannya:

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

main()
{
 int a =5;
 int b =20;
 int c ;

 if( a && b )
 {
 cout <<"Line 1 - Kondisi benar"<< endl ;
 }
```

```
 if( a || b )
 {
 cout <<"Line 2 - Kondisi benar"<< endl ;
 }

 /* Mengubah nilai a dan b */
 a =0;
 b =10;

 if( a && b )
 {
 cout <<"Line 3 - Kondisi benar"<< endl ;
 }
 else
 {
 cout <<"Line 4 - Kondisi tidak benar"<< endl ;
 }

 if(!(a && b))
 {
 cout <<"Line 5 - Kondisi benar"<< endl ;
 }

 return0;
}
```

Kode tersebut akan menghasilkan output berikut:

```
Line 1 - Kondisi benar
Line 2 - Kondisi benar
Line 3 - Kondisi tidak benar
Line 4 - Kondisi benar
```

Operator Assignment

Berikut ini daftar operator assignment dalam C++:

Operator	Deskripsi	Contoh
=	Memasukkan nilai pada operan di sebelah kanan ke dalam operan di sebelah kiri	C = A + B akan memasukkan nilai A + B ke dalam C
+=	Menjumlahkan nilai operan di sebelah kiri dengan nilai operan di sebelah kanan dan memasukkan hasilnya ke dalam operan di sebelah kiri	C += A hasilnya sama dengan C = C + A

-=	Mengurangkan nilai operan di sebelah kiri dengan nilai operan di sebelah kanan dan memasukkan hasilnya ke dalam operan di sebelah kiri	C -= A hasilnya sama dengan C = C - A
*=	Mengalikan nilai operan di sebelah kiri dengan nilai operan di sebelah kanan dan memasukkan hasilnya ke dalam operan di sebelah kiri	C *= A hasilnya sama dengan C = C * A
/=	Membagi nilai operan di sebelah kiri dengan nilai operan di sebelah kanan dan memasukkan hasilnya ke dalam operan di sebelah kiri	C /= A hasilnya sama dengan C = C / A
%=	Menggunakan operasi modulus terhadap kedua operan dan memasukkan hasilnya ke dalam operan di sebelah kiri	C %= A hasilnya sama dengan C = C % A

Operator assignment dalam C++

Berikut contoh penggunaan operator assignment:

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

main()
{
 int a =21;
 int c ;

 c = a;
 cout <<"Line 1 - Operator =, Nilai c = : "<<c<< endl ;

 c += a;
 cout <<"Line 2 - Operator +=, Nilai c = : "<<c<< endl ;

 c -= a;
 cout <<"Line 3 - Operator -=, Nilai c = : "<<c<< endl ;

 c *= a;
 cout <<"Line 4 - Operator *=, Nilai c = : "<<c<< endl ;

 c /= a;
 cout <<"Line 5 - Operator /=, Nilai c = : "<<c<< endl ;
```

```
 c =200;

 c %= a;
 cout <<"Line 6 - Operator %=, Nilai c = : "<<c<< endl ;

 return0;
}
```

Kode tersebut akan menghasilkan output berikut:

```
Line 1 - Operator =, Nilai c =:21
Line 2 - Operator +=,Nilai c =:42
Line 3 - Operator -=, Nilai c =:21
Line 4 - Operator *=, Nilai c =:441
Line 5 - Operator /=, Nilai c =:21
Line 6 - Operator %=, Nilai c =:11
```

BAB 10

Perulangan dalam C++

Perulangan While

Perulangan while adalah struktur kontrol yang memungkinkan Anda mengulangi suatu proses dengan jumlah perulangan tertentu.

Berikut ini sintaks dari perulangan while:

```
while(kondisi)
{
 statemen;
}
```

Selama kondisi bernilai benar, maka statemen dalam tubuh while akan terus dieksekusi. Berikut ini contoh penggunaannya:

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

int main()
{
 // Deklarasi variabel lokal:
 int a = 10;

 // eksekusi perulangan while
 while( a < 20 )
```

```
 {
 cout << "nilai a: " << a << endl;
 a++;
 }

 return 0;
}
```

Kode tersebut akan menghasilkan output berikut:

```
nilai a: 10
nilai a: 11
nilai a: 12
nilai a: 13
nilai a: 14
nilai a: 15
nilai a: 16
nilai a: 17
nilai a: 18
nilai a: 19
```

Perulangan do…while

Perulangan do...while sama seperti pengulangan while, tetapi perulangan do…while pasti akan dieksekusi minimal satu kali.

Berikut sintaksnya:

```
do
{
 statemen
}
while(kondisi);
```

Berikut ini contoh penggunaannya:

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

int main ()
{
 // Deklarasi variabel lokal:
 int a = 10;

 // eksekusi perulangan do
 do
 {
 cout << "nilai a: " << a << endl;
 a = a + 1;
 }while( a < 20 );

 return 0;
}
```

Kode tersebut akan menghasilkan output berikut:

```
nilai a: 10
nilai a: 11
nilai a: 12
nilai a: 13
nilai a: 14
nilai a: 15
nilai a: 16
nilai a: 17
nilai a: 18
nilai a: 19
```

Perulangan for

Perulangan for adalah struktur kontrol repetitif yang memungkinkan Anda untuk mejalankan proses dengan jumlah perulangan tertentu (jumlah perulangan sudah diketahui sebelumnya).

Berikut ini sintaksnya:

```
for(inisialisasi; kondisi; pertambahan)
{
 statemen
}
```

Berikut ini aliran proses dalam perulangan for:

- Bagian inisialisasi dieksekusi pertama kali dan hanya sekali. Bagian ini memungkinkan Anda untuk mendeklarasikan dan menginisialisasi variabel kontrol perulangan.
- Setelah itu, kondisi dievaluasi. Jika bernilai benar, statemen dalam tubuh for dieksekusi. Jika tidak, statemen tidak dieksekusi dan proses berlanjut pada bagian setelah perulangan for.
- Setelah statemen dalam for dieksekusi, aliran proses kembali pada bagian pertambahan. Statemen pertambahan ini memungkinkan Anda untuk meng-update variabel kontrol dalam for.
- Kondisi kemudian dievaluasi lagi. Jika benar, statemen dalam perulangan for kembali dieksekusi, dan dilanjutkan ke bagian pertambahan. Jika tidak, perulangan for selesai dan proses berlanjut pada bagian setelah perulangan for.

Berikut ini contoh penggunaannya:

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

int main ()
{
 // eksekusi perulangan for
 for( int a = 10; a < 20; a = a + 1 )
 {
 cout << "nilai a: " << a << endl;
 }

 return 0;
}
```

Berikut ini hasil output:

```
nilai a: 10
nilai a: 11
nilai a: 12
nilai a: 13
nilai a: 14
nilai a: 15
nilai a: 16
nilai a: 17
nilai a: 18
nilai a: 19
```

Perulangan Bersarang

C# memungkinkan Anda untuk menggunakan suatu perulangan di dalam perulangan yang lain.

Sintaks untuk perulangan for bersarang:

```
for ( inisialisasi; kondisi; pertambahan )
{
 for ( inisialisasi; kondisi; pertambahan )
 {
 statemen;
 }

 statemen;
}
```

Sintaks untuk perulangan while bersarang:

```
while(kondisi)
{
```

```
 while(kondisi)
 {
 statemen;
 }

 statemen;
}
```

Sintaks untuk perulangan do…while bersarang:

```
do
{
 statemen; // Anda dapat menggunakan beberapa statemen

 do
 {
 statemen;
 }
while( kondisi );

}
while( kondisi );
```

Anda juga dapat memasukkan jenis perulangan yang berbeda di dalam suatu perulangan. Perulangan for dapat berisi perulangan while dan sebaliknya.

Contoh berikut menggunakan perulangan for bersarang untuk menemukan bilangan prima di antara 2 sampai 100:

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

int main()
{
 int i, j;

 for(i=2; i<100; i++)
 {
 for(j=2; j <= (i/j); j++)
 if(!(i%j))
 break; // jika faktor ditemukan, bukan
 // bilangan prima
 if(j > (i/j))
 cout << i << " adalah bilangan prima\n";
 }

 return 0;
}
```

Kode tersebut akan menghasilkan output berikut:

```
2 adalah bilangan prima
3 adalah bilangan prima
5 adalah bilangan prima
7 adalah bilangan prima
11 adalah bilangan prima
13 adalah bilangan prima
17 adalah bilangan prima
19 adalah bilangan prima
23 adalah bilangan prima
29 adalah bilangan prima
31 adalah bilangan prima
37 adalah bilangan prima
41 adalah bilangan prima
43 adalah bilangan prima
47 adalah bilangan prima
53 adalah bilangan prima
59 adalah bilangan prima
61 adalah bilangan prima
67 adalah bilangan prima
71 adalah bilangan prima
73 adalah bilangan prima
79 adalah bilangan prima
83 adalah bilangan prima
89 adalah bilangan prima
97 adalah bilangan prima
```

Statemen Break

Kata kunci break digunakan untuk menghentikan eksekusi perulangan.

Berikut ini contoh penggunaannya:

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

int main ()
{
 // Deklarasi variabel lokal:
 int a = 10;

 // eksekusi perulangan do
 do
 {
 cout << "nilai a: " << a << endl;
 a = a + 1;

 if( a > 15)
 {
 // menghentikan perulangan
 break;
 }
 }
 while( a < 20 );

 return 0;
}
```

Berikut ini output dari kode tersebut:

```
nilai a: 10
nilai a: 11
nilai a: 12
nilai a: 13
nilai a: 14
nilai a: 15
```

Kata Kunci Continue

Kata kunci continue digunakan untuk melompat pada iterasi selanjutnya pada perulangan.

- Pada perulangan for, kata kunci menyebabkan aliran proses melompat langsung pada bagian pertambahan.
- Pada perulangan while, aliran proses melompat langsung pada bagian kondisi.

Berikut ini contoh penggunaannya:

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

int main()
{
 // Deklarasi variabel lokal:
 int a = 10;

 // eksekusi perulangan do
 do
 {
 if( a == 15)
 {
 // melewatkan iterasi.
 a = a + 1;
 continue;
 }

 cout << "nilai a: " << a << endl;
 a = a + 1;
 }
 while( a < 20 );

 return 0;
}
```

Kode tersebut akan menghasilkan output berikut:

```
nilai a: 10
nilai a: 11
nilai a: 12
nilai a: 13
nilai a: 14
nilai a: 16
nilai a: 17
nilai a: 18
nilai a: 19
```

BAB 11

Pengambilan Keputusan dalam C++

Statemen if

Statemen if terdiri dari sebuah ekspresi Boolean yang diikuti dengan satu statemen atau lebih.

Berikut ini sintaks untuk statemen if:

```
if(ekspresi_boolean)
{
 // Statemen akan dieksekusi jika ekspresi Boolean
 // bernilai benar
}
```

Berikut ini contoh penggunaannya:

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

int main()
{
 // deklarasi variabel lokal:
 int a = 10;

 // memeriksa kondisi boolean
 if( a < 20 )
 {
 // jika kondisi benar tampilkan baris berikut
 cout << "a kurang dari 20;" << endl;
 }

 cout << "nilai a adalah : " << a << endl;
```

```
 return 0;
}
```

Kode tersebut akan menghasilkan output berikut:

```
a kurang dari 20
nilai a adalah : 10
```

Statemen if…else

Statemen if dapat diikuti dengan statemen else opsional, yang dieksekusi jika ekspresi Boolean bernilai salah.

Berikut ini sintaksnya:

```
if(ekspresi_boolean)
{
 // Dieksekusi jika ekspresi Boolean bernilai benar
}
else
{
 // Dieksekusi jika ekspresi Boolean bernilai salah
}
```

Berikut ini contoh penggunaannya:

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

int main()
{
 // deklarasi variabel lokal:
 int a = 100;

 // memeriksa kondisi boolean
 if( a < 20 )
 {
 // jika kondisi benar tampilkan baris berikut
 cout << "a kurang dari 20;" << endl;
 }
 else
 {
 // jika kondisi tidak benar tampilkan baris berikut
 cout << "a tidak kurang dari 20;" << endl;
 }

 cout << "nilai a adalah : " << a << endl;

 return 0;
}
```

Kode tersebut akan menghasilkan output berikut:

```
a tidak kurang dari 20
nilai a adalah : 100
```

Statemen if…else if…else

Statemen if dapat diikuti dengan statemen else if…else opsional.

Saat menggunakan statemen if, else if, else ada beberapa hal yang harus Anda perhatikan:

- Sebuah statemen if dapat memiliki nol atau satu statemen else dan harus digunakan setelah statemen else if.
- Sebuah statemen if dapat memiliki nol atau banyak statemen else if dan harus digunakan sebelum statemen else.
- Setelah sebuah statemen else if dieksekusi, statemen else if atau else yang lain tidak akan diperiksa (akan langsung melompat pada kode setelah else terakhir).

Berikut sintaksnya:

```
if(ekspresi_boolean1)
{
 // Dieksekusi jika ekspresi boolean 1 bernilai benar
}
else if(ekspresi_boolean2)
{
 // Dieksekusi jika ekspresi boolean 2 bernilai benar
}
else if(ekspresi_boolean3)
{
 // Dieksekusi jika ekspresi boolean 3 bernilai benar
}
else
{
 // Dieksekusi jika tidak ada kondisi di atas yang
 // bernilai benar.
}
```

Berikut ini contoh penggunaannya:

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

int main ()
{
 // deklarasi variabel lokal:
```

```
 int a = 100;

 // memeriksa kondisi boolean
 if( a == 10 )
 {
 // jika kondisi benar tampilkan baris berikut
 cout << "Nilai a adalah 10" << endl;
 }
 else if( a == 20 )
 {
 // jika kondisi else if bernilai benar
 cout << "Nilai a adalah 20" << endl;
 }
 else if( a == 30 )
 {
 // jika kondisi else if bernilai benar
 cout << "Nilai a adalah 30" << endl;
 }
 else
 {
 // jika tidak ada kondisi di atas yang bernilai benar
 cout << "Tidak ada nilai yang cocok" << endl;
 }

 cout << "Nilai a adalah : " << a << endl;

 return 0;
}
```

Kode tersebut akan menghasilkan output berikut:

```
Tidak ada nilai yang cocok
Nilai a adalah: 100
```

Statemen if…else Bersarang

Anda dapat memasukkan statemen if else ke dalam statemen if else lainnya.

Berikut ini sintaks dari statemen if else bersarang:

```
if(ekspresi_boolean1)
{
 // Dieksekusi jika ekspresi boolean 1 bernilai benar
 if(ekspresi_boolean2)
 {
 // Dieksekusi jika ekspresi boolean 2 bernilai benar
 }
}
```

Berikut ini contoh penggunaannya:

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

int main()
{
 // deklarasi variabel lokal:
 int a = 100;
 int b = 200;

 // memeriksa kondisi boolean
 if( a == 100 )
 {
 // jika kondisi benar, periksa kondisi berikut
 if( b == 200 )
 {
 // jika kondisi benar, tampilkan baris berikut
 cout << "Nilai a 100 dan b 200" << endl;
 }
 }

 cout << "Nilai a adalah : " << a << endl;
 cout << "Nilai b adalah : " << b << endl;

 return 0;
}
```

Hasil dari kode tersebut adalah sebagai berikut:

```
Nilai a 100 dan b 200
Nilai a adalah : 100
Nilai b adalah : 200
```

Statemen Switch

Statemen switch memungkinkan sebuah variabel diperiksa kesamaannya dengan daftar nilai-nilai yang ada. Setiap nilai dalam daftar disebut case. Berikut ini sintaks statemen switch:

```
switch(expression)
{
 case nilai :
 // Statemen
 break; // opsional
 case nilai :
 // Statemen
 break; // opsional

 // Anda dapat menggunakan beberapa statemen case.
 default: // opsional
 // Statemen
}
```

Berikut ini aturan penggunaan statemen switch:

- Variabel yang digunakan dalam switch hanya boleh berupa byte, short, int, atau char.
- Anda dapat menggunakan beberapa case di dalam switch. Setiap case diikuti dengan nilai yang akan dibandingkan dan tanda titik dua (:).
- Nilai case harus sama tipe datanya dengan variabel dalam switch dan harus berupa konstanta atau literal.
- Ketika variabel yang dibandingkan sama dengan dengan case, statemen pada case tersebut akan dieksekusi sampai pada kata kunci break.
- Ketika sampai pada kata kunci break, switch berhenti dieksekusi, dan aliran proses melompat pada kode sesudah statemen switch.
- Tidak semua case memerlukan break. Jika tidak ada break, aliran proses akan terus dilanjutkan pada case selanjutnya sampai pada break.
- Statemen switch dapat memiliki case default, yang diletakkan pada akhir statemen. Default dijalankan jika tidak ada case yang dijalankan.

Berikut ini contoh penggunaannya:

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

int main()
{
 // deklarasi variabel lokal:
 char nilai = 'D';

 switch(nilai)
 {
 case 'A' :
 cout << "Sempurna!" << endl;
 break;
 case 'B' :
 case 'C' :
 cout << "Bagus" << endl;
 break;
 case 'D' :
 cout << "Anda lulus" << endl;
 break;
 case 'F' :
 cout << "Coba lagi" << endl;
 break;
```

```
 default :
 cout << "Nilai tidak valid" << endl;
 }

 cout << "Nilai Anda adalah " << nilai << endl;

 return 0;
}
```

Kode tersebut menghasilkan output sebagai berikut:

```
Anda lulus
Nilai Anda adalah D
```

Statemen Switch Bersarang

Anda dapat memasukkan statemen switch di dalam statemen switch lain.

Berikut ini sintaks untuk statemen switch bersarang:

```
switch(ch1)
{
 case 'A':
 cout << "Bagian A dari switch luar";
 switch(ch2)
 {
 case 'A':
 cout << "Bagian A dari switch dalam";
 break;
 case 'B': /* kode untuk case B dari switch dalam */
 }
 break;
 case 'B': /* kode untuk case B dari switch luar */
}
```

Berikut ini contoh penggunaannya:

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

int main()
{
 // deklarasi variabel lokal:
 int a = 100;
 int b = 200;

 switch(a)
 {
 case 100:
 cout << "Ini bagian switch luar" << endl;
 switch(b)
 {
 case 200:
```

```
    cout << "Ini bagian switch dalam" << endl;
    }
    }

    cout << "Nilai a adalah : " << a << endl;
    cout << "Nilai b adalah : " << b << endl;

    return 0;
}
```

Kode tersebut akan menghasilkan output berikut:

```
Ini bagian switch luar
Ini bagian switch dalam
Nilai a adalah : 100
Nilai b adalah : 200
```

Operator ? :

Operator ini juga disebut operator ternary. Operator ini terdiri dari tiga operand dan digunakan untuk mengevaluasi ekspresi Boolean.

Berikut ini syntax penulisan operator ternary:

```
Ekspresi1 ? Ekspresi2 : Ekspresi3
```

Urutan prosesnya adalah sebagai berikut:

1. Ekspresi1 dievaluasi.
2. Jika bernilai benar, Ekspresi2 dievaluasi dan menjadi nilai keluaran (perintah yang dieksekusi).
3. Jika bernilai salah, Ekspresi3 dievaluasi dan menjadi nilai keluaran (perintah yang dieksekusi).

BAB 12

Fungsi dalam C++

Fungsi adalah sekumpulan statemen yang secara bersama-sama melakukan suatu tugas. Setiap program C++ memiliki minimal satu fungsi main(). Fungsi juga sering disebut method, sub-rutin, prosedur, dan sebagainya.

Mendefinisikan Fungsi

Saat mendefinisikan fungsi, Anda dasarnya mendeklarasikan elemen-elemen dari strukturnya.

Berikut sintaks untuk mendefinisikan fungsi dalam C++:

```
tipe_balikan nama_fungsi ( daftar parameter )
{
 tubuh fungsi
}
```

Berikut penjelasan masing-masing elemen:

- tipe_balikan: tipe data dari nilai yang dikembalikan oleh fungsi. Jika fungsi tidak mengembalikan nilai maka tipe balikannya void.
- nama_fungsi: nama fungsi merupakan identifier yang bersifat unik dan case sensitive.

- daftar parameter: parameter dalam tanda kurung digunakan untuk memberikan dan menerima data dari fungsi. Parameter bersifat opsional sehingga fungsi tidak harus memiliki daftar parameter.
- tubuh fungsi: mengandung instruksi yang dibutuhkan untuk menyelesaikan tugas tertentu.

Berikut ini contoh penggunaan fungsi maks() yang menggunakan dua nilai integer dan mengembalikan nilai terbesar di antaranya.

```
// fungsi mengembalikan nilai maksimum di antara dua angka

int maks(int num1, int num2)
{
 // deklarasi variabel lokal
 int hasil;

 if (num1 > num2)
 hasil = num1;
 else
 hasil = num2;

 return hasil;
}
```

Memanggil Fungsi

Anda dapat memanggil fungsi dengan menggunakan nama fungsi dan memberikan parameter yang sesuai (jika fungsi memiliki parameter).

Berikut ini contohnya:

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

// deklarasi fungsi
int maks(int num1, int num2);

int main()
{
 // deklarasi variabel lokal:
 int a = 100;
 int b = 200;
 int ret;

 // memanggil fungsi untuk mendapatkan nilai maksimum
 ret = maks(a, b);

 cout << "Nilai maksimum adalah : " << ret << endl;

 return 0;
}
```

```
// fungsi mengembalikan nilai maksimum di antara dua angka
int maks(int num1, int num2)
{
 // deklarasi variabel lokal
 int hasil;

 if (num1 > num2)
 hasil = num1;
 else
 hasil = num2;

 return hasil;
}
```

Kode tersebut akan menghasilkan output berikut:

```
Nilai maksimum adalah : 200
```

Argumen Fungsi

Jika sebuah fungsi menggunakan argumen/parameter, fungsi tersebut harus mendeklarasikan variabel yang akan menampung nilai argumen. Variabel ini disebut parameter formal dari fungsi.

Dalam pemanggilan fungsi, ada beberapa cara untuk memberikan nilai argumen ke dalam fungsi.

Pemanggilan Berdasarkan Nilai

Metode pemanggilan fungsi berdasarkan nilai memberikan salinan nilai argumen ke dalam parameter formal dari fungsi. Perubahan pada parameter di dalam fungsi tidak akan berpengaruh pada argumen.

Berikut ini contoh fungsi swap():

```
// definisi fungsi swap untuk menukar nilai.
void swap(int x, int y)
{
 int temp;
 temp = x; /* menyimpan nilai x */
 x = y; /* memasukkan y ke dalam x */
 y = temp; /* memasukkan x ke dalam y */

 return;
}
```

Berikut ini contoh pemanggilan fungsi swap() dengan memberikan nilai aktual:

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

// deklarasi fungsi
void swap(int x, int y);

int main()
{
 // deklarasi variabel lokal:
 int a = 100;
 int b = 200;

 cout << "Sebelum swap, nilai a :" << a << endl;
 cout << "Sebelum swap, nilai b :" << b << endl;

 // memanggil fungsi swap.
 swap(a, b);

 cout << "Setelah swap, nilai a :" << a << endl;
 cout << "Setelah swap, nilai b :" << b << endl;

 return 0;
}
```

Kode tersebut akan menghasilkan output berikut:

```
Sebelum swap, nilai a :100
Sebelum swap, nilai b :200
Setelah swap, nilai a :100
Setelah swap, nilai b :200
```


Pemanggilan Berdasarkan Pointer

Metode pemanggilan fungsi berdasarkan pointer memberikan salinan alamat dari argumen ke dalam parameter formal dari fungsi. Perubahan terhadap parameter di dalam fungsi akan berpengaruh pada argumen aslinya. Berikut ini contoh fungsi swap():

```
// definisi fungsi swap untuk menukar nilai.
void swap(int *x, int *y)
{
 int temp;
 temp = *x; /* menyimpan nilai pada alamat x */
 *x = *y; /* memasukkan y ke dalam x */
 *y = temp; /* memasukkan x ke dalam y */

 return;
}
```

Berikut ini contoh pemanggilan fungsi swap() dengan menggunakan pointer:

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

// deklarasi fungsi
void swap(int *x, int *y);

int main ()
{
 // deklarasi variabel lokal:
 int a = 100;
 int b = 200;

 cout << "Sebelum swap, nilai a :" << a << endl;
 cout << "Sebelum swap, nilai b :" << b << endl;

 /* memanggil fungsi untuk menukar nilai.
 * &a mengindikasikan pointer a (alamat variabel a) dan
 * &b mengindikasikan pointer b (alamat variabel b).
 */
 swap(&a, &b);

 cout << "Setelah swap, nilai a :" << a << endl;
 cout << "Setelah swap, nilai b :" << b << endl;

 return 0;
}
```

Berikut output dari kode tersebut:

```
Sebelum swap, nilai a :100
Sebelum swap, nilai b :200
Setelah swap, nilai a :200
Setelah swap, nilai b :100
```

Pemanggilan Berdasarkan Referensi

Metode pemanggilan fungsi berdasarkan referensi memberikan salinan referensi dari argumen ke dalam parameter formal dari fungsi. Perubahan terhadap parameter di dalam fungsi akan berpengaruh pada argumen aslinya. Berikut ini contoh fungsi swap():

```
// definisi fungsi swap untuk menukar nilai.
void swap(int &x, int &y)
{
 int temp;
 temp = x; /* menyimpan nilai pada alamat x */
 x = y; /* memasukkan y ke dalam x */
 y = temp; /* memasukkan x ke dalam y */

 return;
}
```

Berikut ini contoh pemanggilan fungsi swap() dengan menggunakan referensi:

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

// deklarasi fungsi
void swap(int &x, int &y);

int main ()
{
 // deklarasi variabel lokal:
 int a = 100;
 int b = 200;

 cout << "Sebelum swap, nilai a :" << a << endl;
 cout << "Sebelum swap, nilai b :" << b << endl;

 /* memanggil fungsi untuk menukar nilai menggunakan
 * referensi variabel.
 */
 swap(a, b);

 cout << "Setelah swap, nilai a :" << a << endl;
 cout << "Setelah swap, nilai b :" << b << endl;

 return 0;
}
```

Berikut output dari kode tersebut:

```
Sebelum swap, nilai a :100
Sebelum swap, nilai b :200
Setelah swap, nilai a :200
Setelah swap, nilai b :100
```

BAB 13

Array dalam C++

C++ menyediakan sebuah struktur data, array, yang menampung koleksi elemen-elemen dengan tipe yang sama. Array dapat digambarkan sebagai koleksi variabel-variabel dengan tipe yang sama.

Deklarasi Array

Untuk mendeklarasi array dalam C++, Anda perlu menspesifikasikan tipe elemen dan jumlah elemen yang akan ditampung dalam array.

Berikut ini sintaksnya:

```
tipe namaArray [ ukuranArray ];
```

Array tersebut merupakan array satu dimensi. Ukuran Array harus merupakan konstanta integer yang lebih dari nol dan tipe dapat berupa tipe data dalam C++.

Berikut ini contoh untuk mendeklarasikan array 10 elemen dengan nama daftarKu dengan tipe double:

```
double daftarKu[10];
```

Inisialisasi Array

Anda dapat melakukan inisialisasi elemen array satu per satu atau dengan menggunakan contoh statemen berikut:

```
double daftarKu[5] = {1000.0, 2.0, 3.4, 17.0, 50.0 };
```

Jumlah nilai di antara tanda { } tidak bisa lebih dari jumlah elemen yang dideklarasikan.

Jika Anda tidak menspesifikasikan jumlah elemen array maka ukuran array akan secara otomatis mengikuti jumlah elemen yang diberikan. Berikut contohnya:

```
double daftarKu = {1000.0, 2.0, 3.4, 17.0, 50.0 };
```

Mengakses Elemen Array

Untuk mengakses elemen array, Anda dapat menggunakan indeks elemen. Berikut contoh untuk mengakses nilai elemen pada posisi indeks 9 (elemen ke-10) dan memasukkannya pada variabel:

```
double gaji = saldo[9];
```

Berikut ini contoh deklarasi, inisialisasi, memasukkan nilai dan mengakses array:

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

#include <iomanip>
using std::setw;

int main ()
{
 int n[ 10 ]; // n merupakan array yang berisi 10 integer

 // menginisialisasi semua elemen array n
 for ( int i = 0; i < 10; i++ )
 {
 n[ i ] = i + 100; // mengatur nilai elemen pada posisi i
 }

 cout << "Elemen" << setw( 12 ) << "Nilai" << endl;

 // menampilkan nilai setiap elemen array
 for ( int j = 0; j < 10; j++ )
 {
```

```
 cout << setw( 6 )<< j << setw( 12 ) << n[ j ] << endl;
 }

 return 0;
}
```

Kode tersebut akan menghasilkan output sebagai berikut:

```
Elemen Nilai
 0 100
 1 101
 2 102
 3 103
 4 104
 5 105
 6 106
 7 107
 8 108
 9 109
```

Memasukkan Array ke dalam Fungsi

C++ tidak memperbolehkan Anda untuk memasukkan sebuah array sebagai argumen dalam fungsi. Anda dapat memasukkan pointer dari sebuah array dengan menspesifikasikan nama array tanpa indeks.

Jika Anda ingin memasukkan array satu dimensi sebagai argumen dalam fungsi, Anda harus mendeklarasikan parameter formal dari fungsi dengan salah satu dari tiga cara berikut:

Cara 1

Parameter formal sebagai pointer:

```
void fungsiKu(int *param)
{
 .....
 .....
 .....
}
```

Cara 2

Parameter formal sebagai array dengan ukuran:

```
void fungsiKu(int param[10])
{
 .....
 .....
 .....
}
```

Cara 3

Parameter formal sebagai array tanpa ukuran:

```
void fungsiKu(int param[])
{
 .....
 .....
 .....
}
```

Berikut ini contoh fungsi yang akan menggunakan array sebagai argumen dan akan mengembalikan nilai rata-rata:

```
double getRataRata(int arr[], int ukuran)
{
 int i, jum = 0;
 double rata;

 for (i = 0; i < ukuran; ++i)
 {
 jum += arr[i];
 }

 rata = double(jum) / ukuran;

 return rata;
}
```

Berikut ini contoh pemanggilan fungsi tersebut:

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

// deklarasi fungsi function declaration:
double getRataRata(int arr[], int ukuran);

int main ()
{
 // array dengan 5 elemen integer.
 int saldo[5] = {1000, 2, 3, 17, 50};
```

```
 double rata;

 // memberikan pointer array sebagai argumen.
 rata = getRataRata( saldo, 5 ) ;

 // menampilkan nilai balikan
 cout << "Nilai rata-rata: " << rata << endl;

 return 0;
}
```

Kode tersebut akan menghasilkan output berikut:

```
Nilai rata-rata: 214.4
```

Mengembalikan Array dari Fungsi

C++ tidak memperbolehkan Anda untuk mengembalikan sebuah array sebagai nilai balikan dari fungsi. Anda dapat mengembalikan pointer dari sebuah array dengan menspesifikasikan nama array tanpa indeks.

Jika Anda ingin mengembalikan array satu dimensi dari fungsi, Anda harus mendeklarasikan fungsi yang mengembalikan pointer seperti contoh berikut:

```
void * fungsiKu()
{
 .....
 .....
 .....
}
```

Berikut ini contoh fungsi yang akan membuat 10 angka secara acak dan mengembalikan nilai-nilai tersebut sebagai array:

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

// fungsi untuk membuat dan mengembalikan nilai secara acak.
int * getAcak( )
{
 static int r[10];

 // mengatur seed
 srand( (unsigned)time( NULL ) );

 for (int i = 0; i < 10; ++i)
 {
 r[i] = rand();
 cout << r[i] << endl;
```

```
 }

 return r;
}

// fungsi main untuk memanggil fungsi di atas.
int main ()
{
 // pointer integer
 int *p;
 p = getAcak();

 for ( int i = 0; i < 10; i++ )
 {
 cout << "*(p + " << i << ") : ";
 cout << *(p + i) << endl;
 }

 return 0;
}
```

Kode tersebut akan menghasilkan output seperti pada contoh berikut:

```
624723190
1468735695
807113585
976495677
613357504
1377296355
1530315259
1778906708
1820354158
667126415
*(p + 0) : 624723190
*(p + 1) : 1468735695
*(p + 2) : 807113585
*(p + 3) : 976495677
*(p + 4) : 613357504
*(p + 5) : 1377296355
*(p + 6) : 1530315259
*(p + 7) : 1778906708
*(p + 8) : 1820354158
*(p + 9) : 667126415
```

BAB 14

String dalam C++

C++ menyediakan dua tipe representasi string:

- Karakter string C-style.
- Tipe kelas string dalam C++.

Karakter String C-style

Karakter string C-style digunakan pada bahasa pemrograman C dan juga didukung dalam C++. String ini merupakan array satu dimensi dengan tipe karakter yang diakhiri dengan karakter null '\0'.

Berikut contoh deklarasi dan inisialisasi string "Halo":

```
char salam[5] = {'H', 'a', 'l', 'o', '\0'};
```

Anda dapat menuliskan statemen tersebut sebagai berikut:

```
char salam[] = "Halo";
```

Berikut ini contoh kode untuk menampilkan string di atas:

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

int main ()
{
  char salam[5] = {'H', 'a', 'l', 'o', '\0'};
  cout << "Salam: ";
  cout << salam << endl;

  return 0;
}
```

Kode tersebut akan menghasilkan output sebagai berikut:

```
Salam: Halo
```

Berikut ini fungsi-fungsi yang dapat Anda gunakan untuk memanipulasi string berakhiran null:

No.	Fungsi & Kegunaan
1	**strcpy(s1, s2);** Menyalin string s2 ke dalam string s1.
2	**strcat(s1, s2);** Menambahkan string s2 di bagian akhir string s1.
3	**strlen(s1);** Mengembalikan panjang string s1.
4	**strcmp(s1, s2);** Mengembalikan 0 jika s1 dan s2 sama; kurang dari 0 jika s1 < s2; lebih dari 0 jika s1 > s2.
5	**strchr(s1, ch);** Mengembalikan sebuah pointer dari karakter ch pertama yang ada dalam string s1.
6	**strstr(s1, s2);** Mengembalikan sebuah pointer dari string s2 pertama yang ada dalam string s1.

Fungsi untuk memanipulasi string

Berikut ini contoh penggunaan beberapa fungsi di atas:

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

int main ()
```

```
{
  char str1[10] = "Hello";
  char str2[10] = "World";
  char str3[10];
  int len ;

  // menyalin str1 ke dalam str3
  strcpy( str3, str1);
  cout << "strcpy( str3, str1) : " << str3 << endl;

  // menyambung str1 dan str2
  strcat( str1, str2);
  cout << "strcat( str1, str2): " << str1 << endl;

  // panjang total dari str1 setelah disambung
  len = strlen(str1);
  cout << "strlen(str1) : " << len << endl;

  return 0;
}
```

Kode tersebut akan menghasilkan output berikut:

```
strcpy( str3, str1) : Hello
strcat( str1, str2): HelloWorld
strlen(str1) : 10
```

Kelas String dalam C++

Library standar C++ menyediakan tipe kelas string yang mendukung semua operasi di atas ditambah dengan banyak fungsionalitas lainnya.

Berikut ini contoh penggunaannya:

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

int main ()
{
  string str1 = "Hello";
  string str2 = "World";
  string str3;
  int len ;

  // menyalin str1 ke dalam str3
  str3 = str1;
  cout << "str3 : " << str3 << endl;

  // menyambung str1 dan str2
  str3 = str1 + str2;
  cout << "str1 + str2 : " << str3 << endl;

  // panjang total dari str3 setelah disambung
  len = str3.size();
```

```
    cout << "str3.size() : " << len << endl;

    return 0;
}
```

Kode tersebut akan menghasilkan output berikut:

```
str3 : Hello
str1 + str2 : HelloWorld
str3.size() : 10
```

BAB 15

Pointer dalam C++

Dalam C++, setiap variabel merupakan lokasi memori dan setiap lokasi memori memiliki alamat yang dapat diakses menggunakan operator &.

Berikut contoh kode untuk menampilkan alamat variabel:

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

int main ()
{
  int var1;
  char var2[10];

  cout << "Alamat variabel var1: ";
  cout << &var1 << endl;

  cout << "Alamat variabel var2: ";
  cout << &var2 << endl;

  return 0;
}
```

Kode tersebut akan menghasilkan output seperti pada contoh berikut:

```
Alamat variabel var1: 0xbfebd5c0
Alamat variabel var2: 0xbfebd5b6
```

Apa Itu Pointer?

Pointer adalah variabel yang nilainya merupakan alamat dari variabel lain. Untuk menggunakan pointer, Anda perlu mendeklarasikannya terlebih dahulu. Berikut sintaksnya:

```
tipe *nama-var;
```

Pada sintaks di atas, tipe merupakan tipe pointer yang valid dalam C++ dan nama-var merupakan nama dari variabel pointer. Tanda * yang digunakan merupakan penanda bahwa sebuah variabel merupakan pointer.

Berikut ini contoh deklarasi pointer:

```
int *ip; // pointer untuk sebuah variabel integer
double *dp; // pointer untuk sebuah variabel double
float *fp; // pointer untuk sebuah variabel float
char *ch // pointer untuk sebuah variabel karakter
```

Menggunakan Pointer

Ada beberapa operasi penting yang perlu Anda lakukan dengan pointer:

- Mendefiniskan variabel pointer.
- Memasukkan alamat dari variabel ke dalam pointer.
- Mengakses nilai pada alamat yang ada dalam variabel pointer.

Operasi di atas dilakukan dengan menggunakan operator * yang mengembalikan nilai dari variabel yang ada pada alamat operan.

Berikut contoh penggunaannya:

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

int main ()
{
  int var = 20; // deklarasi variabel aktual.
  int *ip; // variabel pointer
  ip = &var; // memasukkan alamat dari var ke dalam variabel
             // pointer

  cout << "Nilai variabel var: ";
  cout << var << endl;
```

```
  // menampilkan alamat dalam variabel pointer ip
  cout << "Alamat yang disimpan dalam variabel ip: ";
  cout << ip << endl;

  // mengakses nilai dalam alamat yang ada pada pointer
  cout << "Nilai variabel *ip: ";
  cout << *ip << endl;

  return 0;
}
```

Kode tersebut akan menghasilkan output seperti contoh berikut:

```
Nilai variabel var: 20
Alamat yang disimpan dalam variabel ip: 0xbfc601ac
Nilai variabel *ip: 20
```

Pointer Null

Pointer NULL merupakan konstanta dengan nilai nol yang ada pada beberapa library, termasuk iostream.

Berikut contohnya:

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

int main ()
{
  int *ptr = NULL;

  cout << "Nilai ptr adalah " << ptr ;

  return 0;
}
```

Kode tersebut akan menghasilkan output berikut:

```
Nilai ptr adalah 0
```

Untuk memeriksa pointer null Anda dapat menggunakan statemen if seperti pada contoh berikut:

```
if(ptr)  // berhasil jika p tidak null
if(!ptr) // berhasil jika p null
```

Aritmatika Pointer

Ada empat operator aritmatika yang dapat digunakan pada pointer: ++, --, +, dan -.

Pertambahan

Berikut contoh kode yang melakukan operasi pertambahan pada pointer variabel:

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

const int MAX = 3;

int main ()
{
  int var[MAX] = {10, 100, 200};
  int *ptr;

  // mamasukkan alamat array dalam pointer.
  ptr = var;

  for (int i = 0; i < MAX; i++)
  {
    cout << "Alamat var[" << i << "] = ";
    cout << ptr << endl;

    cout << "Nilai var[" << i << "] = ";
    cout << *ptr << endl;

    // menunjuk pada lokasi selanjutnya
    ptr++;
  }

  return 0;
}
```

Kode tersebut akan menghasilkan contoh output sebagai berikut:

```
Alamat var[0] = 0xbfa088b0
Nilai var[0] = 10
Alamat var[1] = 0xbfa088b4
Nilai var[1] = 100
Alamat var[2] = 0xbfa088b8
Nilai var[2] = 200
```

Pengurangan

Berikut contoh kode yang melakukan operasi pertambahan pada pointer variabel:

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

const int MAX = 3;

int main ()
{
  int var[MAX] = {10, 100, 200};
  int *ptr;

  // mamasukkan alamat elemen terakhir ke dalam pointer.
  ptr = &var[MAX-1];

  for (int i = MAX; i > 0; i--)
  {
    cout << "Alamat var[" << i << "] = ";
    cout << ptr << endl;

    cout << "Nilai var[" << i << "] = ";
    cout << *ptr << endl;

    // menunjuk pada lokasi sebelumnya
    ptr--;
  }

  return 0;
}
```

Kode tersebut akan menghasilkan contoh output sebagai berikut:

```
Alamat var[3] = 0xbfdb70f8
Nilai var[3] = 200
Alamat var[2] = 0xbfdb70f4
Nilai var[2] = 100
Alamat var[1] = 0xbfdb70f0
Nilai var[1] = 10
```

Perbandingan Pointer

Pointer dapat dibandingkan dengan menggunakan operator perbandingan, seperti ==, < dan >.

Program berikut memodifikasi contoh sebelumnya:

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

const int MAX = 3;
```

```
int main ()
{
  int var[MAX] = {10, 100, 200};
  int *ptr;

  // mamasukkan alamat elemen pertama ke dalam pointer.
  ptr = var;
  int i = 0;

  while ( ptr <= &var[MAX - 1] )
  {
    cout << "Alamat var[" << i << "] = ";
    cout << ptr << endl;

    cout << "Nilai var[" << i << "] = ";
    cout << *ptr << endl;

    // menunjuk pada lokasi sebelumnya
    ptr++;
    i++;
  }

  return 0;
}
```

Kode tersebut akan menghasilkan output berikut:

```
Alamat var[0] = 0xbfce42d0
Nilai var[0] = 10
Alamat var[1] = 0xbfce42d4
Nilai var[1] = 100
Alamat var[2] = 0xbfce42d8
Nilai var[2] = 200
```

Memasukkan Pointer ke dalam Fungsi

Berikut ini contoh memasukkan pointer ke dalam fungsi dan mengubah nilai di dalam fungsi:

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

void getDetik(unsigned long *par);

int main ()
{
  unsigned long sec;

  getDetik( &sec );

  // menampilkan nilai aktual
  cout << "Jumlah detik :" << sec << endl;

  return 0;
```

```
}

void getDetik(unsigned long *par)
{
  // mendapatkan jumlah detik yang ada
  *par = time( NULL );

  return;
}
```

Kode tersebut akan menghasilkan output berikut:

```
Jumlah detik :1294450468
```

Mengembalikan Pointer dari Fungsi

Dalam C++, Anda dapat mengembalikan sebuah pointer sebagai nilai balikan dari fungsi. Berikut ini sintaksnya:

```
int * fungsiKu()
{
  .....
  .....
  .....
}
```

Berikut ini contoh kode yang membuat 10 angka secara acak dan mengembalikannya dengan menggunakan nama array yang merepresentasikan alamat pointer dari elemen array pertama:

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

// fungsi untuk membuat dan mengembalikan angka acak.
int * getAcak( )
{
  static int r[10];

  // mengatur seed
  srand( (unsigned)time( NULL ) );

  for (int i = 0; i < 10; ++i)
  {
    r[i] = rand();
    cout << r[i] << endl;
  }

  return r;
}
```

```
// fungsi main untuk memanggil fungsi di atas.
int main ()
{
  // pointer integer.
  int *p;

  p = getAcak();

  for ( int i = 0; i < 10; i++ )
  {
    cout << "*(p + " << i << ") : ";
    cout << *(p + i) << endl;
  }

  return 0;
}
```

Kode tersebut akan menghasilkan output seperti contoh berikut:

```
624723190
1468735695
807113585
976495677
613357504
1377296355
1530315259
1778906708
1820354158
667126415
*(p + 0) : 624723190
*(p + 1) : 1468735695
*(p + 2) : 807113585
*(p + 3) : 976495677
*(p + 4) : 613357504
*(p + 5) : 1377296355
*(p + 6) : 1530315259
*(p + 7) : 1778906708
*(p + 8) : 1820354158
*(p + 9) : 667126415
```

BAB 16

Referensi dalam C++

Variabel referensi merupakan alias atau nama lain untuk variabel yang sudah ada. Setelah referensi diinisialisasi dengan sebuah variabel, Anda dapat mengakses variabel tersebut dengan nama variabel maupun nama referensi.

Referensi vs Pointer

Berikut ini perbedaan mendasar antara referensi dan pointer:

- Anda tidak dapat memiliki referensi NULL. Referensi harus terhubung dengan sebuah variabel.
- Setelah referensi diinisialisasi pada suatu objek, referensi tersebut tidak dapat diubah untuk menunjuk pada objek lain. Pointer dapat menunjuk pada objek lain kapanpun Anda melakukannya.
- Sebuah referensi harus diinisialisasi ketika dibuat. Pointer dapat diinisialisasi kapanpun setelah dibuat.

Membuat Referensi

Berikut ini contoh variabel i:

```
int i = 17;
```

Anda kemudian dapat mendeklarasikan variabel referensi untuk i sebagai berikut:

```
int &r = i;
```

Berikut ini contoh penggunaannya:

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

int main ()
{
  // mendeklarasikan variabel
  int i;
  double d;

  // mendeklarasikan variabel referensi
  int& r = i;
  double& s = d;

  i = 5;
  cout << "Nilai i : " << i << endl;
  cout << "Nilai referensi i : " << r << endl;

  d = 11.7;
  cout << "Nilai d : " << d << endl;
  cout << "Nilai referensi d : " << s << endl;

  return 0;
}
```

Kode tersebut akan menghasilkan output berikut:

```
Nilai i : 5
Nilai referensi i : 5
Nilai d : 11.7
Nilai referensi d : 11.7
```

Referensi biasanya digunakan sebagai argumen dalam fungsi dan nilai balikan dari fungsi.

Referensi sebagai Parameter

Berikut ini contoh konsep pemanggilan berdasarkan referensi:

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

// deklarasi fungsi
void swap(int& x, int& y);

int main ()
{
  // deklarasi variabel lokal:
  int a = 100;
  int b = 200;

  cout << "Sebelum swap, nilai a :" << a << endl;
  cout << "Sebelum swap, nilai b :" << b << endl;

  /* memanggil fungsi untuk menukar nilai.*/
  swap(a, b);

  cout << "Setelah swap, nilai a :" << a << endl;
  cout << "Setelah swap, nilai b :" << b << endl;

  return 0;
}

// definisi fungsi untuk menukar nilai.
void swap(int& x, int& y)
{
  int temp;
  temp = x; /* menyimpan nilai pada alamat x */
  x = y; /* memasukkan y ke dalam x */
  y = temp; /* memasukkan x ke dalam y */

  return;
}
```

Kode tersebut akan menghasilkan output berikut:

```
Sebelum swap, nilai a :100
Sebelum swap, nilai b :200
Setelah swap, nilai a :200
Setelah swap, nilai b :100
```

Referensi Sebagai Nilai Balikan

Ketika fungsi mengembalikan sebuah referensi, fungsi tersebut mengembalikan pointer implisit pada nilai balikannya.

Berikut contohnya:

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

double vals[] = {10.1, 12.6, 33.1, 24.1, 50.0};

double& setNilai( int i )
{
  return vals[i]; // mengembalikan referensi dari elemen ke-i
}

// fungsi main untuk memanggil fungsi di atas.
int main ()
{
  cout << "Nilai sebelum diubah" << endl;

  for ( int i = 0; i < 5; i++ )
  {
    cout << "vals[" << i << "] = ";
    cout << vals[i] << endl;
  }

  setNilai(1) = 20.23; // mengubah elemen ke-2
  setNilai(3) = 70.8; // mengubah elemen ke-4

  cout << "Nilai setelah diubah" << endl;

  for ( int i = 0; i < 5; i++ )
  {
    cout << "vals[" << i << "] = ";
    cout << vals[i] << endl;
  }

  return 0;
}
```

Kode tersebut akan menghasilkan output berikut:

```
Nilai sebelum diubah
vals[0] = 10.1
vals[1] = 12.6
vals[2] = 33.1
vals[3] = 24.1
vals[4] = 50
Nilai setelah diubah
vals[0] = 10.1
vals[1] = 20.23
vals[2] = 33.1
vals[3] = 70.8
vals[4] = 50
```

BAB 17

Kelas dan Objek dalam C++

Kelas digunakan untuk menspesifikasikan bentuk dari sebuah objek dan menggabungkan representasi data dan method untuk memanipulasi data tersebut. Data dan fungsi dalam kelas disebut member kelas.

Definisi Kelas

Definisi kelas dimulai dengan kata kunci class diikuti dengan nama kelas; dan tubuh kelas yang dibungkus dengan tanda { } dan diakhiri dengan tanda ;

Berikut ini contoh definisi kelas Kotak:

```
class Kotak
{
  public:
    double panjang; // Panjang kotak
    double lebar; // Lebar kotak
    double tinggi; // Tinggi kotak
};
```

Kata kunci public menentukan atribut akses dari member kelas yang mengikutinya. Member public dapat diakses dari luar kelas. Anda juga dapat mengatur atribut akses menjadi private atau protected.

Mendefinisikan Objek

Kelas menyediakan cetakan/blueprint dari objek, sehingga dasarnya sebuah objek dibuat dari sebuah kelas.

Berikut ini statemen yang mendeklarasikan dua objek dari kelas Kotak:

```
Kotak Kotak1; // mendeklarasikan Kotak1 dengan tipe Kotak
Kotak Kotak2; // mendeklarasikan Kotak2 dengan tipe Kotak
```

Mengakses Data Member

Data member public dari sebuah objek dapat diakses dengan menggunakan operator akses langsung (.)

Berikut contohnya:

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

class Kotak
{
  public:
    double panjang; // Panjang kotak
    double lebar; // Lebar kotak
    double tinggi; // Tinggi kotak
};

int main( )
{
  Kotak Kotak1; // mendeklarasikan Kotak1 dengan tipe Kotak
  Kotak Kotak2; // mendeklarasikan Kotak2 dengan tipe Kotak

  double volume = 0.0; // variabel volume kotak

  // spesifikasi kotak 1
  Kotak1.tinggi = 5.0;
  Kotak1.panjang = 6.0;
  Kotak1.lebar = 7.0;

  // spesifikasi kotak 2
  Kotak2.tinggi = 10.0;
  Kotak2.panjang = 12.0;
  Kotak2.lebar = 13.0;

  // volume kotak 1
  volume = Kotak1.tinggi * Kotak1.panjang * Kotak1.lebar;
  cout << "Volume Kotak1 : " << volume <<endl;

  // volume kotak 2
  volume = Kotak2.tinggi * Kotak2.panjang * Kotak2.lebar;
  cout << "Volume Kotak2 : " << volume <<endl;

  return 0;
}
```

Kode tersebut akan menghasilkan ouput berikut:

```
Volume Kotak1 : 210
Volume Kotak2 : 1560
```

Fungsi Member Kelas

Fungsi member kelas adalah fungsi yang definisinya atau prototype-nya berada dalam kelas tersebut. Fungsi ini beroperasi pada objek dari kelas, dan dapat mengakses semua member dari kelas tersebut.

Berikut contohnya:

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

class Kotak
{
  public:
    double panjang; // Panjang kotak
    double lebar; // Lebar kotak
    double tinggi; // Tinggi kotak

    // Deklarasi fungsi member
    double getVolume(void);
    void setPanjang( double pan );
    void setLebar( double leb );
    void setTinggi( double ting );
};

// Definisi fungsi member
double Kotak::getVolume(void)
{
  return panjang * lebar * tinggi;
}

void Kotak::setPanjang( double pan )
{
  panjang = pan;
}

void Kotak::setLebar( double leb )
{
  lebar = leb;
}

void Kotak::setTinggi( double ting )
{
  tinggi = ting;
}

// Fungsi main dari program
int main( )
{
  Kotak Kotak1; // mendeklarasikan Kotak1 dengan tipe Kotak
  Kotak Kotak2; // mendeklarasikan Kotak2 dengan tipe Kotak
```

```
    double volume = 0.0; // variabel volume kotak

    // spesifikasi kotak 1
    Kotak1.tinggi = 5.0;
    Kotak1.panjang = 6.0;
    Kotak1.lebar = 7.0;

    // spesifikasi kotak 2
    Kotak2.tinggi = 10.0;
    Kotak2.panjang = 12.0;
    Kotak2.lebar = 13.0;

    // volume kotak 1
    volume = Kotak1.getVolume();
    cout << "Volume Kotak1 : " << volume <<endl;

    // volume kotak 2
    volume = Kotak2.getVolume();
    cout << "Volume Kotak2 : " << volume <<endl;

    return 0;
}
```

Kode tersebut akan menghasilkan output berikut:

```
Volume Kotak1 : 210
Volume Kotak2 : 1560
```

Modifier Akses Kelas

Batasan akses terhadap member kelas dispesifikasikan dengan spesifier akses public, private, dan protected. Jika tidak dispesifikasikan, spesifier akses dari member dan kelas adalah private.

```
class Base
{
  public:
    // daftar member public
  protected:
    // daftar member protected
  private:
    // daftar member private
};
```

Member Public

Member public dapat diakses dari mana saja di luar kelas dalam suatu program. Berikut contohnya:

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

```
class Garis
{
  public:
    double panjang;
    void setPanjang( double pan );
    double getPanjang( void );
};

// definisi fungsi member
double Garis::getPanjang(void)
{
  return panjang ;
}

void Garis::setPanjang( double pan )
{
  panjang = pan;
}

// fungsi main
int main( )
{
  Garis garis;

  // mengatur panjang garis
  garis.setPanjang(6.0);
  cout << "Panjang garis : " << garis.getPanjang() <<endl;

  // mengatur panjang garis tanpa fungsi member
  garis.panjang = 10.0; // Bisa: karena panjang public
  cout << "Panjang garis : " << garis.panjang <<endl;

  return 0;
}
```

Kode tersebut akan menghasilkan output berikut:

```
Panjang garis : 6
Panjang garis : 10
```

Member Private

Member private tidak dapat diakses, atau ditampilkan dari luar kelas. Hanya kelas tersebut dan fungsi di dalamnya yang dapat mengakses member private.

Pada contoh berikut, lebar adalah member private karena modifier aksesnya tidak dituliskan:

```
class Kotak
{
  double lebar;

  public:
    double panjang;
```

```
    void setLebar( double leb );
    double getLebar( void );
};
```

Berikut contoh untuk definisi data sebagai private dan fungsi sebagai public untuk mengakses data:

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

class Kotak
{
  public:
    double panjang;
    void setLebar( double leb );
    double getLebar( void );

  private:
    double lebar;
};

// definisi fungsi member
double Kotak::getLebar(void)
{
  return lebar ;
}

void Kotak::setLebar( double leb )
{
  lebar = leb;
}

// Fungsi main
int main( )
{
  Kotak kotak;

  // mengatur panjang kotak tanpa fungsi member
  kotak.panjang = 10.0; // Bisa: karena panjang public
  cout << "Panjang kotak : " << kotak.panjang <<endl;

  // mengatur lebar kotak tanpa fungsi member
  // kotak.lebar = 10.0; // Error: karena lebar private

  kotak.setLebar(10.0); // Menggunakan fungsi member.
  cout << "Lebar kotak : " << kotak.getLebar() <<endl;

  return 0;
}
```

Kode tersebut akan menghasilkan output berikut:

```
Panjang kotak : 10
Lebar kotak : 10
```

Member Protected

Variabel member protected sama dengan member private tetapi menyediakan satu fitur tambahan yaitu dapat diakses oleh subkelas yang mewarisi kelas tersebut.

Pada contoh berikut subkelas KotakKecil mewarisi Kotak:

```
using namespace std;

class Kotak
{
  protected:
    double lebar;
};

class KotakKecil:Kotak // KotakKecil mewarisi Kotak.
{
  public:
    void setLebarKecil( double leb );
    double getLebarKecil( void );
};

// Fungsi member dari subkelas
double KotakKecil::getLebarKecil(void)
{
  return lebar ;
}

void KotakKecil::setLebarKecil( double leb )
{
  lebar = leb;
}

// Fungsi main
int main( )
{
  KotakKecil kotak;

  // mengatur lebar kotak dengan fungsi member
  kotak.setLebarKecil(5.0);
  cout << "Lebar kotak : "<< kotak.getLebarKecil() << endl;

  return 0;
}
```

Kode tersebut akan menghasilkan output berikut:

```
Lebar kotak : 5
```

Konstruktor dan Destruktor

Konstruktor Kelas

Konstruktor kelas merupakan member khusus dari kelas yang dieksekusi ketika objek dari kelas tersebut dibuat.

Konstruktor memiliki nama yang sama dengan kelas dan tidak memiliki tipe balikan.

Berikut ini contohnya:

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

class Garis
{
  public:
    void setPanjang( double pan );
    double getPanjang( void );
    Garis(); // Ini adalah konstruktor

  private:
    double panjang;
};

// definisi fungsi member dan konstruktor
Garis::Garis(void)
{
  cout << "Objek sementara dibuat" << endl;
}

void Garis::setPanjang( double pan )
{
  panjang = pan;
}

double Garis::getPanjang( void )
{
  return panjang;
}

// Fungsi main
int main( )
{
  Garis garis;

  // mengatur panjang garis
  garis.setPanjang(6.0);
  cout << "Panjang garis : " << garis.getPanjang() <<endl;

  return 0;
}
```

Kode tersebut akan menghasilkan output berikut:

```
Objek sementara dibuat
Panjang garis : 6
```

Konstruktor Berparameter

Konstruktor default tidak memiliki parameter, tetapi jika Anda dibutuhkan, Anda dapat membuat konstruktor berparameter.

Berikut ini contohnya:

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

class Garis
{
  public:
    void setPanjang( double pan );
    double getPanjang( void );
    Garis( double pan ); // Ini adalah konstruktor

  private:
    double panjang;
};

// Definisi fungsi member dan konstruktor
Garis::Garis( double pan )
{
  cout << "Objek sementara dibuat, panjang = " << pan << endl;
  panjang = pan;
}

void Garis::setPanjang( double pan )
{
  panjang = pan;
}

double Garis::getPanjang( void )
{
  return panjang;
}

// Fungsi main
int main( )
{
  Garis garis(10.0);

  // mengatur panjang garis
  cout << "Panjang garis : " << garis.getPanjang() <<endl;

  // mengatur kembali panjang garis
  garis.setPanjang(6.0);
  cout << "Panjang garis : " << garis.getPanjang() <<endl;

  return 0;
}
```

Kode tersebut akan menghasilkan output berikut:

```
Objek sementara dibuat, panjang = 10
Panjang garis : 10
Panjang garis : 6
```

Destruktor Kelas

Destruktor kelas merupakan member khusus dari kelas yang dieksekusi ketika objek dari kelas tersebut berada di luar lingkup atau dihapus.

Destruktor memiliki nama yang sama dengan kelas dan diawali dengan tanda ~. Destruktor tidak mengembalikan nilai dan tidak memiliki parameter.

Berikut ini contohnya:

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

class Garis
{
  public:
    void setPanjang( double pan );
    double getPanjang( void );
    Garis(); // Ini adalah konstruktor
    ~Garis(); // Ini adalah destruktor

  private:
    double panjang;
};

// definisi fungsi member, konstruktor, dan destruktor
Garis::Garis(void)
{
  cout << "Objek sementara dibuat" << endl;
}

Garis::~Garis(void)
{
  cout << "Objek sementara dihapus" << endl;
}

void Garis::setPanjang( double pan )
{
  panjang = pan;
}

double Garis::getPanjang( void )
{
  return panjang;
}

// Fungsi main
int main( )
{
```

```
  Garis garis;

  // mengatur panjang garis
  garis.setPanjang(6.0);
  cout << "Panjang garis : " << garis.getPanjang() <<endl;

  return 0;
}
```

Kode tersebut akan menghasilkan output berikut:

```
Objek sementara dibuat
Panjang garis : 6
Objek sementara dihapus
```

Pointer this

Setiap objek dalam C++ dapat mengakses alamatnya sendiri dengan pointer this. Pointer this merupakan parameter implisit bagi semua fungsi member.

Berikut contohnya:

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

class Kotak
{
  public:
    // Definisi konstruktor
    Kotak(double p=2.0, double l=2.0, double t=2.0)
    {
      cout <<"Konstruktor dipanggil." << endl;
      panjang = p;
      lebar = l;
      tinggi = t;
    }

    double Volume()
    {
      return panjang * lebar * tinggi;
    }

    int bandingkan(Kotak kotak)
    {
      return this->Volume() > kotak.Volume();
    }

  private:
    double panjang; // Panjang kotak
    double lebar; // Lebar kotak
    double tinggi; // Tinggi kotak
};

int main(void)
```

```
{
  Kotak Kotak1(3.3, 1.2, 1.5); // Deklarasi kotak1
  Kotak Kotak2(8.5, 6.0, 2.0); // Deklarasi kotak2

  if(Kotak1.bandingkan(Kotak2))
  {
    cout << "Kotak2 lebih kecil dari Kotak1" <<endl;
  }
  else
  {
    cout << "Kotak2 sama dengan atau lebih besar dari Kotak1"
         << endl;
  }

  return 0;
}
```

Kode tersebut akan menghasilkan output berikut:

```
Konstruktor dipanggil.
Konstruktor dipanggil.
Kotak2 sama dengan atau lebih besar dari Kotak1
```

Member Static dalam Kelas

Variabel static

Ketika Anda mendeklarasikan variabel member kelas dengan kata kunci static, variabel member tersebut hanya akan memiliki satu salinan saja, berapa pun objek yang dibuat.

Berikut ini contohnya:

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

class Kotak
{
  public:
    static int jumlahObjek;

    // Definisi konstruktor
    Kotak(double p=2.0, double l=2.0, double t=2.0)
    {
      cout <<"Konstruktor dipanggil." << endl;
      panjang = p;
      lebar = l;
      tinggi = t;

      // ditambahkan setiap kali objek dibuat
      jumlahObjek++;
    }
```

```
    double Volume()
    {
      return panjang * lebar * tinggi;
    }

  private:
    double panjang; // Panjang kotak
    double lebar; // Lebar kotak
    double tinggi; // Tinggi kotak
};

// Inisialisasi member static dari kelas Kotak
int Kotak::jumlahObjek = 0;

int main(void)
{
  Kotak Kotak1(3.3, 1.2, 1.5); // Deklarasi kotak1
  Kotak Kotak2(8.5, 6.0, 2.0); // Deklarasi kotak2

  // Menampilkan jumlah objek.
  cout << "Jumlah objek: " << Kotak::jumlahObjek << endl;

  return 0;
}
```

Kode tersebut akan menghasilkan output berikut:

```
Konstruktor dipanggil.
Konstruktor dipanggil.
Jumlah objek: 2
```

Fungsi static

Fungsi member static dapat dipanggil bahkan jika tidak ada objek kelas yang dibuat dan fungsi static dipanggil hanya menggunakan nama kelas dan operator ::.

Fungsi member static hanya dapat mengakses variabel member static, fungsi static lain, dan fungsi lainnya dari luar kelas.

Berikut ini contohnya:

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

class Kotak
{
  public:
    static int jumlahObjek;

    // Definisi konstruktor
    Kotak(double p=2.0, double l=2.0, double t=2.0)
    {
      cout <<"Konstruktor dipanggil." << endl;
      panjang = p;
```

```
        lebar = l;
        tinggi = t;

        // ditambahkan setiap kali objek dibuat
        jumlahObjek++;
      }

      double Volume()
      {
        return panjang * lebar * tinggi;
      }

      static int getJumlah()
      {
        return jumlahObjek;
      }

    private:
      double panjang; // Panjang kotak
      double lebar; // Lebar kotak
      double tinggi; // Tinggi kotak
};

// Inisialisasi member static dari kelas Kotak
int Kotak::jumlahObjek = 0;

int main(void)
{
  // Menampilkan jumlah objek sebelum objek dibuat.
  cout << "Jumlah objek awal: " << Kotak::getJumlah() << endl;

  Kotak Kotak1(3.3, 1.2, 1.5); // Deklarasi kotak1
  Kotak Kotak2(8.5, 6.0, 2.0); // Deklarasi kotak2

  // Menampilkan jumlah objek setelah objek dibuat.
  cout << "Jumlah objek akhir: " << Kotak::jumlahObjek
       << endl;

  return 0;
}
```

Kode tersebut akan menghasilkan output berikut:

```
Jumlah objek awal: 0
Konstruktor dipanggil.
Konstruktor dipanggil.
Jumlah objek akhir: 2
```

BAB 18

Pewarisan dalam C++

Konsep pewarisan dalam pemrograman berorientasi objek memungkinkan suatu kelas untuk mewarisi member dari kelas lain.

Konsep pewarisan mengimplementasikan hubungan ‘adalah’. Contohnya, mamalia ‘adalah’ binatang, anjing ‘adalah’ mamalia, anjing ‘adalah’ binatang, dan sebagainya.

Superkelas dan Subkelas

Sebuah kelas dapat mewarisi variabel dan fungsi member dari satu atau lebih superkelas.

Berikut sintaksnya:

```
class <superkelas>
{
  ...
}

class <subkelas> : <spesifier akses> <superkelas>
{
  ...
}
```

Berikut ini contoh superkelas Bentuk dan subkelas SegiEmpat:

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

// Superkelas
class Bentuk
{
  public:
    void setLebar(int l)
    {
      lebar = l;
    }

    void setTinggi(int t)
    {
      tinggi = t;
    }

  protected:
    int lebar;
    int tinggi;
};

// Subkelas
class SegiEmpat: public Bentuk
{
  public:
    int getLuas()
    {
      return (lebar * tinggi);
    }
};

int main(void)
{
  SegiEmpat Rect;
  Rect.setLebar(5);
  Rect.setTinggi(7);

  // Menampilkan luas objek.
  cout << "Luas: " << Rect.getLuas() << endl;

  return 0;
}
```

Kode tersebut akan menghasilkan output berikut:

```
Luas: 35
```

Pewarisan Multiple

Sebuah kelas C++ dapat mewarisi member dari lebih dari satu kelas. Berikut sintaksnya:

```
class <subkelas> : <spesifier akses> <superkelas1>, <spesifier
                   akses> <superkelas2>, .....
{
  ...
}
```

Berikut contohnya:

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

// Superkelas Bentuk
class Bentuk
{
  public:
    void setLebar(int l)
    {
      lebar = l;
    }

    void setTinggi(int t)
    {
      tinggi = t;
    }

  protected:
    int lebar;
    int tinggi;
};

// Superkelas BiayaCat
class BiayaCat
{
  public:
    int getBiaya(int luas)
    {
      return luas * 70;
    }
};

// Subkelas
class SegiEmpat: public Bentuk, public BiayaCat
{
  public:
    int getLuas()
    {
      return (lebar * tinggi);
    }
};

int main(void)
{
```

```
    SegiEmpat Rect;
    int luas;

    Rect.setLebar(5);
    Rect.setTinggi(7);

    luas = Rect.getLuas();

    // Menampilkan luas objek.
    cout << "Luas: " << Rect.getLuas() << endl;

    // Menampilkan biaya cat.
    cout << "Biaya cat: " << Rect.getBiaya() << endl;

    return 0;
}
```

Kode tersebut akan menghasilkan output berikut:

```
Luas: 35
Biaya cat: $2450
```

BAB 19

Overload dalam C++

C++ memungkinkan Anda untuk menspesifikasikan lebih dari satu definisi fungsi atau operator dengan nama yang sama. Hal ini disebut overload fungsi atau overload operator.

Deklarasi overload memiliki nama yang sama tetapi memiliki tipe argumen atau jumlah argumen yang berbeda.

Overload Fungsi

Anda dapat memiliki beberapa definisi fungsi dengan nama yang sama dalam lingkup yang sama. Definisi fungsi-fungsi tersebut harus memiliki tipe atau jumlah argumen yang berbeda.

Berikut contoh overload fungsi tampilkan ():

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

class tampilkanData
{
  public:
    void tampilkan(int i)
    {
      cout << "Menampilkan int: " << i << endl;
    }

    void tampilkan(double f)
    {
      cout << "Menampilkan float: " << f << endl;
    }
```

```
    void tampilkan(char* c)
    {
      cout << "Menampilkan karakter: " << c << endl;
    }
};

int main(void)
{
  tampilkanData td;

  // Menampilkan integer
  td.tampilkan(5);

  // Menampilkan float
  td.tampilkan(500.263);

  // Menampilkan karakter
  td.tampilkan("Halo C++");

  return 0;
}
```

Kode tersebut akan menghasilkan output berikut:

```
Menampilkan int: 5
Menampilkan float: 500.263
Menampilkan karakter: Halo C++
```

BAB 20

Polimorfisme dalam C++

Polimorfisme berarti mempunyai banyak bentuk. Polimorfisme dalam C++ berarti bahwa pemanggilan terhadap fungsi member akan menyebabkan fungsi tertentu dijalankan sesuai dengan tipe objek yang memanggil fungsi tersebut.

Berikut contoh superkelas yang diwarisi oleh dua subkelas:

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

class Bentuk
{
  protected:
    int lebar, tinggi;

  public:
    Bentuk( int a=0, int b=0)
    {
      lebar = a;
      tinggi = b;
    }

    virtual int luas()
    {
      cout << "Luas superkelas :" <<endl;
      return 0;
    }
};

class SegiEmpat: public Bentuk
```

```
{
  public:
    SegiEmpat( int a=0, int b=0)
    {
      Bentuk(a, b);
    }

    int luas ()
    {
      cout << "Luas kelas SegiEmpat :" <<endl;
      return (lebar * tinggi);
    }
};

class SegiTiga: public Bentuk
{
  public:
    SegiTiga( int a=0, int b=0)
    {
      Bentuk(a, b);
    }

    int luas ()
    {
      cout << "Luas kelas SegiTiga :" <<endl;
      return (lebar * tinggi / 2);
    }
};

// Fungsi main
int main( )
{
  Bentuk *bentuk;
  SegiEmpat rec(10,7);
  SegiTiga tri(10,5);

  // menyimpan alamat SegiEmpat
  bentuk = &rec;

  // memanggil luas segiempat.
  bentuk->luas();

  // menyimpan alamat SegiTiga
  bentuk = &tri;

  // memanggil luas segitiga.
  bentuk->luas();

  return 0;
}
```

Kode tersebut akan menghasilkan ouput berikut:

```
Luas kelas SegiEmpat
Luas kelas SegiTiga
```

Fungsi Virtual

Fungsi virtual adalah fungsi dalam superkelas yang dideklarasikan dengan kata kunci virtual.

Anda dapat membuat fungsi virtual tanpa definisi pada superkelas untuk didefinisikan pada subkelas yang mewarisinya. Hal ini disebut fungsi virtual murni.

Berikut ini contoh fungsi luas() yang dibuat menjadi fungsi virtual murni:

```
class Bentuk
{
  protected:
    int lebar, tinggi;

  public:
    Bentuk( int a=0, int b=0)
    {
      lebar = a;
      tinggi = b;
    }

    // fungsi virtual murni
    virtual int luas() = 0;
};
```

BAB 21

Abstraksi dan Enkapsulasi Data dalam C++

Abstraksi Data

Abstraksi data artinya hanya menyediakan informasi penting saja dari kelas untuk diakses dari luar dan menyembunyikan data detailnya.

Berikut contohnya:

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

class Adder
{
  public:
    // konstruktor
    Adder(int i = 0)
    {
      total = i;
    }

    // interface untuk diakses dari luar
    void addNum(int number)
    {
      total += number;
    }

    // interface untuk diakses dari luar
    int getTotal()
    {
      return total;
    };

  private:
    // menyembunyikan data
    int total;
};
```

```
int main( )
{
  Adder a;
  a.addNum(10);
  a.addNum(20);
  a.addNum(30);

  cout << "Total " << a.getTotal() <<endl;

  return 0;
}
```

Kode tersebut akan menghasilkan output berikut:

```
Total 60
```

Enkapsulasi Data

Enkapsulasi data merupakan mekanisme untuk membungkus data, dan fungsi yang menggunakannya sedangkan abtraksi data adalah mekanisme yang hanya menyediakan interface dan menyembunyikan detail implementasinya.

Berikut contohnya:

```
class Kotak
{
  public:
    double getVolume(void)
    {
      return panjang * lebar * tinggi;
    }

  private:
    double panjang; // Panjang kotak
    double lebar; // Lebar kotak
    double tinggi; // Tinggi kotak
};
```

Pada kode tersebut, variabel panjang, lebar, dan tinggi adalah private. Ini berarti variabel tersebut hanya dapat diakses oleh member lain dalam kelas Kotak saja. Hal ini merupakan praktek dari enkapsulasi data.

BAB 22

Interface dalam C++

Interface mendeskripsikan perilaku atau kapabilitas dari sebuah kelas tanpa menyediakan implementasi dari kelas tersebut.

Interface diimplementasikan dengan menggunakan kelas abstrak. Kelas abstrak memiliki minimal satu fungsi virtual murni. Fungsi virtual murni dispesifikasikan dengan menambahkan "=0" pada deklarasinya.

```
class Kotak
{
  public:
    // fungsi virtual murni
    virtual double getVolume() = 0;

  private:
    double panjang; // Panjang kotak
    double lebar; // Lebar kotak
    double tinggi; // Tinggi kotak
};
```

Contoh Kelas Abstrak

Berikut ini contoh superkelas yang menyediakan interface bagi subkelas untuk mengimplimentasikan fungsi getLuas():

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

// Superkelas
class Bentuk
{
  public:
    // fungsi virtual murni.
    virtual int getLuas() = 0;
```

```
    void setLebar(int l)
    {
      lebar = l;
    }

    void setTinggi(int t)
    {
      tinggi = t;
    }

  protected:
    int lebar;
    int tinggi;
};

// Subkelas
class Segiempat: public Bentuk
{
  public:
    int getLuas()
    {
      return (lebar * tinggi);
    }
};

class Segitiga: public Bentuk
{
  public:
    int getLuas()
    {
      return (lebar * tinggi)/2;
    }
};

int main(void)
{
  Segiempat Rect;
  Segitiga Tri;

  Rect.setLebar(5);
  Rect.setTinggi(7);

  // Menampilkan luas objek.
  cout << "Luas segiempat: " << Rect.getLuas() << endl;

  Tri.setLebar(5);
  Tri.setTinggi(7);

  // Menampilkan luas objek.
  cout << "Luas segitiga: " << Tri.getLuas() << endl;

  return 0;
}
```

Kode tersebut akan menghasilkan output berikut:

```
Luas segiempat: 35
Luas segitiga: 17
```

BAB 23

File dan Stream dalam C++

Pada bab ini Anda akan menggunakan library fstream yang mendefinisikan tiga tipe data baru.

Tipe Data	**Deskripsi**
Ofstream	Tipe data ini merepresentasikan stream file output dan digunakan untuk membuat file dan menuliskan informasi ke dalam file.
Ifstream	Tipe data ini merepresentasikan stream file input dan digunakan untuk membaca informasi dari dalam file.
Fstream	Tipe data ini merepresentasikan stream file secara umum, dan memiliki semua kemampuan dari ofstream dan ifstream, yaitu dapat membuat file, menuliskan informasi ke dalam file, dan membaca informasi dari dalam file.

Tipe data dalam library fstream

Untuk menggunakan tipe data di atas, Anda perlu menambahkan file header <iostream> dan <fstream> pada kode program.

Membuka File

File harus dibuka sebelum Anda membaca atau menuliskan informasi ke dalamnya. Anda dapat membuka file menggunakan objek ofstream atau fstream untuk menuliskan informasi, atau objek ifstream untuk membaca file. Berikut sintaksnya:

```
void open(const char *namafile, ios::openmode mode);
```

Berikut ini daftar mode yang dapat digunakan:

Mode	Deskripsi
ios::app	Mode Append. Semua output pada file tersebut akan ditambahkan di bagian akhir.
ios::ate	Membuka file untuk output dan memindahkan kontrol read/write pada bagian akhir file.
ios::in	Membuka file untuk membacanya.
ios::out	Membuka file untuk menuliskan ke dalamnya.
ios::trunk	Jika file sudah ada, kontennya akan dipotong sebelum membuka file.

Daftar mode

Anda dapat menggabungkan dua mode atau lebih dengan menggunakan OR. Berikut contohnya:

```
ofstream outfile;
outfile.open("file.dat", ios::out | ios::trunc );

fstream afile;
afile.open("file.dat", ios::out | ios::in );
```

Menutup File

Ketika program C++ selesai digunakan, semua stream akan secara otomatis dihapus dan semua file yang digunakan akan ditutup.

Anda dapat menutup file secara manual dengan fungsi close():

```
void close();
```

Menulis ke dalam File

Untuk menuliskan informasi ke dalam file, Anda dapat menggunakan operator << dengan objek ofstream atau fstream.

Membaca File

Untuk membaca informasi yang ada di dalam file, Anda dapat menggunakan operator << dengan objek ifstream atau fstream.

Contoh Membaca dan Menulis ke dalam File

Berikut ini contoh program yang membuka file kemudian menuliskan informasi dan membaca informasi dalam file:

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

int main ()
{
  char data[100];

  // membuka file dengan mode tulis.
  ofstream outfile;
  outfile.open("afile.dat");

  cout << "Menuliskan ke dalam file" << endl;
  cout << "Nama Anda: ";
  cin.getline(data, 100);

  // menuliskan data ke dalam file.
  outfile << data << endl;
  cout << "Umur Anda: ";
  cin >> data;
  cin.ignore();

  // menuliskan data ke dalam file.
  outfile << data << endl;
  // menutup file.
  outfile.close();

  // membuka file dengan mode baca.
  ifstream infile;
  infile.open("afile.dat");

  cout << "Membaca file" << endl;
  infile >> data;

  // menampilkan data pada layar monitor.
  cout << data << endl;

  // membaca data dari file dan menampilkannya.
```

```
  infile >> data;
  cout << data << endl;

  // menutup file.
  infile.close();

  return 0;
}
```

Kode tersebut akan menghasilkan contoh output sebagai berikut:

```
$./a.out
Menuliskan ke dalam file
Nama Anda: Zara
Umur Anda: 9
Membaca file
Zara
9
```

Pointer Posisi File

Pointer posisi file merupakan nilai integer yang menspesifikasikan lokasi pada file.

istream dan ostream menyediakan fungsi member untuk mengatur pointer posisi file. Fungsi tersebut adalah seekg untuk istream dan seekp untuk ostream.

Berikut contohnya penggunaannya:

```
// posisi byte ke-n dari fileObject (diasumsikan ios::beg)
fileObject.seekg( n );

// posisi n byte maju pada fileObject
fileObject.seekg( n, ios::cur );

// posisi n byte mundur dari bagian akhir fileObject
fileObject.seekg( n, ios::end );

// posisi akhir dari fileObject
fileObject.seekg( 0, ios::end );
```

BAB 24

Penanganan Eksepsi dalam C++

Eksepsi merupakan masalah yang muncul pada saat eksekusi program. Eksepsi C++ merupakan response terhadap keadaan yang muncul saat program berjalan, seperti percobaan untuk pembagian dengan nol.

Penanganan eksepsi dalam C++ menggunakan tiga kata kunci:

- **throw:** Program melempar eksepsi ketika masalah terjadi. Hal ini dilakukan dengan menggunakan kata kunci throw.
- **catch:** Program menangkap eksepsi dengan handler eksepsi pada tempat di mana Anda ingin menangani masalah tersebut. Kata kunci catch mengindikasikan penangkapan suatu eksepsi.
- **try:** Blok try mengidentifikasikan sebuah blok kode yang mungkin mengaktifkan eksepsi tertentu. Blok try diikuti dengan satu blok catch atau lebih.

Berikut ini sintaksnya:

```
try
{
  // kode
}
catch( NamaEksepsi e1 )
{
  // blok catch
```

```
}
catch( NamaEksepsi e2 )
{
  // blok catch
}
catch( NamaEksepsi eN )
{
  // blok catch
}
```

Melempar Eksepsi

Eksepsi dapat dilempar dengan menggunakan statemen throw. Operan dari statemen throw menentukan tipe eksepsi dan dapat berupa ekspresi apa saja.

Berikut ini contohnya:

```
double pembagian(int a, int b)
{
  if( b == 0 )
  {
    throw "Kondisi pembagian dengan nol!";
  }

  return (a/b);
}
```

Menangkap Eksepsi

Blok catch yang mengikuti blok try menangkap eksepsi yang muncul. Anda dapat menentukan tipe eksepsi yang ingin ditangkap.

Berikut sintaksnya:

```
try
{
  // kode
}
catch( NamaEksepsi e )
{
  // kode untuk menangani eksepsi NamaEksepsi
}
```

Kode tersebut akan menangkap eksepsi dengan tipe NamaEksepsi. Jika Anda ingin blok catch untuk menangani tipe eksepsi apa saja, Anda perlu menggunakan tanda .... :

```
try
{
  // kode
}
catch(...)
{
  // kode untuk menangani eksepsi apa saja
}
```

Berikut ini contoh yang melempar eksepsi pembagian dengan nol:

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

double pembagian(int a, int b)
{
  if( b == 0 )
  {
    throw "Kondisi pembagian dengan nol!";
  }

  return (a/b);
}

int main ()
{
  int x = 50;
  int y = 0;
  double z = 0;

  try
  {
    z = pembagian(x, y);
    cout << z << endl;
  }
  catch (const char* msg)
  {
    cerr << msg << endl;
  }

  return 0;
}
```

Kode tersebut akan menghasilkan output berikut:

```
Kondisi pembagian dengan nol!
```

Eksepsi Standar C++

C++ menyediakan eksepsi standar dalam <exception> yang dapat Anda gunakan dalam program. Berikut ini daftar eksepsi standar C++:

Eksepsi	Deskripsi
std::exception	Eksepsi dan superkelas dari semua eksepsi standar.
std::bad_alloc	Ini dapat dilempar oleh **new**.
std::bad_cast	Ini dapat dilempar oleh **dynamic_cast**.
std::bad_exception	Ini dapat digunakan untuk menangani eksepsi tak terduga dalam program.
std::bad_typeid	Ini dapat dilempar oleh **typeid**.
std::logic_error	Eksepsi yang secara teori dapat dideteksi dengan cara membaca kode.
std::domain_error	Ini merupakan eksepsi yang dilempar ketika domain invalid digunakan.
std::invalid_argument	Ini dilempar karena adanya argumen invalid.
std::length_error	Ini dilempar ketika std::string yang terlalu besar dibuat.
std::out_of_range	Ini dapat dilempar oleh method seperti contoh std::vector dan std::bitset<>::operator[].
std::runtime_error	Eksepsi yang secara teori tidak dapat dideteksi dengan cara membaca kode.
std::overflow_error	Eksepsi ini dilempar jika terjadi overflow.
std::range_error	Ini terjadi ketika Anda mencoba menyimpan nilai yang berada di luar jangkauan.
std::underflow_error	Eksepsi ini dilempar jika terjadi underflow.

Daftar eksepsi standar C++

Membuat Eksepsi Baru

Anda dapat membuat eksepsi baru dengan mewarisi dan melakukan override fungsionalitas pada kelas Exception.

Berikut ini contohnya:

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

struct EksepsiKu : public exception
{
  const char * what() const throw ()
  {
    return "Eksepsi C++";
  }
};

int main()
{
  try
  {
    throw EksepsiKu();
  }
  catch(EksepsiKu &e)
  {
    std::cout << "EksepsiKu muncul" << std::endl;
    std::cout << e.what() << std::endl;
  }
  catch(std::exception &e)
  {
    // Error lainnya
  }
}
```

Kode tersebut akan menghasilkan output berikut:

```
EksepsiKu muncul
Eksepsi C++
```

Pada kode di atas, what() merupakan method public yang disediakan oleh kelas exception. Fungsi ini mengembalikan penyebab terjadinya eksepsi.

BAB 25

Namespace dalam C++

Namespace didesain untuk memungkinkan penggunaan nama fungsi, kelas, atau variabel yang sama pada library yang berbeda.

Mendefinisikan Namespace

Definisi namespace dimulai dengan kata kunci namespace dan diikuti dengan nama:

```
namespace nama_namespace
{
  // deklarasi kode
}
```

Berikut contoh penggunaannya:

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

// namespace pertama
namespace pertama
{
  void func()
  {
    cout << "Dalam namespace pertama" << endl;
  }
}

// namespace kedua
namespace kedua
{
  void func()
  {
    cout << "Dalam namespace kedua" << endl;
  }
}
```

```
int main ()
{
  // Memanggil fungsi dari namespace pertama.
  pertama::func();

  // Memanggil fungsi dari namespace kedua.
  kedua::func();

  return 0;
}
```

Kode tersebut akan menghasilkan output berikut:

```
Dalam namespace pertama
Dalam namespace kedua
```

Directive Using

Anda juga dapat menggunakan directive namespace using sehingga tidak harus menuliskan nama namespace di awal fungsi. Berikut contohnya:

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

// namespace pertama
namespace pertama
{
  void func()
  {
    cout << "Dalam namespace pertama" << endl;
  }
}

// namespace kedua
namespace kedua
{
  void func()
  {
    cout << "Dalam namespace kedua" << endl;
  }
}

using namespace pertama;

int main ()
{
  // Memanggil fungsi dari namespace pertama.
  func();

  return 0;
}
```

Kode tersebut akan menghasilkan output berikut:

```
Dalam namespace pertama
```

Tentang Penulis

JUBILEE ENTERPRISE

Jubilee Enterprise adalah "a Creative Media Content Provider" dengan misi "Mengeksplorasi Teknologi Informasi tercanggih di dunia dan menyajikannya dalam bentuk media dengan gaya bahasa yang sederhana, mudah dicerna, dan gampang dipraktikkan oleh siapa pun".

Di Jubilee Enterprise, "Information Technology is our passion". Itulah mengapa setiap hari kami mengeksplorasi, meneliti, dan bereksperimen dengan banyak teknologi tercanggih saat ini. Hasil penelitian tersebut kami persembahkan dalam bentuk media cetak (buku) dan elektronik (blog).

Buku-buku kami, yang diterbitkan oleh PT Elex Media Komputindo (Kelompok Kompas Gramedia), telah didistribusikan ke seluruh Indonesia dan Malaysia, membantu dan menginspirasi pembaca-pembaca kami ketika menggunakan program Photoshop, CorelDraw, MS Office, Internet, Gagdet, dan lain sebagainya secara mudah dan praktis.

Catatan:

- Untuk melakukan pemesanan buku, hubungi
  Layanan Langsung PT Elex Media Komputindo:
  Gramedia Direct
  Jl. Palmerah Barat No. 33, Jakarta 10270
  Telemarketing/CS: 021-53650110/111 ext: 3901/3902
  Email: **endang@gramediapublishers.com**

Cepat Menguasai C++

C++ dianggap sebagai bahasa pemrograman masa depan. Oleh karena itu, mempelajari C++ di saat sekarang bernilai investasi. Dari sudut pandang ekonomis inilah, kami suguhkan tema C++ kepada para pembaca sebagai salah satu alternatif pemrograman di tengah beragamnya tema programming.

Buku ini mengupas tentang C++ dan bagaimana Anda bisa mempelajari pemrograman ini dengan cepat dan ringkas. Diharapkan, Anda akan mengenal bahasa pemrograman yang di masa mendatang akan dihargai sangat tinggi ini. Dengan penjelasan yang praktis, singkat, dan jelas akan memudahkan Anda selangkah demi selangkah mengenal C++. Bacalah dan praktikkan dalam latihan sehari-hari untuk membuat script yang praktis dan fungsional.

Materi selengkapnya antara lain:

- Pengenalan C++
- Sintak-sintak dasar C++
- String dalam C++
- Abstraksi dan enkapsulasi data dalam C++
- Pengambilan keputusan dalam C++
- Interface dalam C++
- Penanganan eksepsi dalam C++
- Namespace dalam C++
- Pointer dalam C++
- Pengulangan dalam C++
- Operator dalam C++

PT ELEX MEDIA KOMPUTINDO
Kompas Gramedia Building
Jl. Palmerah Barat 29-37, Jakarta 10270
Telp. (021) 53650110-53650111, Ext 3214
Webpage: http://www.elexmedia.co.id

Kelompok
Pemrograman
Keterampilan
- [x] Tingkat Pemula
- [x] Tingkat Menengah
- [ ] Tingkat Mahir

Jenis Buku
- [x] Referensi
- [x] Tutorial
- [ ] Latihan

gramediana
ISBN 978-602-02-5826-3
9 786020 258263
121150272