

# Modul 1 CODE BLOCKS IDE & PENGENALAN BAHASA C++ (BAGIAN PERTAMA)

## TUJUAN PRAKTIKUM

1. Mengenal *environment* Code Blocks dengan baik.
2. Memahami cara menggunakan dan *troubleshooting* Code Blocks IDE.
3. Mengimplementasikan operator-operator dalam program.
4. Memahami cara membuat program sederhana dalam bahasa C++.
5. Memahami penggunaan tipe data dan variabel dalam bahasa C++.
6. Menggunakan operator-operator *input/output* dengan tepat.
7. Memahami dan mengimplementasikan fungsi kondisional dalam program.

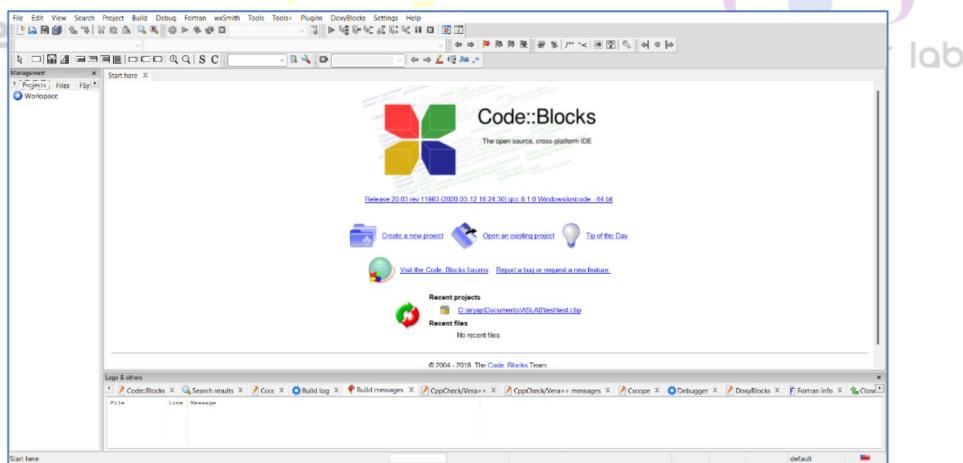
## 1.1 Pengenalan Code Blocks

Pada praktikum Struktur Data ini, kertas (*tool*) yang digunakan adalah Code Blocks. Kertas ini merupakan *free*, *open-source*, dan *cross-platform IDE*. Saat ini, Code Blocks berorientasi pada C/C++/Fortran (codeblocks, 2016).

### 1.1.1 Instalasi Code Blocks

Adapun cara menginstall Code Blocks adalah sebagai berikut.

1. Download terlebih dahulu *file exe* pada <http://www.codeblocks.org/downloads>. Pilih *Download the binary release* kemudian pilih *file* yang menggunakan mingw-setup (e.g. codeblocks-20.03mingw-setup.exe).
2. Setelah itu install *file* tersebut, akan muncul tampilan seperti pada Gambar 1. 1.

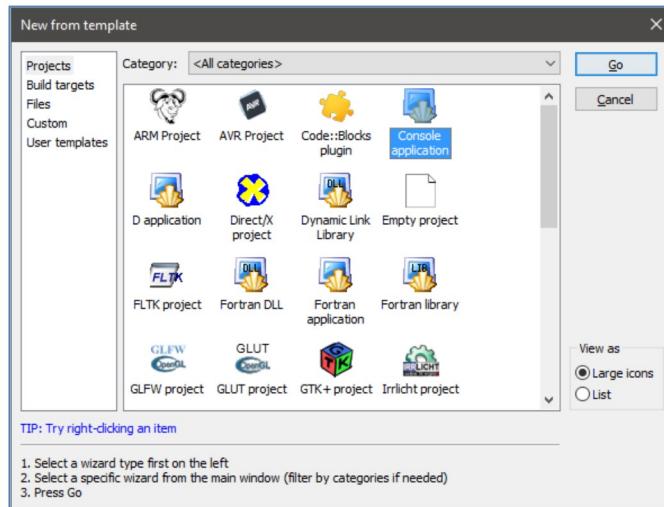


Gambar 1. 2 Code Blocks IDE

### 1.1.2 Cara Menggunakan Code Blocks

Adapun cara menggunakan Code Blocks adalah sebagai berikut.

1. Membuat *Project* Baru dengan cara memilih *File > New > Projects*. Kemudian pada panel kiri pilih *Project*, pada panel kanan pilih *Console application* kemudian klik *Go* seperti pada Gambar 1. 3.



Gambar 1. 4 Membuat *project* baru

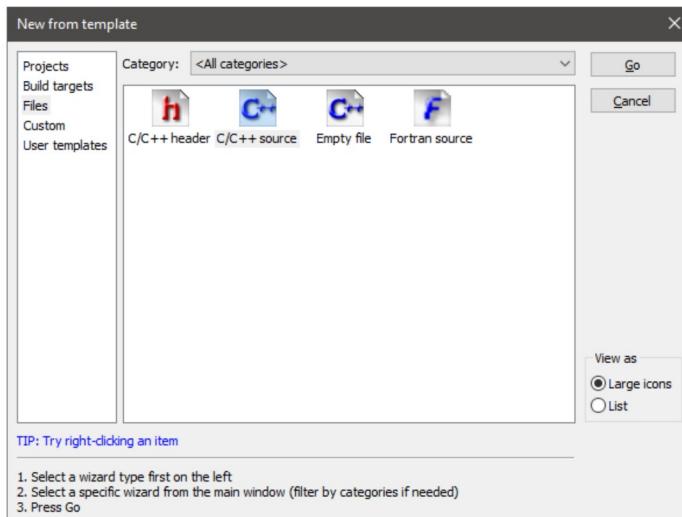
Setelah itu klik *Next >* Pilih Bahasa Pemrograman yang akan digunakan > Isi *Project title* dan *Folder to create project in* (tempat menyimpan *project*) > Klik *Finish*.

2. Menulis *syntax* pada *editor* seperti pada Gambar 1. 5.

```
#include <iostream>
using namespace std;
int main()
{
    cout << "Hello world!" << endl;
    return 0;
}
```

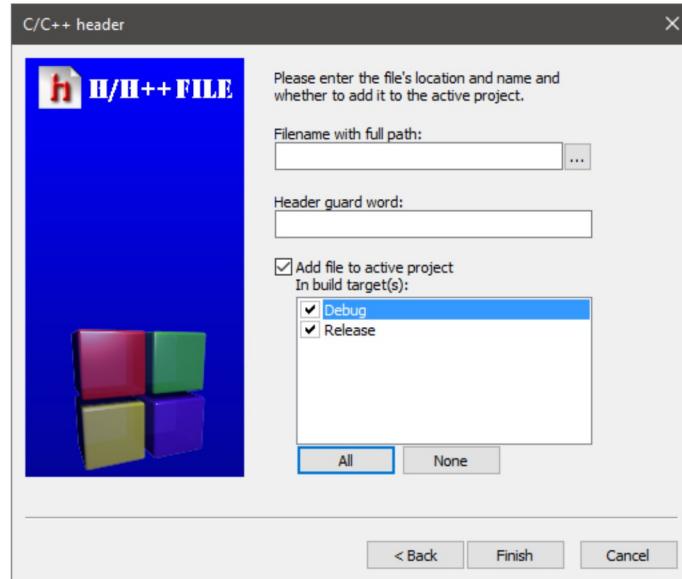
Gambar 1. 6 Menulis *syntax*

3. Membuat *Class* Baru dengan cara klik *File > New > File*. Pada panel kiri pilih *Files*, dan pada panel kanan pilih *C/C++ source* Kemudian Klik *Go* seperti pada Gambar 1. 7.



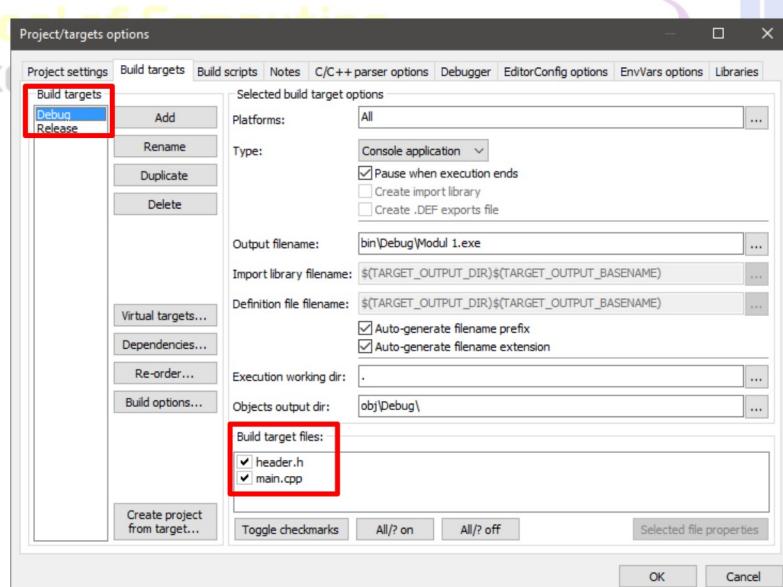
Gambar 1. 8 Membuat *class* baru

Kemudian klik *Next >* Pilih bahasa pemrograman > Isi *Filename with full path* > Centang *all in build target* > *Finish*, seperti pada Gambar 1. 9 1.



Gambar 1. 10 Centang all in build target

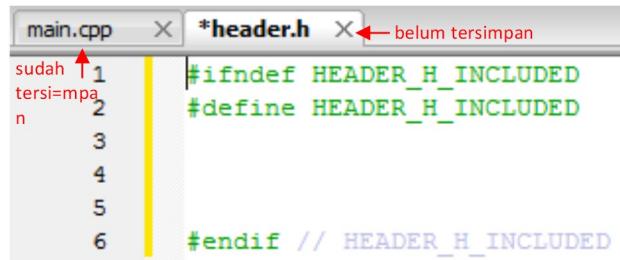
Jika anda lupa mencentang *build target*, dapat dilakukan *setting* manual dengan cara klik kanan pada *project > properties > Build targets > Debug* > Centang semua *target files* seperti pada Gambar 1. 11.



Gambar 1. 12 Target options

Setiap *class* harus memiliki sebuah nama yang dapat digunakan untuk membedakannya dari *class* lain. Penamaan *class* menggunakan kata benda tunggal yang merupakan abstraksi yang terbaik. Penulisan nama *class*, huruf pertama dari setiap kata pada nama *class* ditulis dengan menggunakan huruf kapital. Contohnya, Major dan StudentName. Jangan lupa untuk selalu menyimpan hasil pekerjaan anda. *Shortcut* untuk *save* satu *file* adalah *Ctrl+S* sedangkan

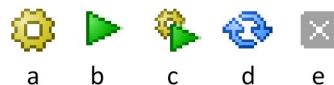
untuk menyimpan seluruh file adalah Ctrl+Shift+S. Class yang belum tersimpan akan ditandai dengan \* pada bagian kiri nama class, seperti pada Gambar 1. 13.



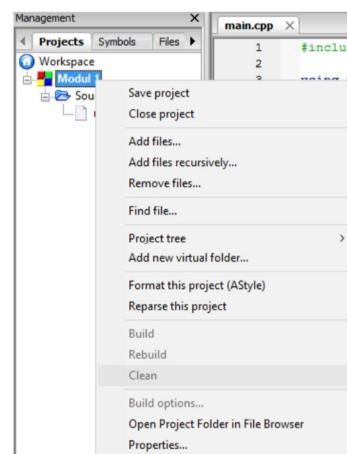
```
main.cpp *header.h
sudah 1 #ifndef HEADER_H_INCLUDED
tersimpan 2 #define HEADER_H_INCLUDED
3
4
5
6 #endif // HEADER_H_INCLUDED
```

Gambar 1. 14 Contoh file

4. *Compile* program, yaitu aksi untuk menjalankan program yang telah ditulis sehingga penulis dapat melihat *output* dari program.



- a. **Build** merupakan aksi untuk membangun syntax menjadi sebuah program. Adapun *shortcut* yang digunakan adalah Ctrl-F9.
  - b. **Run** merupakan aksi untuk menjalankan program yang telah di-build. Program tidak akan berjalan sebelum dilakukan *build*. Adapun *shortcut* yang digunakan adalah Ctrl-F10.
  - c. **Build and Run** adalah aksi yang mengizinkan *Build* dan *Run* berjalan berurutan secara otomatis. Adapun *shortcut* yang digunakan adalah F9.
  - d. **Rebuild** merupakan aksi untuk membangun kembali program. Adapun *shortcut* yang digunakan adalah Ctrl-F11.
  - e. **Abort** merupakan aksi untuk mematikan program yang sedang berjalan.
5. **Clean**. Terkadang program yang kita buat tidak dapat di *run*, untuk itu perlu dilakukan *clean project* dengan cara klik kanan pada *project* kemudian klik *Clean* seperti pada Gambar 1. 15. Setelah dilakukan *clean*, program akan dapat berjalan kembali.



Gambar 1. 16 Clean

6. **Close dan Open Project**. Untuk menutup *project* dilakukan dengan cara klik kanan pada *project* kemudian pilih *close*. *Project* yang di *close* tidak akan terhapus dan tetap ada pada *directory*.

Sedangkan untuk membuka *project* yang telah di *close* dilakukan dengan cara memilih *File > Open > Pilih File \*.cbp*

7. **Error Message** akan muncul ketika terjadi kesalahan pada penulisan *syntax*. Contoh seperti pada Gambar 1. 17, terdapat *error* pada *line 8*. Pada error message diharapkan *semicolon* (;) sebelum *return* dan ternyata pada *line 7* dapat dilihat bahwa penulisan *syntax* belum diakhiri untuk fungsi *cout*.

The screenshot shows the Code::Blocks IDE interface. In the top window, titled 'main.cpp', there is C++ code. Line 8 contains a syntax error: 'cout << "Hello world!" << endl' is followed by a closing brace '}' on the next line and a 'return 0;' statement on the line after that. A red square highlights the error in the code editor. Below the editor is the 'Logs & others' tab bar, which includes tabs like 'Code::Blocks', 'Search results', 'Cccc', 'Build log', 'Build messages', 'CppCheck', and 'CppCheck message'. The 'Build messages' tab is active and displays the following error message:

```
File           Line   Message
C:\Users\tst\...     == Build: Debug in Modul 1 (compiler: GNU GCC Compiler) ==
C:\Users\tst\...     In function 'int main()':
C:\Users\tst\...  8     error: expected ';' before 'return'
C:\Users\tst\...     == Build failed: 1 error(s), 0 warning(s) (0 minute(s), 0 second(s)) ==
```

Gambar 1. 18 Error Message

## 1.2 Sekilas tentang C++

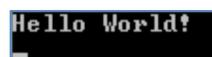
Bahasa C++ diciptakan oleh Bjarne Stroustrup di AT&T Bell Laboratories awal tahun 1980-an berdasarkan C ANSI (American National Standard Institute). Pertama kali, *prototype* C++ muncul sebagai C yang dipercanggih dengan fasilitas kelas. Bahasa tersebut disebut C dengan kelas (C with class). Selama tahun 1983-1984, C dengan kelas disempurnakan dengan menambahkan fasilitas pembebanlebihan operator dan fungsi yang kemudian melahirkan apa yang disebut C++. Simbol ++ merupakan operator C untuk operasi penaikan, muncul untuk menunjukkan bahwa bahasa baru ini merupakan versi yang lebih canggih dari C.

Borland International merilis *compiler* Borland C++ dan Turbo C++. Kedua *compiler* ini sama-sama dapat digunakan untuk mengkompilasi kode C++. Bedanya, Borland C++ selain dapat digunakan dibawah lingkungan DOS, juga dapat digunakan untuk pemrograman Windows. Selain Borland International, beberapa perusahaan lain juga merilis *compiler* C++, seperti Topspeed C++, Zortech C++ dan Code Blocks. Dalam praktikum ini, kita akan menggunakan bahasa C++.

```
1 #include <iostream>
2
3 using namespace std;
4 int main()
5 {
6     cout<<"Hello World!"<<endl;
7     return 0;
8 }
```

Program 1 Hello World

Setelah di-*compile* (F9) dan di-*run* (Ctrl+F9) maka hasil keluaran programnya adalah sebagai berikut:



Gambar 1. 19 Hello World

## 1.3 Dasar Pemrograman

### 1.3.1 Struktur Program C++

Secara umum, pembagian struktur bahasa pemrograman C++ adalah sebagai berikut.

C++	Keterangan
<code>#include &lt;iostream&gt;</code>	Pendeklarasian <i>library</i> yang akan digunakan di dalam program
<code>#define PHI 3.14</code> <code>const int constant1;</code> <code>const float constant2 = 0.5;</code>	Pendefinisian konstanta
<code>struct new_record_type {</code> <code>int element1;</code> <code>float element2;</code> }	Pendefinisian tipe data bentukan / <i>record type</i> / struktur
<code>int var1;</code> <code>float var2[2];</code>	Pendeklarasian variabel
<code>int function_A(){</code> <code>//</code> } <code>void procedure_B(){</code> <code>//</code> }	Pendeklarasian fungsi dan prosedur
<code>int main(){</code> <code>// blok program</code> <code>return 0;</code> }	Program utama

### 1.3.2 Pengenal (*Identifier*) Computing

*Identifier* merupakan nama yang biasa digunakan untuk variabel, konstanta, fungsi atau objek lain yang didefinisikan oleh program.

Aturan yang digunakan untuk menentukan *identifier*:

1. Harus diawali dengan huruf (A....Z, a....z) atau garis bawah (\_).
2. Karakter selanjutnya bisa berupa huruf, digit atau karakter garis bawah (\_) atau dollar (\$).
3. Panjang maksimal *identifier* adalah 32 karakter, jika lebih maka yang dianggap adalah 32 karakter awal.
4. Tidak boleh mengandung spasi.
5. Tidak boleh menggunakan operator aritmatika (+ - \* %).

Bahasa C++ bersifat *case sensitive*, jadi huruf besar dan huruf kecil dianggap berbeda.

Contoh:

panjang (berbeda dengan: Panjang)  
nilai4  
luas\_total  
harga\_beli\$

### 1.3.3 Tipe data dasar

Data merupakan suatu nilai yang dapat dinyatakan dalam bentuk konstanta atau variabel. Data berdasarkan jenisnya dibagi dalam 5 kelompok, yang dinamakan sebagai tipe data dasar. Kelima kelompok tersebut:

1. Bilangan bulat (*integer*).
2. Bilangan *real* presisi – tunggal.
3. Bilangan *real* presisi – ganda.
4. Karakter.

5. Tak-bertipe.

Daftar tipe data dasar:

Tabel 1-1 Tipe Data Dasar

Type data	Contoh	Ukuran	Jangkauan nilai
char	char nama[20];	1 byte	-128 s.d. +127
int	int nilai; int jumlah = 0;	2 byte	-32768 s.d +32767
long	long selisih;	4 byte	-2.147.438.648 s.d +2.147.438.647
float	float jumlah;	4 byte	3.4e-38 s.d 3.4e+38
double	double hasil;	8 byte	1.7e-308 s.d 1.7e+308

### 1.3.4 Variabel

Variabel dalam program digunakan untuk menyimpan nilai, nilai variabel bisa berubah – ubah selama program berjalan. Aturan penamaan variabel sesuai dengan aturan penamaan *identifier*. Bentuk umum pendeklarasian suatu variabel dalam bahasa pemrograman C dapat ditulis sebagai berikut.

```
tipe_data nama_variabel;
```

Contoh:

```
int x,y;  
char ch;
```

Kita juga dapat langsung memberikan nilai awal (inisialisasi) pada suatu variabel pada saat variabel tersebut di deklarasikan.

Contoh :

```
int x=20, y=6;  
char nama[30] = "Budi";
```

### 1.3.5 Konstanta

Konstanta menyatakan nilai yang selalu tetap. Seperti halnya dengan variabel, konstanta juga mempunyai tipe.

Untuk mendeklarasikan suatu nilai yang sifatnya konstan kita cukup menambahkan kata **const** di depan tipe data dan variabel.

Contoh:

```
const float phi = 3.14;  
const int n = 20;
```

## 1.4 Input / Output

### 1.4.1 Output

#### A. Fungsi cout()

Fungsi ini digunakan untuk mencetak data baik yang bertipe numerik, ataupun teks, baik konstanta ataupun variabel.

Algoritma	Program coba_output Algoritma output("saya lagi belajar bahasa C++ nih!!!") endprogram
-----------	---

C++	<pre>#include &lt;iostream&gt; using namespace std; int main() {     cout&lt;&lt;"saya lagi belajar bahasa C++ nih!!!"&lt;&lt;endl;     return 0; }</pre>
-----	---

*Output:*

saya lagi belajar bahasa C++ nih!!!

### B. Penentu Format

Penggunaan penentu format sangat berkaitan erat dengan suatu tipe data. Artinya suatu tipe data memiliki penentu format masing-masing. Format tersebut dipakai di bahasa C, sedangkan pada C++ tidak harus dipakai.

Tabel 1-2 Penentu Format

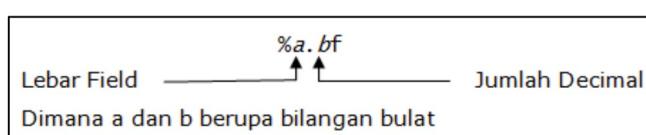
Tipe Data	Penentu Format
<i>Integer</i>	%d
<i>Floating Point</i> Bentuk Desimal	%f
<i>Floating Point</i> Bentuk Berpangkat	%e
<i>Floating Point</i> yang lebih pendek antara Desimal dan Berpangkat	%g
<i>Double Precision</i>	%lf
<i>Character</i>	%c
<i>String</i>	%s
<i>Unsigned integer</i>	%u
<i>Long Integer</i>	%ld
<i>Long Unsigned Integer</i>	%lu
<i>Unsigned hexadecimal integer</i>	%x
<i>Unsigned octal integer</i>	%o

### C. Penentu Lebar Field

Bila kita mencetak data bertipe *float*, seringkali tampilan yang diberikan tampak kurang manis. Misalnya desimal yang dicetak terlalu banyak.

```
1 #include <iostream>
2 using namespace std;
3 int main(){
4     float bil;
5     bil = 2.5;
6     cout<< "bilangan = " <<bil;
7     return 0;
}
```

Sebenarnya kita dapat mengatur lebar *field* dan jumlah desimal yang ingin dicetak pada layar dengan cara memberikan tambahan format %f dengan bentuk sebagai berikut:



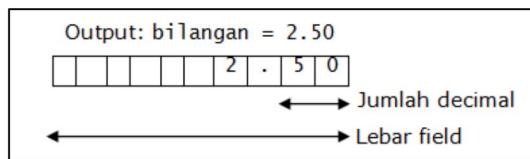
Gambar 1. 20 Penentu Lebar Field

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

```

4     float bil;
5     bil = 2.5;
6     cout<< "bilangan = %10.2f\n" << bil;
7     return 0;
}

```



Gambar 1. 21 Output cout dengan Penentu Lebar Field

#### D. Escape Sequence

Disebut *escape sequence*, karena notasi '\' dianggap sebagai karakter "*escape*" (menghindar), dalam arti bahwa karakter yang terdapat setelah tanda '\' dianggap merupakan bukan teks biasa.

Berikut ini beberapa *escape sequence*.

Tabel 1-3 Escape Sequence

Escape sequence	Pengertian
\b	Backspace
\f	Formfeed
\n	Baris baru
\r	Carriage return
\t	Tabulasi
'	Tanda kutip tunggal
"	Tanda kutip ganda
\\"	Backslash
\xaa	Kode ASCII dalam hexadecimal (aa: menunjukan angka ASCII)

#### 1.4.2 Input

##### A. Fungsi cin()

`cin()` merupakan salah satu fungsi yang digunakan untuk meminta inputan *keyboard* dari user. Bentuk umumnya pendeklarasiannya adalah sebagai berikut :

```
cin>>nama_variabel;
```

Algoritma	C++
<pre> Program coba_input kamus     inp : int  algoritma     input(inp)     output("nilai = ", inp) endprogram </pre>	<pre> #include &lt;iostream&gt; using namespace std; int main(){     int inp;     cin &gt;&gt; inp;     cout &lt;&lt; "nilai = " &lt;&lt; inp;     return 0; } </pre>

## B. Penentu Format

Penentu format untuk fungsi `cin()` **tidak** sama seperti penentu format untuk fungsi `printf()` (digunakan pada C atau C++).

Pada `cin()`, kita tidak perlu menggunakan penentu format seperti pada `printf()`. `cin` menggunakan operator `>>` untuk langsung memasukkan nilai yang diinput user ke dalam variabel yang ditentukan.

## C. Fungsi `getchar()`

`getchar()` adalah fungsi yang digunakan untuk membaca satu karakter dari input standar (biasanya keyboard). Fungsi ini akan menunggu pengguna menekan sebuah tombol pada keyboard, lalu mengembalikan karakter tersebut. Fungsi ini sering digunakan untuk membaca input karakter tunggal dari pengguna.

Berbeda dengan `cin`, yang memungkinkan pengguna memasukkan lebih dari satu karakter dan membutuhkan menekan tombol enter untuk menyelesaikan input, `getchar()` hanya membaca satu karakter dan tidak memerlukan enter untuk melanjutkan eksekusi program. Begitu satu karakter diketikkan, program akan langsung melanjutkan ke perintah berikutnya.

Dalam C++, `getchar()` juga dapat digunakan melalui namespace `std` jika kita hanya menggunakan header `<iostream>`.

Contoh Penggunaan `getchar()`:

```
1 #include <iostream>
2 using namespace std;
3 int main() {
4     char ch;
5
6     cout << "Masukkan sebuah karakter: ";
7     ch = getchar(); // Menggunakan getchar() untuk membaca satu karakter
8
9     cout << "Karakter yang Anda masukkan adalah: " << ch << endl;
10
11 }
```

Penjelasan :

1. Mengimpor Header `<iostream>`: Hanya menggunakan header `<iostream>` untuk mengakses fungsi `getchar()` serta fungsi input/output standar seperti `cout`.
2. Membaca Karakter: Fungsi `getchar()` membaca satu karakter dari input pengguna. Karakter tersebut kemudian disimpan dalam variabel `ch`.
3. Menampilkan Karakter: Karakter yang dimasukkan oleh pengguna ditampilkan kembali menggunakan `cout`.

## 1.5 Operator

Operator adalah suatu simbol yang digunakan untuk melakukan suatu operasi atau manipulasi. Bahasa C++ merupakan bahasa yang kaya dengan operator yaitu: Operator Aritmatika, Operator Penggeraan (*assignment operator*), Operator Logika, Operator Unary, Operator Bitwise, Operator Kondisional, dan lain-lain.

Tabel 1-4 Operator, Arah Proses, dan Jenjangnya

Kategori (Arti)	Operator	Kategori (Arti)	Operator
Panggilan fungsi, subscript <i>array</i> , dan elemen struktur data	() [] ->	Operator Hubungan (sama dengan, tidak sama dengan )	== !=
		Operator Bitwise AND	&

Operator Unary (NOT, komplemen, negasi, inkremen, dekremen , address, indirection)	!	Operator Bitwise XOR	<b>^</b>
	<b>~</b>	Operator Bitwise OR	<b> </b>
	<b>-</b>	Operator Logika AND	<b>&amp;&amp;</b>
	<b>++</b>	Operator Logika OR	<b>  </b>
	<b>--</b>	Operator Kondisional	<b>?:</b>
	<b>&amp;</b>		
	<b>*</b>		
Operator Aritmatika(Perkalian, pembagian, Sisa Pembagian/mod)	<b>*</b>	Operator Penggerjaan Aritmatika <i>(assignment, assignment perkalian, assignment pembagian, assignment mod, assignment penjumlahan, assignment pengurangan )</i>	<b>=</b> <b>*=</b> <b>/=</b> <b>%=</b> <b>+=</b> <b>-=</b>
Operator Aritmatika (Pertambahan, Pengurangan )	<b>/</b>		
	<b>%</b>		
	<b>+</b>	Operator Penggerjaan Bitwise <i>(assignment AND bitwise, assignment OR bitwise, assignment XOR bitwise, assignment shift kanan</i>	<b>&amp;=</b> <b>^=</b>
	<b>-</b>		<b> =</b> <b>&lt;=</b> <b>&gt;=</b>
Operator Bitwise Pergeseran Bit (shift kiri, shift kanan)	<b>&lt;&lt;</b>		
	<b>&gt;&gt;</b>		
Operasi Hubungan (kurang dari, kurang dari atau sama dengan lebih dari, lebih dari atau sama dengan )	<b>&lt;</b>	Operator Koma	<b>,</b>
	<b>&lt;=</b>		
	<b>&gt;</b>		
	<b>&gt;=</b>		

### A. Operator Aritmatika

Misalkan terdapat ungkapan sebagai berikut :  $A + B / C + D$ . Untuk mengubah jenjang dapat digunakan tanda kurung '()' (sebagai operator jenjang tertinggi) sebagai berikut:  $(A + B)/(C + D)$ .

```

1 #include <iostream>
2 using namespace std;
3 int main(){
4     int W, X, Y; float Z;
5     X = 7; Y = 3; W = 1;
6     Z = (X + Y)/(Y + W);
7     cout<< "Nilai z = " << Z << endl;
8     return 0;
9 }
10

```

### B. Operator Penggerjaan (*Assignment*)

Operator ini digunakan untuk memindahkan nilai dari suatu ungkapan ke suatu pengenal. Di samping operator penggerjaan **=** yang umumnya digunakan di bahasa-bahasa pemrograman, bahasa C++ menyediakan beberapa operator penggerjaan lain.

Misalnya:  $A += 7$ , ekuivalen dengan,  $A = A + 7$

### C. Operator Logika

Operasi logika membandingkan dua buah nilai logika. Nilai logika adalah nilai benar atau salah.

Misalnya:

1.  $(\text{kar} > 'A') \&\& (\text{kar} < 'Z')$   
Hasil operasi **&&** bernilai benar hanya jika kar > 'A' dan kar < 'Z'
2.  $(\text{pilihan} == 'Y') || (\text{pilihan} == 'y')$   
Hasil operasi logika **||** bernilai benar jika pilihan berupa 'Y' atau 'y'.
3. **! operand**  
Hasil operand **!** akan bernilai benar jika operand bernilai salah dan sebaliknya.

## D. Operator Unary

### 1) Operator Unary tipe

Jika pernyataan  $Z = (X + Y)/(Y + W)$  yang memiliki *output* 2.000000 diubah menjadi  $Z = (\text{float})(X + Y)/(Y + W)$  maka outputnya menjadi : Nilai Z = 2.500000. Mengapa demikian?

Itulah peranan dari operator tipe ini. Operator ini akan menghasilkan tipe data yang diinginkan walau berasal dari operand-operand dengan tipe berbeda.

### 2) Operator Unary sizeof

Operator Unary sizeof digunakan untuk mengetahui ukuran *memory* dari operandnya dalam satuan byte. Misalnya untuk komputer IBM PC, ukuran dari nilai float adalah 4 byte, kemungkinan di komputer yang lain, ukuran float bukan 4 byte. Contoh penggalan program penggunaan sizeof:

```
printf("Ukuran karakter = %d byte \n", sizeof(char));
```

## E. Operator sizeof

Operator sizeof menghasilkan ukuran dari variabel atau suatu tipe pada saat kompilasi program. Penggunaan sizeof dengan operan (berupa variabel atau tipe) yang ditempatkan dalam tanda kurung.

Contoh:

```
sizeof(char) → 1  
sizeof(int) → 2
```

Operator ini sangat bermanfaat untuk menghitung besarnya sebarang tipe atau variabel, terutama untuk variabel dan tipe yang kompleks (seperti struktur).

```
1 #include <iostream>  
2 using namespace std;  
3 int main(){  
4     char x;  
5     float y;  
6  
7     cout << "ukuran variabel char " << sizeof(x) << endl;  
8     cout << "ukuran variabel float " << sizeof(y) << endl;  
9     cout << "ukuran variabel int " << sizeof(int) << endl;  
10  
11     return 0;  
12 }
```

## F. Operator Increment dan Decrement

Bahasa C++ menyediakan 2 operator yang tidak biasa untuk *increment* dan *decrement*. Operator *increment* ++ akan menambahkan nilai 1 dari variabel, sedangkan operator *decrement* -- akan mengurangi variabel dengan nilai 1.

Contoh:

```
++i; // ekuivalen dengan i=i+1  
--j; // ekuivalen dengan j=j-1
```

Kedua operator ini bisa diletakkan sebelum variabel (prefix, contoh: ++i), dan sesudah variabel (postfix, contoh: i++) . Effect dari keduanya sama yaitu akan menambahkan nilai 1 ke variabel i. Akan tetapi, untuk ekspresi ++i dan i++ ada bedanya. Jika ekspresi ++i maka variabel akan di-*increment* dulu sebelum digunakan sedangkan untuk operator i++ maka nilai i akan digunakan terlebih dahulu setelah itu baru di-*increment*.

Untuk lebih jelasnya perhatikan penggalan program berikut.

Algoritma	C++
-----------	-----

```

Program coba_increment_belakang
kamus
    r,s : integer
algoritma
    r ← 10
    r ← r + 1
    s ← 10 + r
    output(r,s)
endprogram

```

```

#include <iostream>
using namespace std;
int main(){
    int r = 10;
    int s;
    s=10 + ++r;
    cout<< "Nilai r= "<<r<<endl;
    cout<< "Nilai s= "<<s<<endl;
    return 0;
}

```

```

Nilai r=11
Nilai s=21_

```

Gambar 1. 22 *Increment* di Belakang

Penjelasan: pada contoh di atas nilai r pertama di-*increment* terlebih dahulu sehingga nilai r=11 dan kemudian dijumlahkan dengan 10 sehingga nilai s sama dengan 21. *Increment* ini biasanya disebut ***pre-increment***.

Algoritma	C++
<pre> Program coba_increment_depan kamus     r,s : integer algoritma     r ← 10     s ← 10 + r     r ← r + 1     output (r,s) endprogram </pre>	<pre> #include &lt;iostream&gt; #include &lt;stdlib.h&gt; using namespace std; int main(){     int r = 10;     int s;     s=10 + r++;     cout&lt;&lt; "Nilai r= "&lt;&lt;r&lt;&lt;endl;     cout&lt;&lt; "Nilai s= "&lt;&lt;s&lt;&lt;endl;     return 0; } </pre>

```

Nilai r=11
Nilai s=20_

```

Gambar 1. 23 Increment di Depan

Penjelasan: pada contoh di atas nilai s diisi terlebih dahulu dengan penjumlahan antara 10 dengan r sehingga nilai s=20. Kemudian setelah itu baru nilai r di-*increment* sehingga nilai r=11. *Increment* dengan cara seperti ini biasanya disebut ***post-increment***.

## 1.6 Pemodifikasi Tipe

Pemodifikasi tipe (*type modifier*) dapat dikenakan diawal tipe data kecuali untuk void.

### 1.6.1 Unsigned

Tipe data ini digunakan bila kita hanya ingin bekerja dengan data yang bernilai positif saja. Misalnya *unsigned integer* akan menerima data dari 0 – 65.535. (tidak lagi dari -32.768 hingga 32.768).

Contoh cara deklarasinya:

```
unsigned int jumlah;  
unsigned char ch;
```

### 1.6.2 Short

Tipe data ini kadangkala disamakan dengan *integer* dan kadangkala juga dibedakan, tergantung pada sistem dan jenis komputer yang digunakan.

### 1.6.3 Long

Tipe data ini digunakan untuk menaikkan kapasitas dari suatu variabel. Misalnya long *integer* memiliki bilangan bulat dari -2.147.483.648 hingga 2.147.483.647

Contoh Pendeklarasiannya :

```
unsigned long int harga_rumah;
```

## 1.7 Kondisional

Untuk menyelesaikan suatu masalah diperlukan pengambilan keputusan. Bahasa C++ menyediakan beberapa jenis pernyataan berupa operator kondisi sebagai berikut.

1. Pernyataan **if**
2. Pernyataan **if-else**
3. Pernyataan **switch**

Pernyataan pengambilan keputusan di atas memerlukan suatu kondisi sebagai basis pada pengambilan keputusan. Kondisi umum yang dipakai keadaan benar atau salah.

### 1.7.1 Bentuk 1

```
if (kondisi)  
pernyataan ;
```

Arti dari perintah if di atas adalah jika kondisi benar maka pernyataan akan dijalankan. Sedangkan kondisi ditulis di antara tanda kurung, dapat berupa ungkapan yang memiliki nilai benar atau salah. Dan pernyataan berupa sebuah pernyataan tunggal pernyataan majemuk atau pernyataan kosong.

Contoh Bentuk 1	
Algoritma	C++
<pre>Program coba_if kamus     tot_pembelian, diskon : real  algoritma     input(tot_pembelian)     diskon ← 0     if (tot_pembelian &gt;= 100000) then         diskon ← 0.05 * tot_pembelian     endif     output(diskon) endprogram</pre>	<pre>/* contoh penggunaan 'if' */ #include &lt;iostream&gt; using namespace std; int main(){     double tot_pembelian, diskon;     cout&lt;&lt;"total pembelian: Rp";     cin&gt;&gt;tot_pembelian;     diskon = 0;     if(tot_pembelian &gt;= 100000)         diskon = 0.05*tot_pembelian;     cout&lt;&lt;"besar diskon = Rp" &lt;&lt;diskon; }</pre>

### 1.7.2 Bentuk 2

```
if (kondisi)  
pernyataan1 ;  
else  
pernyataan2 ;
```

Arti dari pernyataan *if-else* di atas adalah:

1. Jika kondisi benar, maka pernyataan1 dijalankan.
2. Jika kondisi salah, maka pernyataan2 yang akan dijalankan.

pernyataan1 dan pernyataan2 dapat berupa sebuah pernyataan tunggal, pernyataan majemuk ataupun pernyataan kosong. Bentuk ini dapat disederhanakan dengan kondisional ekspresi dengan bentuk umumnya sebagai berikut.

```
expr1 ? expr2 : expr3
```

Jika *expr1* benar maka *expr2* yang dijalankan, sedangkan jika salah maka *expr3* yang dijalankan. Untuk lebih jelasnya perhatikan contoh berikut.

Contoh Bentuk 2	
Algoritma	C++
Program coba_if kamus tot_pembelian, diskon : real algoritma input(tot_pembelian) diskon ← 0 if (tot_pembelian >= 100000) then diskon ← 0.05 * tot_pembelian else diskon ← 0 endif output(diskon) endprogram	<pre>/* contoh penggunaan 'if else' */ #include &lt;iostream&gt;  using namespace std; int main(){     double tot_pembelian, diskon;     cout&lt;&lt;"total pembelian: Rp";     cin&gt;&gt;tot_pembelian;     diskon = 0;     if(tot_pembelian &gt;= 100000)         diskon = 0.05*tot_pembelian;     else         diskon = 0;     cout&lt;&lt;"besar diskon = Rp" &lt;&lt;diskon; }</pre>
Penyederhanaan	
* contoh penggunaan 'simple if else' */ #include <iostream> using namespace std; int main(){ double tot_pembelian, diskon; cout<<"total pembelian: Rp"; cin>>tot_pembelian; diskon = (tot_pembelian >= 100000) ? 0.05*tot_pembelian : 0; cout<<"besar diskon = Rp" <<diskon; }	

### 1.7.3 Bentuk 3

Bentuk ketiga menggunakan pernyataan *switch*, merupakan pernyataan yang dirancang khusus untuk menangani pengambilan keputusan yang melibatkan banyak alternatif.

Bentuk umumnya:

```
switch (Variabel) {  
    case kondisi1: pernyataan1; break;  
    case kondisi2 : pernyataan2;  
    break;  
    default: pernyataan_n;  
    break;  
}
```

Pengujian pada *switch* akan dimulai dari kondisi1, kalau nilai kondisi1 cocok maka pernyataan1 dilakukan, bila tidak cocok akan diteruskan pada pengecekan pernyataan2. Bila tidak ditemukan kondisi yang cocok maka statement pada default akan dilakukan. Contoh penggunaan *switch*:

Algoritma	C++
<pre> Program coba_switch kamus     kode_hari : integer algoritma     input(kode_hari)     depend on (kode_hari)         kode_hari = 1:         kode_hari = 2:         kode_hari = 3:         kode_hari = 4:         kode_hari = 5:             output ("Hari Kerja")         kode_hari = 6:         kode_hari = 7:             output ("Hari Libur")         else:             output("kode salah!!!");     enddependon endprogram </pre>	<pre> #include &lt;iostream&gt;  using namespace std; int main(){     int kode_hari;     puts("Menentukan hari kerja/libur\n");     puts("1=Senin 3=Rabu 5=Jumat 7=Minggu ");     puts("2=Selasa 4=Kamis 6=Sabtu ");     cin&gt;&gt;kode_hari;     switch(kode_hari){         case 1:         case 2:         case 3:         case 4:         case 5:             cout("Hari Kerja");             break;         case 6:         case 7:             cout("Hari Libur");             break;         default:             cout("Kode masukan salah!!!!");     }     return 0; } </pre>

## 1.8 Perulangan

Apabila kita ingin menuliskan angka 1 s.d. 5 secara berurutan maka kita bisa saja menuliskan semua angka tersebut secara manual karena *range* yang ditulis masih kecil. Lain halnya apabila angka yang ingin kita tulis *range*-nya dari 1 s.d. 10000 apakah kita akan menulisnya secara manual 1, 2, 3, 4, ..., 10000? Tentu tidak. Diperlukan suatu cara yaitu perulangan (looping). Perulangan digunakan untuk mengefisiensikan waktu dan meringkas kode program dalam pengeksekusian sub-program yang sama. Hal yang terpenting dalam perulangan adalah harus ada kondisi berhenti.

### 1.8.1 Perulangan dengan *for* dan *while*

Perulangan *for* dan *while* biasa digunakan saat kondisi ekspresi terpenuhi. Jika tidak maka perulangan akan terhenti.

Di bawah ini merupakan bentuk umum perulangan *for*:

```
for (initialization; condition; increment/decrement)
    statement;
```

Bentuk *for* di atas ekuivalen dengan bentuk *while* di bawah ini:

```
initialization;
while (condition) {
    statement;
    increment/decrement;
}
```

Keterangan :

1. *initialization*: pernyataan untuk menyatakan keadaan awal dari variabel *control*.
2. *condition*: ekspresi relasi yang menyatakan kondisi untuk keluar dari perulangan.
3. *increment/decrement*: pengatur perubahan nilai variabel kontrol.

Berikut adalah contoh program perulangan dengan *for*:

Algoritma	C++
<pre>Program coba_for kamus     jum,i : integer algoritma     input(jum)     for i ← 0 to jum do         output("saya pintar\n")     endfor endprogram</pre>	<pre>#include &lt;iostream&gt; using namespace std; int main(){     int jum;     cout&lt;&lt;"jumlah perulangan: ";     cin&gt;&gt;jum;     for(int i=0; i&lt;jum; i++) {         cout&lt;&lt;"saya pintar\n";     }     return 0; }</pre>

Berikut adalah contoh program perulangan dengan *while*:

Algoritma	C++
<pre>Program coba_while kamus     jum,i : integer algoritma     i ← 1     input(jum)      while i&lt;=jum do         output("baris ke-",i ,"\n")         i ← i + 1     endwhile endprogram</pre>	<pre>#include &lt;iostream&gt; using namespace std; int main(){     int i=1;     int jum;     cout&lt;&lt;"masukan banyak baris: ";     cin&gt;&gt;jum;     while(i&lt;=jum) {         cout&lt;&lt;"baris ke-"&lt;&lt;i&lt;&lt;endl;         i++; //sama dengan i=i+1     }     return 0; }</pre>

### 1.8.2 Perulangan dengan *do ... while*

Pada dasarnya struktur perulangan *do...while* sama saja dengan struktur *while*, hanya saja pada proses perulangan dengan *while*, seleksi berada di *while* yang letaknya di atas sementara pada perulangan *do...while*, seleksi *while* berada di bawah batas perulangan. Jadi dengan menggunakan struktur *do...while* sekurang-kurangnya akan terjadi satu kali perulangan.

Bentuk umum perulangan *do...while* adalah sebagai berikut:

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

Algoritma	C++
<pre>Program coba_do_while kamus     jum,i : integer algoritma     i ← 0     input(jum)     repeat         output("baris ke-",i ,"\n")         i ← i + 1     until (i &gt;= jum) endprogram</pre>	<pre>#include &lt;iostream&gt; using namespace std; int main(){     int i = 1;     int jum;     Cin &gt;&gt; jum;     do{         cout &lt;&lt; "baris ke-" &lt;&lt;(i+1)&lt;&lt;endl;         i++;     } while(i&lt;jum);     return 0; }</pre>

## 1.9 Struktur

Struktur merupakan tipe data bentukan berupa kumpulan dari variabel yang dinyatakan dalam sebuah nama, setiap variabel bisa memiliki tipe yang berlainan. Struktur bisa digunakan untuk mengelompokkan beberapa informasi yang saling berkaitan menjadi satu kesatuan (dalam bahasa pascal, struktur disebut dengan *record*).

Bentuk umum pendeklarasian struktur:

```
struct nama_tipe_struktur {
    tipe field1;
    tipe field2;
    .
    tipe fieldN;
} variabel_struktur1 ... variabel_strukturN;
```

Mengakses elemen struktur menggunakan tanda dot atau (.):

```
variabel_struktur.nama_field
```

Contoh pendeklarasian:

Algoritma	C++
<pre>Type Type tanggal &lt;     Tanggal : integer;     Bulan : integer;     Tahun : integer; &gt; Type data_rekam &lt;     Nama : char[31];     tgl_lahir : tanggal; &gt; kamus     Info : data_rekam</pre>	<pre>struct tanggal{     int tanggal;     int bulan;     int tahun; };  struct data_rekam{     char nama[31];     tanggal tgl_lahir; };  data_rekam info</pre>

Contoh pengaksesan:

```
cout<<info.nama;
cout<< info.tgl_lahir.tanggal << info.tgl_lahir.bulan << info.tgl_lahir.tahun
<< endl;
```

Penggunaan struktur sering dikombinasikan dengan *array*. Contoh penggunaan struktur dengan *array* misalnya untuk menyimpan data siswa.

Algoritma	C++
<pre>program coba_array_struct Type     Type data &lt;         nama : char[40]         nilai : integer     &gt;      constant maks : integer = 5      kamus         siswa : data[maks]      algoritma         for i ← 0 to MAX-1 do             output("masukkan data ke - ",                 i +1, "\n")             output("nama = ")             input(siswa[i].nama)             output("nilai = ")             input(siswa[i].nilai)         endfor         output("\ndata siswa\n")         output("=====")</pre>	<pre>/* contoh array dan struktur */ #include &lt;iostream&gt; #define MAX 5 using namespace std; int main(){     int i;     struct data{         char nama[40];         int nilai;     };     data siswa[MAX];     for(i=0; i&lt;MAX; i++){         cout&lt;&lt;"masukkan data ke- "&lt;&lt;i+1&lt;&lt;endl;         cout&lt;&lt;"nama = ";         cin&gt;&gt;siswa[i].nama;         cout&lt;&lt;"nilai = ";         cin&gt;&gt;siswa[i].nilai;     }     cout&lt;&lt;"\ndata siswa\n";     cout&lt;&lt;"======";</pre>

<pre> for i ← 0 to MAX - 1 do     output("\n\n\tdata ke -", i+1)     output("\n\n\t\tnama = ", siswa[i].nama)     output("\n\n\t\tnilai ", siswa[i].nilai) endfor endprogram </pre>	<pre> for(i=0; i&lt;MAX; i++){     cout&lt;&lt;"\n\n\tdata ke-"&lt;&lt;i+1;     cout&lt;&lt;"\n\n\t\tnama =&lt;&lt;siswa[i].nama;     cout&lt;&lt;"\n\n\t\tnilai =&lt;&lt;siswa[i].nilai; } return 0; } </pre>
---	--

Tipe data struktur ini cukup rumit, tetapi sangat penting dalam pemrograman bahasa C++, terutama dalam struktur data. Untuk merepresentasikan data sebagian besar akan menggunakan struktur yang dikombinasikan dengan *array* maupun *pointer* (akan dibahas pada bab selanjutnya).

## 1.10 Blok Program

Setiap bahasa komputer disusun dengan struktur yang berbeda. Untuk dapat mengerti bagaimana membuat program maka kita harus dapat memahami struktur dari program tersebut terlebih dahulu. Struktur program dari bahasa C++ terdiri dari fungsi-fungsi, seperti berikut.

```

fungsi_lain() { // fungsi lain yang ditulis programmer
    /* bagian ini berisi statement */
}
main () { // fungsi utama
    /* bagian ini berisi statement */
}

```

Perhatikan contoh di bawah ini:

Algoritma	C++
<pre> Program coba1 kamus     celcius, farenheit : real  function ctof(in:celcius:real)→ real algoritma     input(celcius)     farenheit ← ctot(celcius)     output(celcius) endprogram  function ctot(in: celcius: real)→real algoritma     → celcius * 1.8 + 32 endfunction </pre>	<pre> #include &lt;iostream&gt; using namespace std; /*deklarasi fungsi */ float ctot(float celcius); int main() {     float celcius, fahrenheit;     cout&lt;&lt;"nilai celcius? ";     cin&gt;&gt;celcius;     /*hitung konversi*/     fahrenheit = ctot(celcius);     cout&lt;&lt;celcius&lt;&lt;" celcius adalah "     &lt;&lt;fahrenheit&lt;&lt;"Fahrenheit ="&lt;&lt;endl;     return 0; } /*pendefinisian fungsi*/ float ctot(float celcius){     return celcius * 1.8 + 32; } </pre>

*Output :*

```

Nilai celcius? 10
10.000000 celcius adalah 50.000000 fahrenheit

```

Pembahasan :

1. Komentar bebas dapat diletakkan di manapun dalam blok program, diapit tanda /\* dan \*/.

- Contoh dari program di atas: /\* program pertama kita \*/
2. File judul (*header file*) adalah file berisi prototype sekumpulan fungsi pustaka.
  3. Fungsi pustakanya sendiri terdapat pada file pustaka (*library file*).  
Contoh: Penggunaan fungsi pustaka `cout` dan `cin` harus menyertakan file judul `<iostream>` yang berisi prototype fungsi pustaka operasi `input` dan `output` serta menggunakan preprocessor `#include`
  4. Penulisan statement dalam bahasa C++ diakhiri dengan titik koma ( ; ).
  5. Semua variabel yang digunakan harus telah dideklarasikan dahulu.  
Contoh: float celcius, fahrenheit;

## 1.11 Latihan

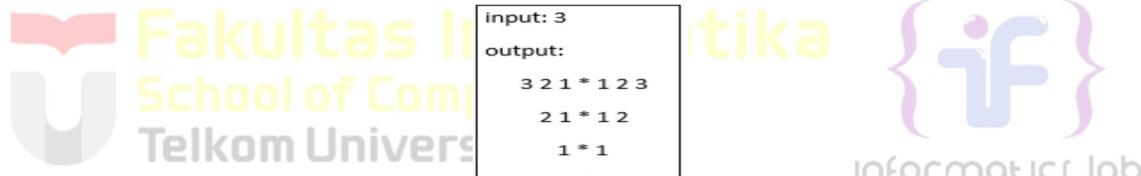
1. Buatlah program yang menerima *input*-an dua buah bilangan bertipe float, kemudian memberikan *output*-an hasil penjumlahan, pengurangan, perkalian, dan pembagian dari dua bilangan tersebut.
2. Buatlah sebuah program yang menerima masukan angka dan mengeluarkan *output* nilai angka tersebut dalam bentuk tulisan. Angka yang akan di-*input*-kan user adalah bilangan bulat positif mulai dari 0 s.d 100

contoh:

**79 : tujuh puluh Sembilan**

Gambar 1. 24

3. Buatlah program yang dapat memberikan *input* dan *output* sbb.



Gambar 1. 25 Mirror