

BAB 1

PENGANTAR BAHASA C/C++, TIPE DATA, VARIABEL, DAN PERINTAH KELUARAN (OUTPUT)

1.1 Tujuan Praktikum

1. Praktikan mengerti struktur bahasa C++
2. Praktikan mengenal Microsoft Visual Studio 2010
3. Praktikan mengetahui tipe data dan variabel dalam C++
4. Praktikan mampu mendeklarasikan tipe data dan variabel C++
5. Praktikan mengerti perintah keluaran di C++

1.2 Dasar Teori

1.2.1 Sejarah Bahasa C/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 diperkanggih dengan fasilitas kelas. Bahasa tersebut disebut C dengan kelas (*C with class*). Selama tahun 1983-1984, bahasa C dengan kelas disempurnakan dengan menambahkan fasilitas pembebanan lebih operator dan fungsi yang kemudian melahirkan apa yang disebut C++. Simbol ++ merupakan operator C untuk operasi kenaikan, simbol itu muncul untuk menunjukkan bahwa bahasa baru ini merupakan versi yang lebih canggih dari C.

1.2.2 Mengenal IDE Microsoft Visual Studio 2010

Dalam praktikum Algoritma dan Pemrograman ini kita menggunakan Microsoft Visual Studio 2010 (disingkat VS 2010) sebagai IDE-nya. IDE merupakan singkatan dari *Integrated Development Environment*, merupakan lembar kerja terpadu untuk pengembangan program.

VS 2010 merupakan sebuah perangkat lunak lengkap (*suite*) yang dapat digunakan untuk melakukan pengembangan aplikasi, baik itu aplikasi bisnis, aplikasi personal, ataupun komponen aplikasinya, dalam bentuk aplikasi *console*, aplikasi Windows, ataupun aplikasi Web. IDE dari VS 2010 C++ dapat digunakan untuk :

1. menulis naskah program;
2. mengompilasi program (*compile*);
3. melakukan pengujian terhadap program (*Debugging*); dll.

1.3 Menggunakan VS 2010

Untuk memulai membuat *project* baru pada VS 2010, pilih **File > New > Project** atau **Ctrl + Shift + N**. Untuk melakukan kompilasi kode program, pilih menu **Build > Compile** atau **Ctrl + F7**. Setelah itu untuk menjalankan program cukup tekan tombol **F5** pada keyboard.

Menu	Item	Shortcut	Penjelasan
Build	Build	Ctrl + F7	Meng- <i>compile</i> program
	Run + Debugging	F5	Menjalankan program sekaligus melakukan <i>Debugging</i>
	Run	Ctrl + F5	Menjalankan program tanpa melakukan <i>debugging</i>
	Rebuild All		Membangun kembali dengan meng- <i>compile</i> program

	<i>Clean</i>		Proses pembersihan terhadap kemungkinan adanya <i>bug</i>
	<i>Program reset</i>	Enter	Menutup program aktif, melepas memori, dan kembali ke IDE VS 2010
Project dan File	<i>New source file</i>	Ctrl + N	Membuat <i>source file</i> baru
	<i>New project</i>	Ctrl + Shift + N	Membuat <i>project</i> baru
	<i>Open file</i>	Ctrl + O	Membuka <i>file</i>
	<i>Open project</i>	Ctrl + Shift + O	Membuka <i>project</i>
	<i>Save</i>	Ctrl + S	Menyimpan <i>file/project</i> dengan nama yang sama
	<i>Save as</i>		Menyimpan <i>file/project</i> dengan nama yang berbeda
	<i>Save all</i>	Ctrl + Shift + S	Menyimpan semua <i>file/project</i> yang terbuka pada jendela IDE VS 2010
	<i>Close</i>		Menutup <i>file/project</i> yang sedang terbuka di jendela IDE VS 2010
	<i>Exit</i>	Alt + F4	Keluar dari program VS 2010

CATATAN! Untuk lebih jelasnya mengenai pemakaian VS 2010 dapat dilihat dibagian lampiran modul.

1.4 Struktur Program Pada C++

Program C maupun C++ secara umum tersusun dari beberapa bagian utama, yaitu :

a. Komentar

Komentar digunakan untuk memberi informasi/dokumentasi tentang program atau *code* yang ada. Dalam C atau C++ setiap tulisan yang diapit oleh simbol `/* ... */` untuk multi baris atau setiap baris yang dimulai dengan simbol `//` dianggap komentar dan tidak akan dikompilasi oleh *compiler*.

Contoh komentar :

```
/* ini adalah contoh
   komentar multibaris */
```

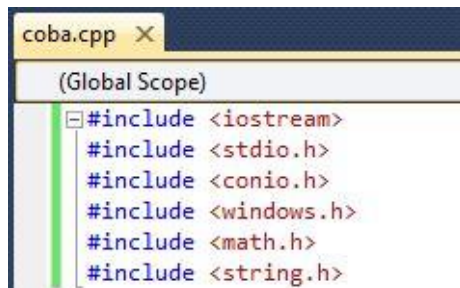
← Komentar multibaris

```
// ini adalah contoh
// komentar 1 baris
```

← Komentar 1 baris

b. *Preprocessor directive*

Preprocessor directive disebut juga pengarah *compiler* karena fungsinya untuk mengatur proses kompilasi.



The screenshot shows a code editor window titled 'coba.cpp'. The code content is as follows:

```
(Global Scope)
#include <iostream>
#include <stdio.h>
#include <conio.h>
#include <windows.h>
#include <math.h>
#include <string.h>
```

#include merupakan satu jenis pengarah *preprocessor* yang digunakan untuk membaca *file* yang dinamakan *file* judul (*header file*). **iostream**, **conio.h**, **stdio.h**, **windows.h**, **math.h**, dan **string.h** merupakan *file header* yang merupakan *standard library* dari C++. Setiap *file header* berhubungan dengan perintah masukan (*input*), perintah

keluaran (*output*), dan fungsi-fungsi yang digunakan dalam suatu program.

File header	Input	Output	Fungsi
<code>iostream</code>	<code>cin</code>	<code>cout</code>	
<code>conio.h</code>	<code>getchar</code> <code>getch</code> <code>getche</code>		<code>clrscr()</code>
<code>stdio.h</code>	<code>scanf</code> <code>gets</code>	<code>printf</code> <code>puts</code> <code>putchar</code>	
<code>windows.h</code>			<code>system()</code>
<code>math.h</code>			<code>sqrt()</code> <code>pow()</code> <code>log10()</code> <code>sin()</code>
<code>string.h</code>			<code>strcpy()</code> <code>strlen()</code> <code>strcat()</code>

Beberapa *file header* yang lainnya dapat Anda lihat di
C:\ProgramFiles\MicrosoftVisualStudio10.0\VC

c. Fungsi utama dan fungsi tambahan

Fungsi utama (*main*) harus ada dalam setiap program karena fungsi utama merupakan fungsi yang akan dieksekusi pertama kali. Lebih lanjut tentang fungsi akan dipelajari pada bab fungsi dipertemuan selanjutnya.

d. Bagian definisi fungsi

Diawali dengan tanda "{" (kurawal buka) sebagai tanda awal fungsi dan tanda "}" (kurawal tutup) sebagai tanda

berakhirnya suatu fungsi, baik fungsi utama maupun fungsi tambahan. Definisi fungsi berisi sekumpulan *code* yang nanti akan dieksekusi bila fungsi tersebut dipanggil. Aturan umum penulisan bahasa C++ adalah sebagai berikut.

- **case-sensitive**, yaitu bahasa C++ membedakan penulisan huruf besar dan huruf kecil, contoh $A \neq a$;
- setiap *statement* diakhiri dengan tanda ";" (titik koma);
- tidak boleh ada variabel ganda, maupun konflik dengan *reserved keywords* (kata-kata bawaan dari IDE), contoh :
`and, and_eq, asm, auto, bitand, bitor, bool, break, case, catch, char, class, compl, const, const_cast, continue, default, delete, do, double, dynamic_cast, else, enum, explicit, export, extern, false, float, for, friend, goto, if, inline, int, long, mutable, namespace, new, not, not_eq, operator, or, or_eq, private, protected, public, register, return, short, signed, sizeof, static, static_cast, struct, switch, template, this, throw, true, try, typedef, typeid, typename, union, unsigned, using, virtual, void, volatile, wchar_t, while, xor, xor_eq;`
- pada *subroutine* atau fungsi harus diapit oleh kurung kurawal (`{.....}`); dan
- setiap variabel yang digunakan **wajib** dideklarasikan terlebih dahulu.

e. Bagian deklarasi

Bagian yang akan mendeklarasikan variabel, konstanta, fungsi, dan lain-lain. Lebih lanjut tentang materi ini akan dibahas pada bab ini dan bab-bab selanjutnya.

1.5 Tipe Data

Tipe data adalah suatu jenis nilai yang dapat dinyatakan dalam bentuk konstanta atau variabel dan operator yang dapat digunakan untuk mendefinisikan objek data yang akan dimanipulasi dalam sebuah program. Jenis-jenis tipe data antara lain:

Jenis Tipe Data	Penjelasan	Contoh
Tipe data dasar (<i>primitive data type</i>)	Tipe data bawaan dari bahasa pemrograman	int, char, float, boolean
Tipe data bentukan (<i>defined data type</i>)	Tipe data bentukan <i>user</i>	struct, enum

Beberapa jenis **tipe data dasar** adalah :

a. Tipe Data Bilangan/Angka

Tipe data bilangan/angka dapat berupa **int** (untuk bilangan bulat/integer) atau **float** dan **double** (untuk bilangan desimal/real). Tiap tipe data tersebut memiliki jangkauan nilai yang berbeda-beda. Tipe data int, float, dan double memiliki tambahan yang dapat mengubah batas atas maupun batas bawah dari rentang konstanta yang dapat ditampung oleh tipe tersebut. Tambahan yang dapat dipergunakan adalah:

- signed dan unsigned

- short dan long

Berikut adalah **macam-macam tipe data angka** :

Tipe Data	Ukuran (bit)	Jangkauan Nilai	Keterangan
int	16 bit	-32768 s/d 32768	Bilangan bulat
short int atau short	16 bit	-32768 s/d 32768	Bilangan bulat
long int atau long	32 bit	-2147483648 s/d 2147483648	Bilangan bulat dengan rentang yang lebih luas
unsigned int	16 bit	0 s/d 65535	Bilangan bulat positif
unsigned short	16 bit	0 s/d 65535	Bilangan bulat positif
unsigned long	32 bit	0 s/d 4294967295	Bilangan bulat positif dgn rentang yang lebih luas
float	32 bit	3.4×10^{-38} s/d 3.4×10^{38}	Bilangan real
double	64 bit	1.7×10^{-308} s/d 1.7×10^{308}	Bilangan real dengan rentang yang lebih luas
long double	80 bits	3.4×10^{-4932} s/d 3.4×10^{4932}	Bilangan real dengan rentang terluas

CATATAN! Rentang nilai ini **berbeda** antara satu bahasa pemrograman dengan bahasa pemrograman yang lainnya dan rentang nilai ini juga dipengaruhi oleh *hardware* yang digunakan.

b. Tipe Data Teks

Pada tipe data teks menggunakan *keyword* “**char**”. Teks dapat berupa **satu karakter (char)** ataupun **kumpulan karakter (string)**. Karakter bisa berupa huruf alfabet, angka, tanda baca, operator aritmatik (+, -, /, *) dan karakter khusus (@, &, \$, #, dll).

Dalam penulisannya, untuk teks yang berupa **1 karakter** caranya yaitu karakter ditulis di antara **dua petik tunggal ('.')**. Sedangkan penulisan teks dengan karakter lebih dari satu ditulis di antara **dua petik ganda (“...”)**. Pendeklarasiannya pun diikutsertakan panjang dari nilai string tersebut.

CATATAN! Perlu diingat, pada tipe data char, data **tidak dapat** diolah secara matematis.

c. Tipe Data Boolean

Tipe data boolean adalah tipe data hanya mempunyai **2 kondisi benar (“true”) atau salah (“false”)**. Untuk kondisi **benar bernilai 1** dan kondisi **salah bernilai 0**.

1.6 Variabel

Variabel menyatakan objek data yang nilainya disimpan dan dapat berubah-ubah nilainya selama eksekusi berlangsung. Syarat-syarat penulisan nama variabel adalah :

- a. nama variabel **tidak boleh sama** dengan **reserved keywords, function (fungsi)**, dan harus unik;
- b. **maksimum 32 karakter**, jika lebih maka karakter selebihnya tidak akan diperhatikan oleh komputer;

- c. nama variabel bersifat **case-sensitive**;
- d. nama variabel tersusun dari **huruf, angka, dan "_" (underscore)**;
- e. nama variabel **harus** diawali **huruf alfabet** atau **"_" (underscore)**, **tidak boleh** diawali dengan **angka** atau **karakter khusus lainnya**; dan
- f. nama variabel **tidak boleh mengandung spasi**, jika ada pemisahan karakter maka spasi pemisah dapat diganti dengan karakter **"_" (underscore)**.

1.7 Pendeklarasian Variabel

Bentuk umum pendeklarasian variabel adalah :

```
type_data<spasi>nama_variabel;
```

Contoh pendeklarasian variabel:

```
int umur;
```

tipe data ← int umur → nama variabel

Beberapa contoh **pendeklarasian variabel berdasarkan tipe datanya** :

a. Tipe Data Bilangan/Angka

Contoh :

```
int umur = 28;
```

```
float ipk = 3.47;
```

b. Tipe Data Teks

Contoh teks **1 karakter (char)**:

```
char nilai = 'A';
```

```
char Golongandarah = 'B';
```

Contoh teks yang **lebih dari 1 karakter (string)**:

```
char nama [10] = "shinta";
```

nama sebagai variabel yang berisi **string** dengan
panjang maksimal 10 karakter

c. Tipe Data Boolean

Contoh :

```
bool kondisi = true;
```

```
bool keadaan = false;
```

Berikut merupakan beberapa contoh pendeklarasian variabel yang benar dan yang salah:

a. Pendeklarasian yang BENAR

- ```
int umur = 17;
```

Pendeklarasian variabel *umur* langsung dengan  
**pemberian nilai (*inisialisasi*)**, yaitu 17.

- ```
int berat_badan;
```

Pendeklarasian variabel *berat badan* dengan **menggunakan tanda “_” (*underscore*) sebagai pemisah antara kata “berat” dan “badan”.**

- ```
char Golongandarah = 'O' ;
```

Pendeklarasian variabel *golongan darah* dengan **menggunakan huruf kapital.**

- ```
int nilai_TUGAS_1 = 100;
```

Pendeklarasian variabel *nilai tugas 1* dengan **mengkombinasikan huruf kapital, huruf kecil, “_” (*underscore*), dan angka.**

- ```
bool _1kondisi = true;
```

Pendeklarasian variabel *1 kondisi* dengan **menggunakan “\_” (*underscore*) untuk mengawali penamaan variabel karena nama variabel tersebut diawali dengan angka 1.**

#### **b. Pendeklarasian variabel yang SALAH**

- ```
int umur = 0.17;
```

Pendeklarasian variabel *umur* tersebut salah karena **nilai yang dimasukkan ke dalam variabel tidak sesuai dengan tipe data yang digunakan.** Seharusnya jika ingin membuat variabel yang dapat menampung nilai berupa angka desimal menggunakan tipe data float atau

double. Jadi, pemberian nilai pada suatu variabel harus sesuai dengan tipe datanya.

- `int berat badan;`

Pendeklarasian variabel *berat badan* tersebut salah karena **menggunakan spasi antara kata "berat" dan "badan"**.

- `int nilai TUGAS_1 = 100;`

Pendeklarasian variabel *nilai tugas 1* tersebut salah karena **antara kata "nilai" dengan "TUGAS_1" mengandung spasi walaupun antara kata "TUGAS" dan "1" sudah menggunakan "_" (*underscore*).** Jadi sepanjang apapun penamaan suatu variabel tetap tidak diperbolehkan mengandung spasi.

- `bool lkondisi = true;`

Pendeklarasian variabel *1 kondisi* tersebut salah karena **penamaan variabel tersebut diawali dengan angka.**

- `char &nilai = 'A';`

Pendeklarasian variabel *& nilai* tersebut salah karena **penamaan variabel tersebut diawali dengan karakter khusus (&).**

1.8 Format Data

Format data adalah suatu kombinasi tanda-tanda khusus berupa karakter yang dipakai untuk mengatur bentuk data tampilan. Format data bisa lebih dari satu sesuai dengan banyak data yang ingin ditampilkan. Format data yang dipakai disesuaikan dengan jenis data tampilan.

Daftar Spesifikasi Format Data

Format Data	Bentuk Tampilan Data
%c	Karakter tunggal
%d	Bilangan bulat
%e	Bilangan bernotasi ilmiah
%f	Bilangan berjenis pecahan bertitik (<i>floating point number</i>)
%g	Pilihan tampilan terpendek antara bentuk %e atau %f
%i	Bilangan bulat desimal (sama dengan %d)
%n	Digunakan untuk perintah menyimpan banyak karakter yang telah ditampilkan oleh perintah print() kedalam suatu petunjuk jenis <i>integer</i>
%o	Notasi bilangan basis delapan (<i>oktal</i>)
%p	Tampilan petunjuk (<i>pointer</i>)
%s	Tampilan deret karakter (string)
%u	Bilangan bulat decimal tak bertanda (<i>unsign</i>)
%x	Notasi bilangan basis 16 (<i>hexadesimal</i>)
%%	Tampilan tanda persen (%)

1.9 Perintah Keluaran (*Output*)

Perintah keluaran (*output*) berfungsi untuk menampilkan sesuatu ke layar *console* program. Hal ini digunakan untuk menambah interaktifitas *console* program Anda dengan *user*. Ada beberapa jenis perintah *output* pada **bahasa C**, antara lain :

a. **printf()**

Perintah *output* yang ada pada *file header* "**stdio.h**" yang paling umum digunakan.

Sintaks :

```
printf("format_data,arg1,arg2,...) ;
```

Keterangan :

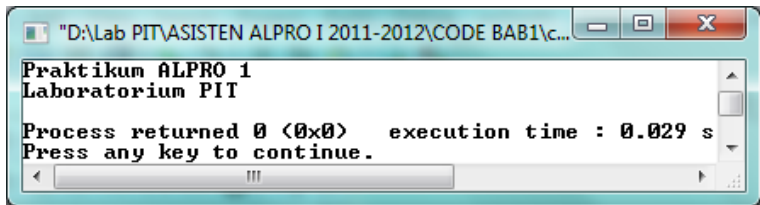
1. **format_data** berupa keterangan jenis data apa yang akan ditampilkan ke layar.
2. **arg1, arg2, dst** merupakan argumen yang dapat berupa variabel, konstanta atau ekspresi.

Contoh 1 penggunaan `printf()` :

```
#include <stdio.h>
#include <Windows.h>

int main() {
    printf("Praktikum ALPRO 1\n");
    printf("Laboratorium PIT\n");
    system("pause");
    return 0;
}
```

Bila program tsb dijalankan, maka akan muncul hasil sbb :



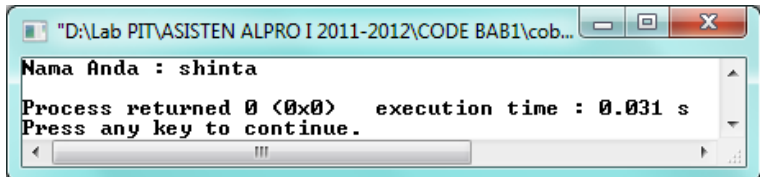
```
"D:\Lab PIT\ASISTEN ALPRO I 2011-2012\CODE BAB1\c...
Praktikum ALPRO 1
Laboratorium PIT
Process returned 0 (0x0) execution time : 0.029 s
Press any key to continue.
```

Contoh 2 penggunaan `printf()`:

```
#include <stdio.h>
#include <Windows.h>

int main(){
    char nama[10] = "shinta";
    printf("Nama Anda : %s\n",nama);
    system("pause");
    return 0;
}
```

Bila program tsb dijalankan, maka akan muncul hasil sbb :



```
"D:\Lab PIT\ASISTEN ALPRO I 2011-2012\CODE BAB1\cob...
Nama Anda : shinta
Process returned 0 (0x0) execution time : 0.031 s
Press any key to continue.
```

b. puts()

Digunakan untuk **mencetak string** ke layar, dimana pencetakan akan **diakhiri dengan karakter *newline*** (baris baru). Perintah keluaran ini terdapat dalam *file header* "**stdio.h**".

Sintaks :

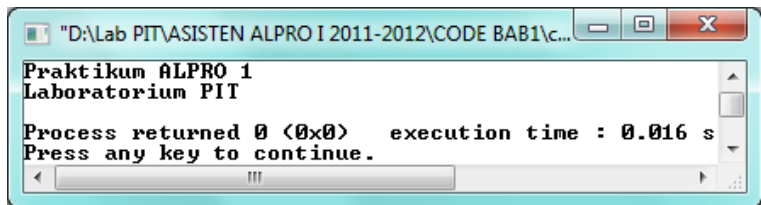

```
puts(<string yang ditampilkan>);
```

Contoh penggunaan `puts()` :

```
#include <stdio.h>
#include <Windows.h>

int main() {
    puts("Praktikum ALPRO 1");
    puts("Laboratorium PIT");
    system("pause");
    return 0;
}
```

Bila program tsb dijalankan, maka akan muncul hasil sbb :



C. `putchar()`

Digunakan untuk menampilkan **sebuah karakter** ke layar. Pencetakan karakter **tidak diakhiri dengan karakter *new line***, terdapat dalam *file header* "**stdio.h**".

Sintaks :

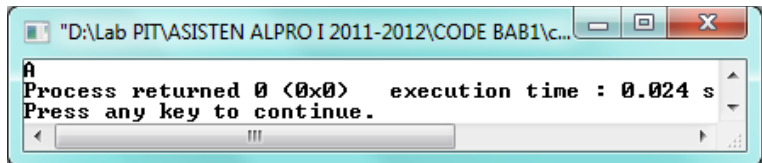
```
putchar (<karakter>);
```

Contoh penggunaan `putchar()` :

```
#include <stdio.h>
#include <Windows.h>

int main() {
    putchar('A');
    system("pause");
    return 0;
}
```

Bila program tsb dijalankan, maka akan muncul hasil sbb :



Sedangkan pada **bahasa C++** perintah keluarannya sbb :

d. cout

Digunakan untuk mencetak sesuatu ke layar, baik berupa **karakter maupun string**. Terdapat pada *file header* "**iostream**", dan menggunakan "**endl**" untuk ganti baris. Sintaks :

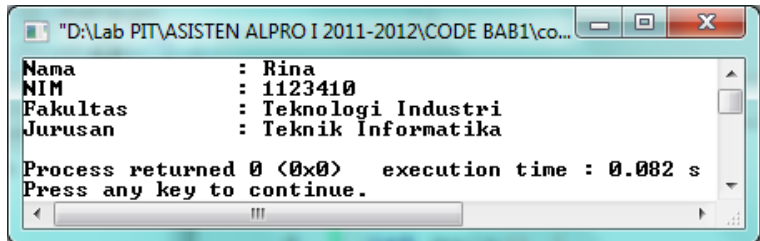
```
cout<<[apa yg ditampilkan]<<endl;
```

Contoh penggunaan cout:

```
#include <iostream>
#include <Windows.h>
using namespace std;
int main() {
    cout<<"Nama \t\t: Rina"<<endl;
    cout<<"NIM \t: 1123410"<<endl;
}
```

```
cout<<"Fakultas \t: Teknologi Industri"<<endl;
cout<<"Jurusan \t: Teknik Informatika"<<endl;
system("pause");
return 0;
}
```

Bila program tsb dijalankan, maka akan muncul hasil sbb :



1.10 Langkah Praktikum

1. Buat program seperti di bawah ini! Simpan dengan nama **LatihanOutput1.cpp!**

Pseudocode

Program LatihanOutput1

{Tanpa masukan, hanya menampilkan angka, kata, dan kalimat yang dimasukkan ke dalam variabel}

Kamus Lokal :

```
nama          : string
umur          : integer
tinggi_badan  : double
ipk           : float
gol_darah    : char
jurusan       : string
```

Algoritma :

```
nama ← "Shinta Dewi Ratnasari"
umur ← 20
tinggi_badan ← 153.5
```

```
ipk ← 3.82
gol_darah ← 'O'
jurusan ← "Teknik Informatika"

output ("Nama : ", nama)
output ("Umur : ", umur)
output ("Tinggi : ", tinggi_badan)
output ("Goldar : ", gol_darah)
output ("Jur : ", jurusan)
output ("IPK : ", ipk)
```

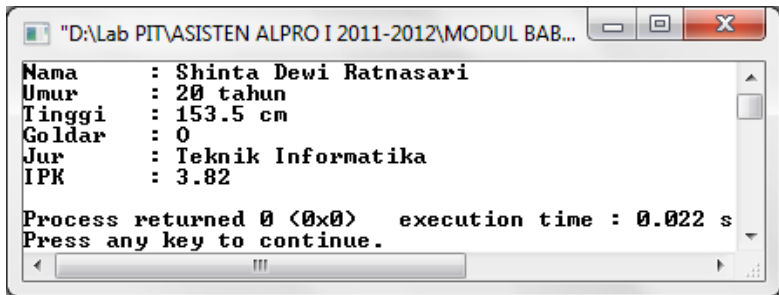
Source Code Program

```
#include <iostream>
#include <stdio.h>
#include <Windows.h>
using namespace std;
int main() {
    //pendeklarasian variabel dan inisialisasi
    char nama[30] = "Shinta Dewi Ratnasari";
    int umur = 20;
    double tinggi_badan = 153.5;
    float ipk = 3.82;
    char gol_darah = 'O';
    char jurusan[30] = "Teknik Informatika";

    //menampilkan ke layar
    printf("Nama \t : "); puts(nama);
    printf("Umur \t : %d tahun\n", umur);
    printf("Tinggi \t : %g cm\n", tinggi_badan);
    cout<<"Goldar \t : "; putchar(gol_darah);
    cout<<endl;
    cout<<"Jur \t : "<<jurusan<<endl;
    cout<<"IPK \t : "<<ipk<<endl;

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

Screen Shoot Program



A screenshot of a Windows command prompt window. The title bar reads '"D:\Lab PIT\ASISTEN ALPRO I 2011-2012\MODUL BAB...'. The window contains the following text:
Nama : Shinta Dewi Ratnasari
Umur : 20 tahun
Tinggi : 153.5 cm
Gol dar : 0
Jur : Teknik Informatika
IPK : 3.82

Process returned 0 (0x0) execution time : 0.022 s
Press any key to continue.

2. Buat program seperti di bawah ini! Simpan dengan nama **LatihanOutput2.cpp!**

Pseudocode

Program LatihanOutput2

{Menampilkan hasil atau nilai dari suatu variabel
bertipe data BOOLEAN}

Kamus Lokal :

kondisi : **boolean**

Algoritma :

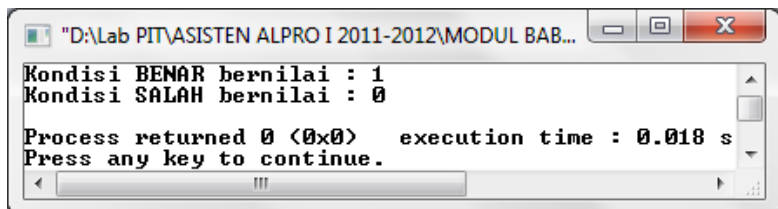
kondisi ← true
output (kondisi)
kondisi ← false
output (kondisi)

Source Code Program

```
#include <iostream>
#include <Windows.h>
using namespace std;
int main() {
    bool kondisi; //mendeklarasikan variabel
    //menampilkan jika kondisi bernilai BENAR
    kondisi = true;
    cout<<"Kondisi BENAR bernilai : ";
    cout<<kondisi <<endl;
```

```
//menampilkan jika kondisi bernilai SALAH  
kondisi = false;  
cout<<"Kondisi SALAH bernilai : ";  
cout<<kondisi <<endl;  
system("pause");  
return 0;  
}
```

Screen Shoot Program



Bab II

Konstanta, Operator, dan Debugging

BAB II

KONSTANTA, OPERATOR DAN *DEBUG*

2.1 Tujuan Praktikum

1. Praktikan mengetahui konstanta dan operator
2. Praktikan mampu mendeklarasikan konstanta dan operator
3. Praktikan mengerti cara melakukan *Debugging*

2.2 Dasar Teori

2.2.1 Konstanta

Konstanta adalah nilai numeris ataupun karakter yang **tetap/tidak berubah** selama program berlangsung. Penggunaan konstanta pada bahasa C++ dan 2 cara yaitu :

```
#define phi 3.14  
#define nama "helmi"
```

atau

```
const float pi= 3.14;  
const char nama = "helmi";
```

Khusus untuk konstanta karakter selalu ditulis dengan diawali dan diakhiri dengan tanda petik tunggal ('). Beberapa konstanta karakter dapat ditulis dengan diawali tanda \ (penempatan tanda \ setelah tanda petik tunggal). Karakter yang berawalan tanda \ disebut rangkaian karakter *escape (escape sequence)*.

Sequence	Karakter	Keterangan
\a	Bel	Bunyi bel
\b	BS	Mundur 1 karakter (<i>backspace</i>)
\f	FF	Ganti halaman (<i>form feed</i>)
\n	LF	Ganti baris (<i>line feed</i>)
\r	CR	Kembali ke awal kolom (<i>carriage return</i>)
\t	HT	Tabulasi horizontal (<i>horizontal tab</i>)
\v	VT	Tabulasi vertical (<i>vertikal tab</i>)
\	\	Backslash
\'	'	Petik tunggal
\"	"	Petik ganda
\?	?	Tanda tanya
\DDD	apapun	DDD adalah digit nilai octal
\xHHH	apapun	HHH adalah digit nilai hexadecimal
\0	Null	Karakter ASCII=0

2.2.2 Operator

Operator merupakan simbol yang digunakan dalam suatu program untuk melakukan suatu operasi atau manipulasi, misalnya untuk :

- menjumlahkan dua nilai;
- memberikan nilai ke suatu variabel (*assignment*); dan
- membandingkan kesamaan dua nilai.

a. Operator pengerjaan

Operator pengerjaan berupa tanda **sama dengan (=)** yang digunakan untuk mengisi nilai yang berada di sebelah kanannya variabel yang ditunjuk disebelah kirinya.

Contoh :

```
a = 7
a = b = c = 7
```

maka **nilai a dan b akan sama dengan nilai c yaitu 7.**

b. Operator Aritmatika

Operator ini digunakan untuk **perhitungan dasar aritmatika**. Operator aritmatika ini digolongkan menjadi dua yaitu **operator unary dan operator binary**.

Operator unary yaitu operator yang dikenakan terhadap **satu buah nilai (operand)**.

Contoh :

```
c = -b
c = -a
```

Tabel operator aritmatika unary

Operator	Keterangan
+	Tanda plus
-	Tanda minus
++	Prefix :preincrement; postfix: postincrement
--	Prefix :predecrement; postfix: postdecrement

Operator peningkatan dan penurunan

Bahasa C++ mempunyai operator unik, yaitu **operator peningkatan/increment (++)** dan **operator penurunan/decrement (--)**. Operator peningkatan (++) digunakan untuk menambahkan nilai 1 pada nilai sebelumnya dan operator

penurunan (--) digunakan untuk mengurangi nilai 1 pada nilai sebelumnya.

Operator binary yaitu operator yang dikenakan terhadap **dua buah nilai (operand)**.

Contoh :

```
c = a + b
c = a * b
```

Tabel operator aritmatika binary

Operator	Keterangan
*	Perkalian
/	Pembagian
%	Modulus atau sisa pembagian
+	Penjumlahan
-	Pengurangan

Operator aritmatika pada C++ mempunyai **urutan prioritas atau hierarki** tertentu, adapun urutan prioritas adalah sbb:

Tertinggi	++ --
↑ ↓	- (unary minus)
	* / %
Terendah	+ -

Operasi parenthesis adalah operasi di dalam tanda **kurung ()** yang akan dikerjakan terlebih dahulu. Apabila bersarang,

yang dikerjakan pertama kali adalah tanda () yang paling dalam. Apabila terdapat tanda () satu level (tidak bersarang) dikerjakan dari kiri sampai kanan.

c. Operator hubungan

Operator hubungan digunakan untuk **membandingkan dua elemen nilai** dan akan dihasilkan nilai perbandingan, yaitu betul (bernilai 1) atau salah (bernilai 0). Operator ini banyak diaplikasikan pada penyelesaian suatu kondisi di *statement* IF.

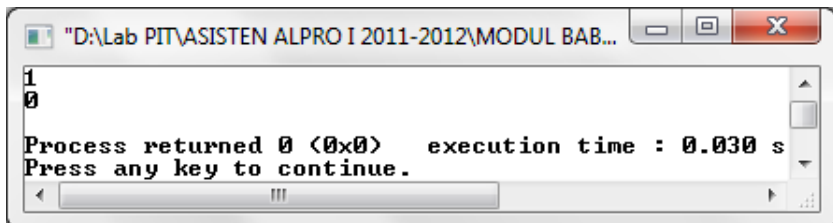
Operator	Keterangan
>	Lebih besar dari
>=	Lebih besar dari sama dengan
<	Lebih kecil dari
<=	Lebih kecil dari sama dengan
==	Sama dengan
!=	Tidak sama dengan

Contoh :

```
#include <iostream>
#include <Windows.h>
using namespace std;

int main() {
    int nilai = 90;
    cout<<(nilai>80)<<endl;
    cout<<(nilai<80)<<endl;
    system("pause");
    return 0;
}
```

Bila program tsb dijalankan, maka akan muncul hasil sbb :



d. Operator logika

Operator logika bisaa digunakan untuk **menghubungkan dua buah ungkapan kondisi menjadi sebuah ungkapan kondisi.**

Tabel Operator Logika

Operator	Keterangan
&&	AND (dan)
	OR (atau)
!	NOT (bukan)

Bentuk umum pemakaian operator logika && dan || :

```
<ungkapan1>&&<ungkapan2>;
<ungkapan1> || <ungkapan2>;
```

Bentuk umum pemakaian operator logika ! :

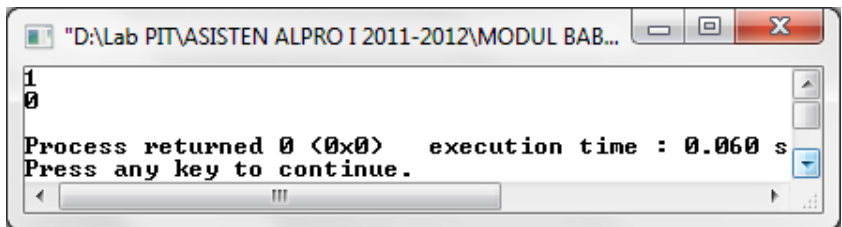
```
!<ungkapan>;
```

Contoh :

```
#include <iostream>
#include <Windows.h>
using namespace std;

int main() {
    int nilai = 90;
    cout<< ( (nilai>80) && (nilai<100) ) << endl;
    cout<< ( (nilai>80) && (nilai>100) ) << endl;
    system("pause");
    return 0;
}
```

Bila program tsb dijalankan, maka akan muncul hasil sbb :

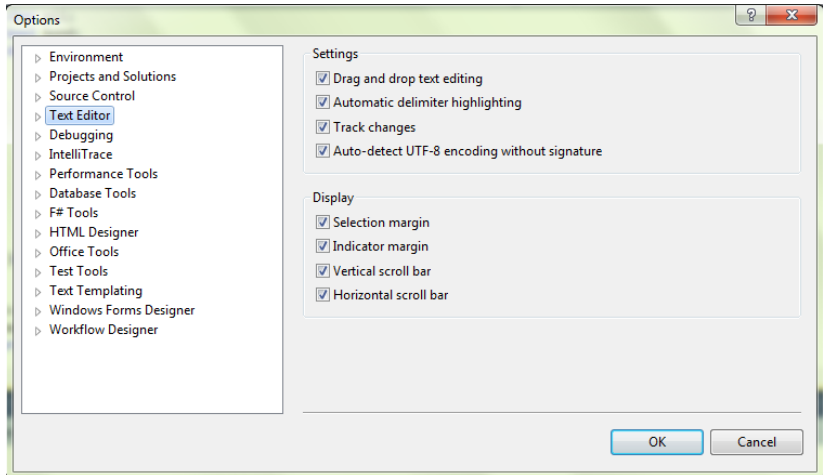


2.3 Debug

Debugging adalah proses menemukan atau melacak kesalahan pada kode sebuah program. Versi *debug ++* digunakan untuk memudahkan programmer dalam melakukan proses *Debugging* atau melacak kesalahan pada kode program. Pada versi *debug* ini akan memberikan informasi-informasi yang berguna saat akan dilakukan *Debugging*. Versi *debug* juga bisa disebut sebagai *debugger* pada Microsoft Visual C++.

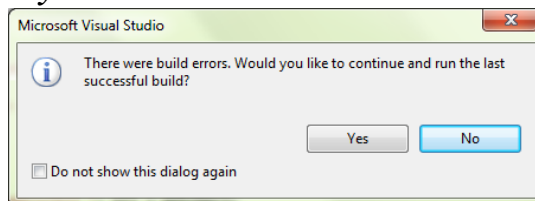
Sebelum memulai proses *Debugging* program menggunakan VS 2010 ada baiknya apabila fitur **line numbers** diaktifkan terlebih

dahulu. Hal ini untuk mempermudah *programmer* dalam mencari baris kode yang bermasalah apabila program gagal dieksekusi.



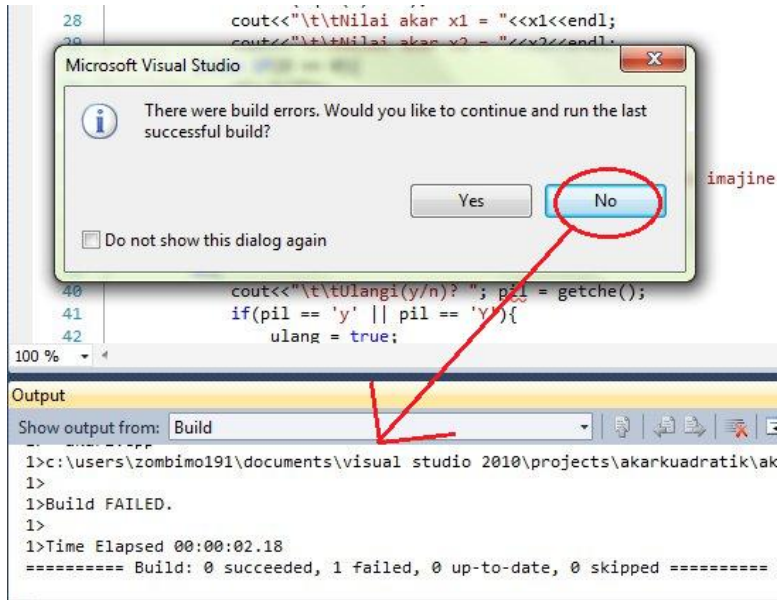
Pada kotak sebelah kiri expand bagian **Text Editor** kemudian pilih **All Languages | General** dan pada bagian **Display** disebelah kanan centang fitur **Line numbers**. Untuk memulai proses *Debugging* sama dengan cara menjalankan program yaitu cukup dengan menekan tombol **F5** pada keyboard. Berikut adalah contoh proses *Debugging* dengan metode **try and error** dan **hasil yang tidak sesuai**.

a. **try and error**




Apabila muncul pesan seperti di atas, maka dapat dipastikan terdapat kesalahan pada kode program yang ditulis. Untuk
Laboratorium Pemrograman dan Informatika Teori
Jurusan Teknik Informatika – Universitas Islam Indonesia

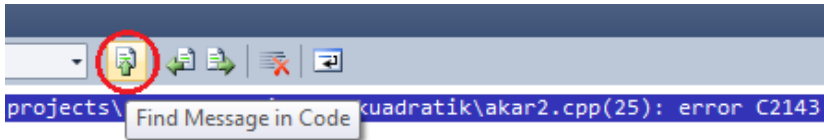
mengatasinya klik **No** kemudian pindah kebagian *Output* yang berada di bawah *code editor* seperti berikut:




Cari teks yang memberikan informasi tentang *error* yang dimaksud seperti tampak pada contoh di bawah ini :



Angka **(25)** menunjukkan baris yang *error* dan **sintaks error : missing ';' before 'if'** menunjukkan jenis *error*-nya. Selanjutnya klik pada bagian teks informasi *error* dan klik icon  untuk berpindah kebagian yang *error* pada *code editor*.



```
te, 0 skipped =====
```

Di bawah ini adalah tampilan bagian yang *error* pada *code editor*. Perhatikan bahwa yang sesungguhnya mengalami kesalahan penulisan bukanlah pada baris ke-25 tetapi pada baris ke-23 yang ditunjukkan oleh tanda .

```
22 |  
23 | D=pow(b,2)-4*a*c  
24 |  
25 | if (D > 0) {  
26 |     x1=-b+(sqrt(D)/2*a);  
27 |     x2=-b-(sqrt(D)/2*a);  
28 |     cout<<"\t\tNilai akar x1 = "<<x1<<endl;  
29 |     cout<<"\t\tNilai akar x2 = "<<x2<<endl;
```

← Tidak ada tanda ';'.

Hal ini sesuai dengan pesan informasi kesalahan yang ditunjukkan pada bagian *output* yaitu tidak ada tanda ';' sebelum 'if'. Jika sudah ditemukan bagian mana yang *error* maka untuk memperbaikinya tambahkan tanda ';' dibagian yang dimaksud.

b. hasil yang tidak sesuai

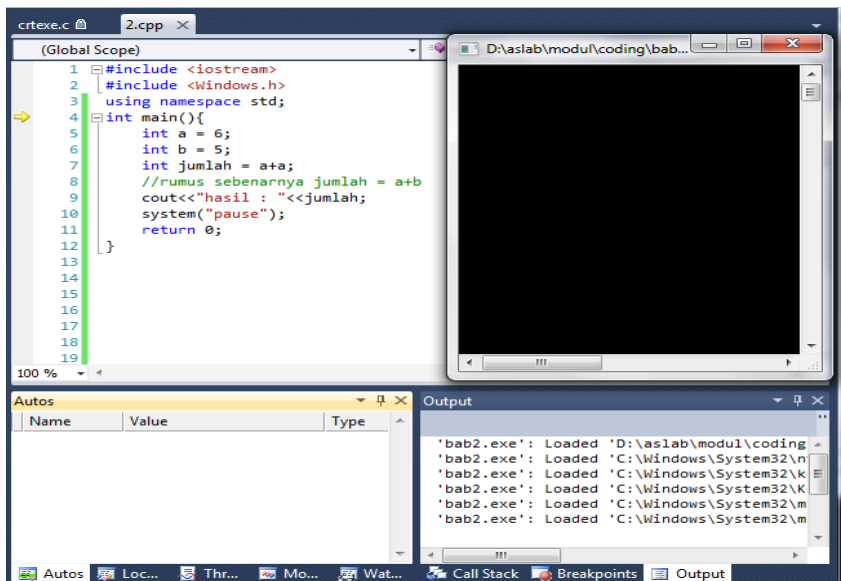
Dalam proses debugging kita juga bisa melakukan eksekusi pada program, dengan tujuan untuk mencoba apakah program tersebut sama seperti yang kita harapkan atau tidak (sama dengan perhitungan manual yang telah kita lakukan), karena pada debugging dengan metode seperti ini program akan melakukan

pengecekan pada setiap baris, dan kita akan mengetahui jalanya program tersebut baris per baris.

Cobalah untuk mengetik program dibawah ini. lalu kita coba dengan metode tersebut.

```
#include <iostream>
#include <Windows.h>
using namespace std;
int main(){
    int a = 6;
    int b = 5;
    int jumlah = a+a;
    //rumus sebenarnya adalah jumlah = a+b
    cout<<"hasil : "<<jumlah;
    system("pause");
    return 0;
}
```

Untuk menggunakan metode tersebut, running program menggunakan **f10**. Dibawah ini ketika program dijalankan.



Setelah kita menekan tombol **f10** muncul jendela CMD dan jendela **Autos** pada Ms VS 2010. Disitu bisa kita lihat program berjalan pada baris ke-4, ketika kita menekan tombol **f10** program akan berjalan ke baris 5.

Autos		
Name	Value	Type
a	-858993460	int

Setelah kita menekan **f10**, pada window Autos mengalami perubahan. Tipe data dan variable pada baris ke-5 terbaca. Dengan value yang belum terlihat jelas. Selanjutnya kita menekan **f10** untuk menuju baris selanjutnya.

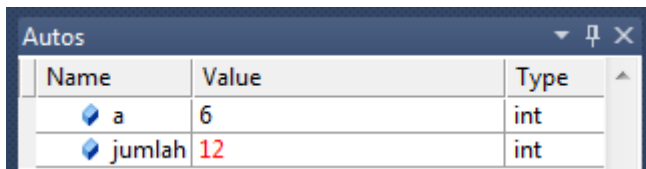
Autos		
Name	Value	Type
a	6	int
b	-858993460	int

Setelah kita menekan **f10** lagi, pada window Autos mengalami perubahan kembali. Untuk variable A sudah kelihatan nilainya dan penambahan satu tipe data dan variable baru yaitu b. Selanjutnya kita menekan **f10** untuk menuju baris selanjutnya.

Autos		
Name	Value	Type
a	6	int
b	5	int
jumlah	-858993460	int

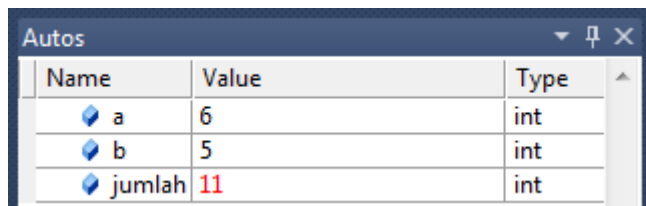
Setelah kita menekan **f10** lagi, pada window Autos mengalami perubahan kembali. Untuk variable A dan B sudah kelihatan nilainya

dan penambahan satu tipe data dan variable baru yaitu jumlah. Selanjutnya kita menekan **f10** untuk menuju baris selanjutnya.



Name	Value	Type
a	6	int
jumlah	12	int

Setelah melewati baris ke-7 yang berisi **int jumlah = a+a;** nilai jumlah pada window autos berubah menjadi 12. Bisa dilihat pada window tersebut variable yang tidak terpakai pada baris ke-7 akan hilang. Dan hasil eksekusi pada baris ke-7 akan muncul sebagai nilai dari variable jumlah. Jika kita melihat rumus yang benar hasil perhitungannya seharusnya adalah 11. Disitu dapat disimpulkan bahwa terjadi kesalahan pada baris ke-7. Dibawah ini hasil dari debugging jika menggunakan rumus **int jumlah = a+b;**



Name	Value	Type
a	6	int
b	5	int
jumlah	11	int

Semua variable yang digunakan akan terbaca, dan nilai untuk variable jumlah adalah 11.

2.4 Langkah Praktikum

1. Buatlah program untuk menampilkan hasil penjumlahan operator aritmatika menggunakan *increment* dan *decrement*, simpan dengan nama **aritmatika.cpp!** , serta lakukanlah pengujian debugging menggunakan metode **hasil yang tidak sesuai.**

Berikut bentuk dari *pseudocode*-nya:

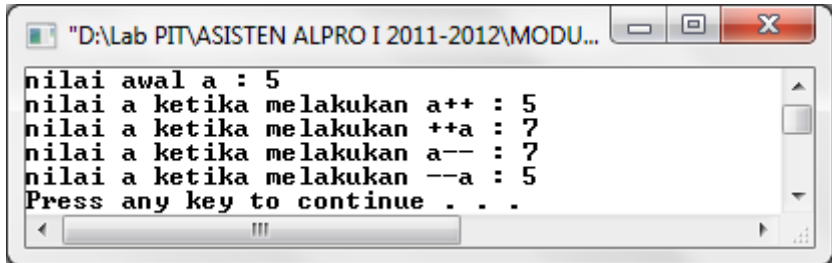
<p><u>Program</u> aritmatika <i>{Program untuk menampilkan hasil penjumlahan operator aritmatika dengan menggunakan increment dan decrement}</i></p>
<p><u>Kamus Lokal :</u> a : <u>integer</u></p>
<p><u>Algoritma :</u> a ← 5 <u>output</u> ("nilai awal a : ", a) <u>output</u> ("nilai a ketika melakukan a++ : ", a++) <u>output</u> ("nilai a ketika melakukan ++a : ", ++a) <u>output</u> ("nilai a ketika melakukan a-- : ", a--) <u>output</u> ("nilai a ketika melakukan --a : ", --a)</p>

Di bawah ini sintaks program untuk menjalankan kasus di atas :

```
#include <iostream>
#include <Windows.h>
using namespace std;

int main(){
    int a = 5;
    cout<<"nilai awal a : "<<a<<endl;
    cout<<"nilai a ketika melakukan a++ : ";
    cout<<a++ <<endl;
    cout<<"nilai a ketika melakukan ++a : ";
    cout<<++a <<endl;
    cout<<"nilai a ketika melakukan a-- : ";
    cout<<a--<<endl;
    cout<<"nilai a ketika melakukan --a : ";
    cout<<--a <<endl;
    system("pause");
    return 0;
}
```

Hasil *screen shoot* dari program di atas :



```
"D:\Lab PIT\ASISTEN ALPRO I 2011-2012\MODU...
nilai awal a : 5
nilai a ketika melakukan a++ : 5
nilai a ketika melakukan ++a : 7
nilai a ketika melakukan a-- : 7
nilai a ketika melakukan --a : 5
Press any key to continue . . .
```

2. Ketikkan sintaks program di bawah ini kemudian *debug*-lah program tersebut dan carilah kesalahan dari program tersebut!

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

int main(){
    int a == 9;
    int b == 0;
    float jumlah;
    jumlah == a/b;
    cout<<jumlah;

    return 5;
    system("pause");
}
```

Bab III

Perintah Masukan (Input) dan Flowchart



BAB III

PERINTAH MASUKAN (*INPUT*) DAN *FLOWCHART*

3.1 Tujuan Praktikum

1. Praktikan mengetahui operator serta perintah masukan yang digunakan dalam bahasa pemrograman C++
2. Praktikan mengerti bentuk-bentuk komponen *flowchart*
3. Praktikan memahami konsep masukan dan implementasinya

3.2 Dasar Teori

Agar bisa berinteraksi dengan pengguna, suatu program memerlukan suatu alat sebagai perantara antara pengguna dengan program tersebut. Misalnya pengguna harus memasukkan data melalui keyboard. Agar program dapat menerima masukan data dan menampilkan hasilnya maka diperlukan perintah *input/output*.

3.2.1 Perintah Masukan

Merupakan operasi yang digunakan untuk **memasukkan data**, yang selanjutnya didefinisikan sebagai data variabel. Masukan tersebut dapat dilakukan secara interaktif menggunakan piranti masukan seperti keyboard. Perintah masukan yang bisa dipakai antara lain :

a. **`scanf()`**

Digunakan untuk memasukkan **berbagai jenis data**, terdapat dalam *file header* "**`stdio.h`**".

Sintaks :

`scanf("format_data", &variabel);`

Keterangan :

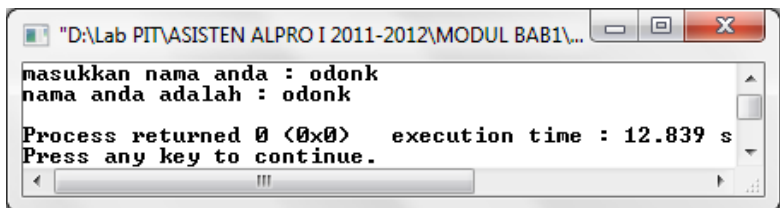
1. `format_data` berupa keterangan yang digunakan untuk menentukan jenis data apa yang dimiliki oleh variabel tersebut.
2. Simbol "&" merupakan pointer yang digunakan untuk menunjuk ke alamat variabel memori yang dituju.

Contoh penggunaan `scanf()` :

```
#include <conio.h>
#include <stdio.h>
#include <Windows.h>

int main () {
    char nama[30];
    printf ("masukkan nama Anda : ");
    scanf ("%s", nama);
    printf ("nama Anda adalah : %s \n", nama);
    system("pause");
    return 0;
}
```

Bila program tsb dijalankan, maka akan muncul hasil sbb :



b. `gets()`

Digunakan untuk memasukkan **data string**, terdapat pada *file*

header "**stdio.h**".

Sintaks:

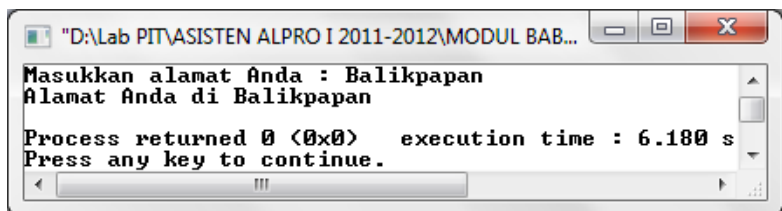
```
gets(<string_yang_dimasukkan>);
```

Contoh penggunaan `gets()` :

```
#include <stdio.h>
#include <Windows.h>

int main() {
    char alamat[50];
    printf("Masukkan alamat Anda : ");
    gets(alamat);
    printf("Alamat Anda di %s\n", alamat);
    system("pause");
    return 0;
}
```

Bila program tsb dijalankan, maka akan muncul hasil sbb :



c. **cin**

Merupakan sebuah objek di dalam C++ yang digunakan untuk memasukkan **data**, terdapat dalam *header file* "**iostream**".

Sintaks :

```
cin >> <var> ;
```

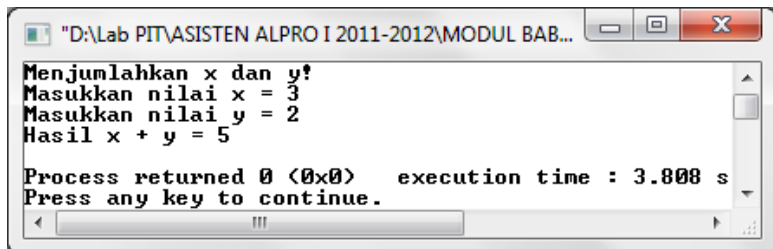
CATATAN! Untuk mendapatkan sebuah *input* data yang mengandung spasi, Anda bisa menggunakan :

```
cin.getline(<var>, sizeof(<var>));
```

Contoh penggunaan `cin` :

```
#include <iostream>
#include <Windows.h>
using namespace std;
int main() {
    int x,y,z;
    cout<<"Menjumlahkan x dan y!"<<endl;
    cout<<"Masukkan nilai x = "; cin>>x;
    cout<<"Masukkan nilai y = "; cin>>y;
    z=x+y;
    cout<<"Hasil x + y = "<<z<<endl;
    system("pause");
    return 0;
}
```

Bila program tsb dijalankan, maka akan muncul hasil sbb :

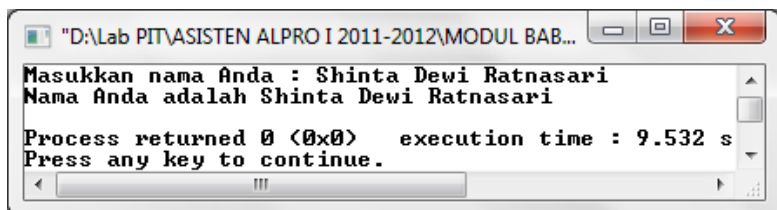


Contoh penggunaan `cin.getline` :

```
#include <iostream>
#include <Windows.h>
using namespace std;
```

```
int main() {
    char nama[50];
    cout<<"Masukkan nama Anda : ";
    cin.getline(nama,50);
    cout<<"Nama Anda adalah "<<nama<<endl;
    system("pause");
    return 0;
}
```

Bila program tsb dijalankan, maka akan muncul hasil sbb :



d. `getch()`

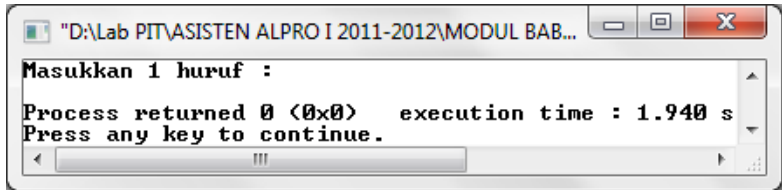
Digunakan untuk membaca **sebuah karakter** dengan sifat karakter yang dimasukkan **tidak perlu diakhiri dengan menekan tombol Enter**, dan karakter yang dimasukkan **tidak akan ditampilkan ke layar**, terdapat dalam *header file* "**conio.h**".

Contoh penggunaan `getch()` :

```
#include <iostream>
#include <conio.h>
#include <Windows.h>
using namespace std;
int main() {
    char huruf;
    cout<<"Masukkan 1 huruf : ";
    huruf = getch();
    cout<<endl;
```

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

Bila program tsb dijalankan, maka akan muncul hasil sbb :



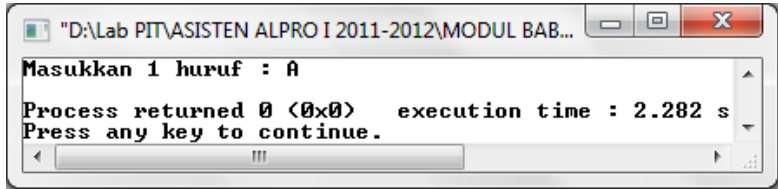
e. `getche()`

Digunakan untuk membaca **sebuah karakter** dengan sifat karakter yang dimasukkan **tidak perlu diakhiri dengan menekan tombol Enter**, dan karakter yang dimasukkan **akan ditampilkan ke layar**, terdapat dalam *header file* "`conio.h`".

Contoh penggunaan `getche()` :

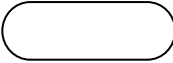


```
#include <iostream>  
#include <conio.h>  
#include <Windows.h>  
using namespace std;  
int main() {  
    char huruf;  
    cout<<"Masukkan 1 huruf : ";  
    huruf = getche();  
    cout<<endl;  
    system("pause");  
    return 0;  
}
```

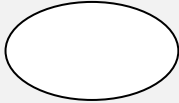


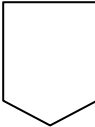


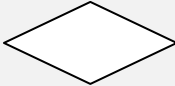
Bila program tsb dijalankan, maka akan muncul hasil sbb :





3.3 Flowchart

Flowchart atau bisa disebut diagram alir merupakan suatu penggambaran dari alur jalannya suatu program yang akan kita buat. *Flowchart* ini dibuat pada tahap perancangan program. Fungsi dari *flowchart* adalah untuk mengevaluasi jalannya suatu program sebelum direalisasikan dalam bentuk sintaks program sehingga kesalahan alur dalam sintaks program dapat diminimalisir. Simbol-simbol pada *flowchart* adalah sebagai berikut :

SIMBOL	NAMA	FUNGSI
	<i>Terminator</i>	Memulai dan mengakhiri suatu program
	<i>Process</i>	Proses perhitungan atau pengolahan data pada program
	<i>Input/Output</i>	Proses <i>input/output</i> data, parameter, informasi

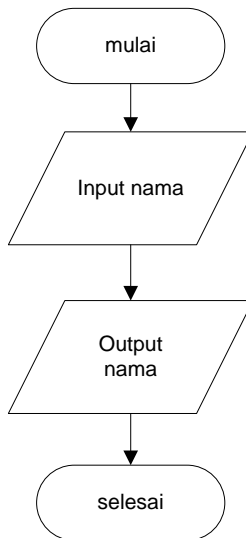
	Decision	Suatu kondisi yang akan menghasilkan dua kemungkinan pilihan (true & false)
	Preparation	Proses inisialiasasi atau pemberian harga awal
	On Page Connector	Penghubung bagian <i>flowchart</i> yang berada pada satu halaman
	Off Page Connector	Penghubung bagian <i>flowchart</i> yang berada pada halaman yang berbeda
	Flow Line	Arah aliran program
	Document	Menuliskan hasil proses eksekusi untuk dokumentasi (data berbentuk informasi)
	Predefined Process (Sub Program)	Untuk menyatakan sekumpulan langkah proses yang ditulis sebagai prosedur

	Display	Simbol untuk <i>output</i> yang ditujukan untuk suatu <i>device</i> , seperti printer atau plotter
	Storage	Untuk menyimpan data

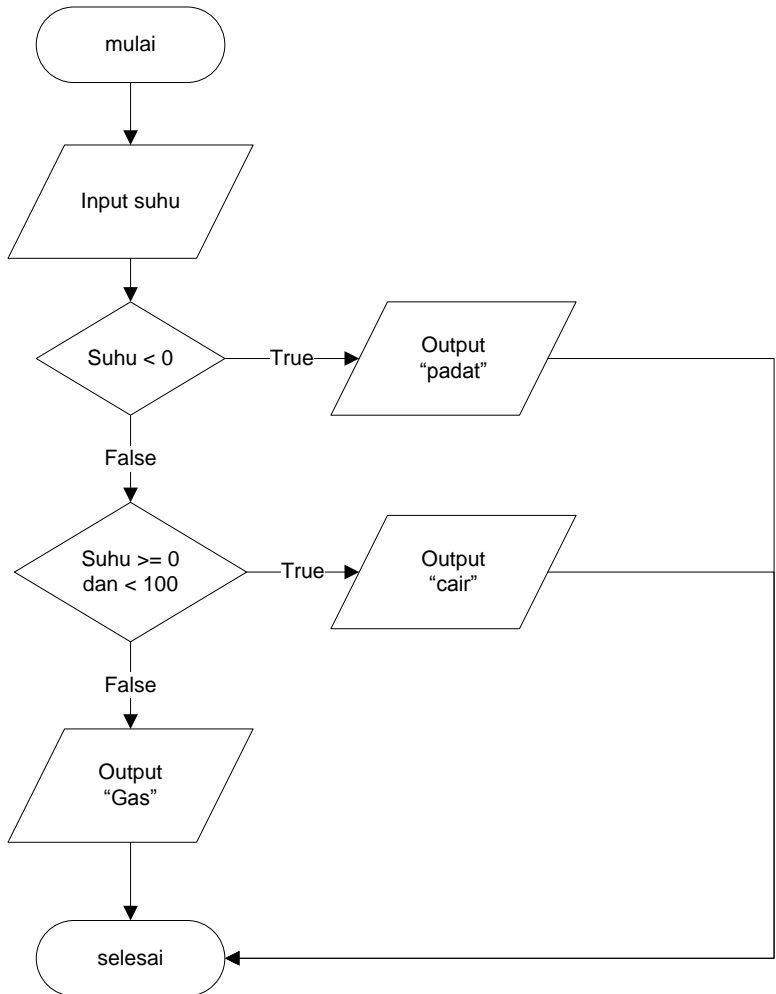
3.4 Langkah Praktikum

Berikut ini merupakan beberapa contoh pembuatan *flowchart* menggunakan simbol-simbol di atas :

1. Penggunaan **input** dan **output** pada *flowchart* :

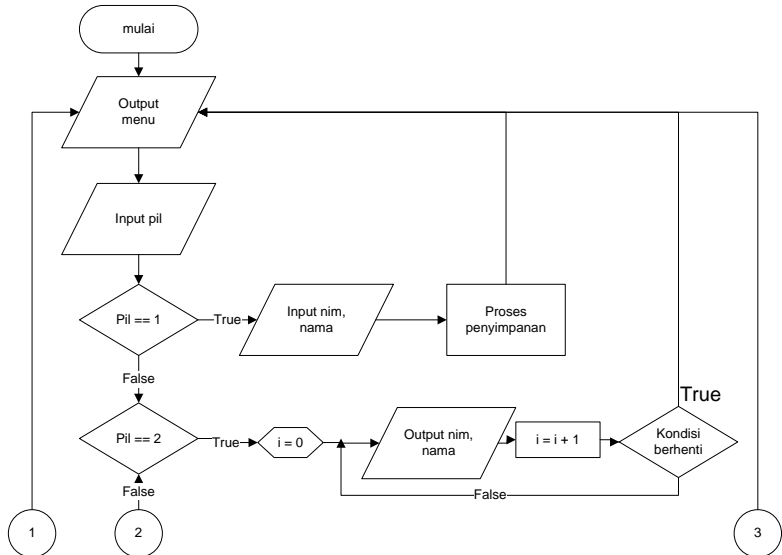


2. Penggunaan **percabangan** pada *flowchart* :



3. Penggunaan **konektor** pada *flowchart* :

Halaman 1



Halaman 2

