

MODUL PRAKTIKUM
“ALGORITMA DAN PEMROGRAMAN I”



Bahasa Pemrograman : C++
Software : Turbo C++ 4.5
Laboran : M. Fachrurrozi
Dwi Rosa Indah

LABORATORIUM DASAR KOMPUTER
PROGRAM ILMU KOMPUTER
UNIVERSITAS SRIWIJAYA
2006

I.PENDAHULUAN

1. 1. ALGORITMA

Algoritma adalah urutan aksi-aksi yang dinyatakan dengan jelas dan tidak rancu untuk memecahkan suatu masalah dalam rentang waktu tertentu. Setiap aksi harus dapat dikerjakan dan mempunyai efek tertentu.

Algoritma dapat dituliskan dengan banyak cara, mulai dari menggunakan bahasa alami yang digunakan sehari-hari, simbol grafik bagan alir, sampai menggunakan bahasa pemrograman seperti bahasa C atau C++.

1.2. C & C++

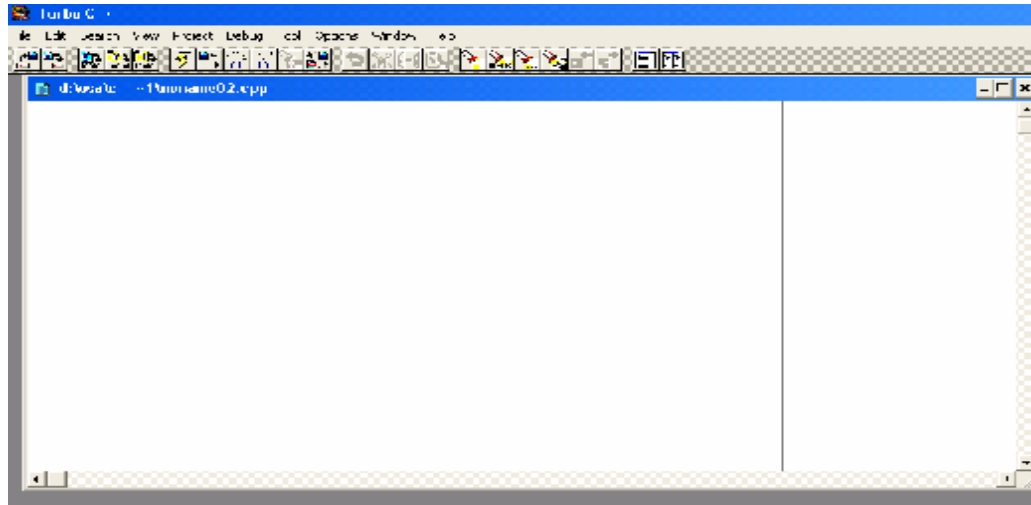
Berbicara tentang C++ biasanya tidak lepas dari C, sebagai bahasa pendahulunya. Pencipta C adalah Brian W. Kerninghan dan Dennis M. Ritchie pada sekitar tahun 1972, dan sekitar satu dekade setelahnya diciptakanlah C++, oleh Bjarne Stroustrup dari Laboratorium Bell, AT&T, pada tahun 1983. C++ cukup kompatibel dengan bahasa pendahulunya C. Pada mulanya C++ disebut “ a better C “. Nama C++ sendiri diberikan oleh Rick Mascitti pada tahun 1983, yang berasal dari operator increment pada bahasa C.

Keistimewaan yang sangat berarti dari C++ ini adalah karena bahasa ini mendukung pemrograman yang berorientasi objek (OOP / Object Oriented Programming).

1.3. LANGKAH-LANGKAH MENULISKAN PROGRAM DALAM TURBO C++

Langkah-langkahnya :

1. **Bukalah software Turbo C++, akan terlihat tampilan awal Turbo C++**
sebagai berikut :



2. Tulis source code program bahasa C++.

Source code C++ dapat ditulis pada text editor Turbo C++.

3. Kompilasi file dengan (ALT + C atau pilih submenu Compile)

Kompilasi file dijalankan Untuk mengubah source code menjadi sebuah program, kita gunakan compiler. Setelah source code tercompile, terbentuklah sebuah file objek dengan ekstension “.obj “. File “.obj “ ini belum merupakan sebuah program executable.

4. Jalankan Program dengan (CTRL+F9 atau pilih submenu Run)

Setelah kita kompilasi file yang berisi source code, maka sebagai hasil kompilasi tersebut kita akan mendapatkan suatu file yang bisa dijalankan (*executable file*). Menjalankan program yang kita buat berarti menjalankan file hasil proses kompilasi tersebut.

5. Untuk menyimpan pilih menu Save As

II. PENGENALAN C++

Setiap program C++ mempunyai bentuk umum seperti di bawah, yaitu:

```
# preprocessor directive
void main()
{
    // Batang Tubuh Program Utama
}
```

Penjelasan :

1. Include

Adalah salah satu pengarah *preprocessor directive* yang tersedia pada C++. Preprocessor selalu dijalankan terlebih dahulu pada saat proses kompilasi terjadi. Bentuk umumnya :

```
# include <nama_file>
```

tidak diakhiri dengan tanda semicolon, karena bentuk tersebut bukanlah suatu bentuk pernyataan, tetapi merupakan preprocessor directive. Baris tersebut menginstruksikan kepada kompiler yang menyisipkan file lain dalam hal ini file yang berakhiran .h(file header) yaitu file yang berisi sebagai deklarasi contohnya:

- # include <iostream.h> : diperlukan pada program yang melibatkan objek cout
- # include <conio.h> : diperlukan bila melibatkan clrscr(), yaitu perintah untuk membersihkan layar.
- # include <iomanip.h> : diperlukan bila melibatkan setw() yang bermanfaat untuk mengatur lebar dari suatu tampilan data.
- # include <math.h> : diperlukan pada program yang menggunakan operasi sqrt () yang bermanfaat untuk operasi matematika kuadrat.

2. Fungsi main ()

Fungsi ini menjadi awal dan akhir eksekusi program C++. **main** adalah nama judul fungsi. Melihat bentuk seperti itu dapat kita ambil kesimpulan bahwa batang tubuh program utama berada didalam fungsi main(). Berarti dalam setiap pembuatan program utama, maka dapat dipastikan seorang pemrogram menggunakan minimal sebuah fungsi. Pembahasan lebih lanjut mengenai fungsi akan diterangkan kemudian. Yang sekarang coba ditekankan adalah kita menuliskan program utama kita didalam sebuah fungsi main().

3. Komentar

Komentar tidak pernah dicompile oleh compiler. Dalam C++ terdapat 2 jenis komentar, yaitu:

Jenis 1 : /* Komentar anda diletakkan di dalam ini

Bisa menggapit lebih dari satu baris */

Jenis 2 : // Komentar anda diletakkan disini (hanya bisa perbaris)

4. Tanda Semicolon

Tanda semicolon “ ; ” digunakan untuk mengakhiri sebuah pernyataan. Setiap pernyataan harus diakhiri dengan sebuah tanda semicolon.

5. Mengenal cout(dibaca : C out)

Pernyataan cout merupakan sebuah objek di dalam C++, yang digunakan untuk mengarahkan data ke dalam standar output (cetak pada layar)

Contoh :

```
# include <iostream.h>

void main ()
{
    cout << " HAI, selamat menggunakan C++ ";
}
```

Tanda “ << “ merupakan sebuah operator yang disebut operator “penyisipan/peletakan”

III. VARIABEL, TIPE DATA

3.1 VARIABEL

Variabel adalah suatu pengenal (identifier) yang digunakan untuk mewakili suatu nilai tertentu di dalam proses program. Berbeda dengan konstanta yang nilainya selalu tetap, nilai dari suatu variable bisa diubah-ubah sesuai kebutuhan. Untuk memperoleh nilai dari suatu variable digunakan pernyataan penugasan (*assignment statement*), yang mempunyai sintaks sebagai berikut :

```
variable = ekspresi ;
```

Nama dari suatu variable dapat ditentukan sendiri oleh pemrogram dengan aturan sebagai berikut :

1. Terdiri dari gabungan huruf dan angka dengan karakter pertama harus berupa huruf. Bahasa C ++ bersifat *case-sensitive* artinya huruf besar dan kecil dianggap berbeda. Jadi antara **nim**, **NIM** dan **Nim** dianggap berbeda.
2. Tidak boleh mengandung spasi.
3. Tidak boleh mengandung symbol-simbol khusus, kecuali garis bawah (underscore). Yang termasuk symbol khusus yang tidak diperbolehkan antara lain : \$, ?, %, #, !, &, *, (,), -, +, =dsb.
4. Panjangnya bebas, tetapi hanya 32 karakter pertama yang terpakai.

Contoh penamaan variabel yang benar :

NIM, a, x, nama_mhs, f3098, f4, nilai, budi, dsb.

Contoh penamaan variable yang salah :

%nilai_mahasiswa, 80mahasiswa, rata-rata, ada spasi, penting!, dsb

3.2 DEKLARASI

Deklarasi diperlukan bila kita akan menggunakan pengenal (identifier) dalam program. Identifier dapat berupa variable, konstanta dan fungsi.

3.2.1 DEKLARASI VARIABEL

Bentuk umumnya :

```
Nama_tipe nama_variabel ;
```

Contoh :

```
int x; // Deklarasi x bertipe integer
char y, huruf, nim[10]; // Deklarasi variable bertipe char
float nilai; // Deklarasi variable bertipe float
double beta; // Deklarasi variable bertipe double
int array[5][4]; // Deklarasi array bertipe integer
```

Contoh :

```
#include <iostream.h>
void main()
{
int n;
n=66; // sama juga jika ditulis int n=66;
cout<<n<<endl; // n sebagai variabel
cout<<'n'<<endl; // end sebagai karakter
}
```

Outputnya :

```
66
n
```

3.2.1 DEKLARASI KONSTANTA

a. Menggunakan keyword const

Contoh : const float PI = 3.14152965;

Berbeda dengan variable, konstanta bernama tidak dapat diubah jika telah diinisialisasi

b. Menggunakan #define

Contoh : #define PI 3.14152965

Keuntungan menggunakan #define apabila dibandingkan dengan **const** adalah kecepatan kompilasi, karena sebelum kompilasi dilaksanakan, kompiler pertama kali mencari symbol #define (oleh sebab itu mengapa # dikatakan preprocessor directive) dan mengganti semua Phi dengan nilai 3.14152965.

Contoh :

```
# include <iostream.h>

void main ()
{
    const float phi = 3.14;
    float jari_jari, luas, keliling;

    jari_jari=7.0;

    luas = 0.5 * phi * jari_jari * jari_jari;
    keliling = 2 * phi * jari_jari;

    cout << " Luas lingkaran    = " << luas << endl;
    cout << " Keliling Lingkaran = " << keliling;
}
```

3.3 TIPE DATA

Tipe data dapat dikelompokkan menjadi atas dua macam :

1. Tipe Dasar.
2. Tipe Bentukkan.

3.3.1 TIPE DASAR

Adalah tipe yang dapat langsung dipakai.

Tipe Dasar	Ukuran Memori (byte)	Jangkauan Nilai	Jumlah Digit Presisi
Char	1	-128 hingga +127	-
Int	2	-32768 hingga +32767	-
Long	4	-2.147.438.648 hingga 2.147.438.647	-
Float	4	3,4E-38 hingga 3,4E38	6-7
Double	8	1.7E-308 hingga 1.7E308	15-16
long double	10	3.4E-4932 hingga 1.1E4932	19

NB : Untuk mengetahui ukuran memori dari suatu tipe digunakan fungsi sizeof(tipe)

Tipe data dapat diubah (type cast), misalkan:

```
float x = 3.345;
```

```
int p = int(x);
```

maka nilai p adalah 3 (terjadi truncating).

Tipe data yang berhubungan dengan bilangan bulat adalah char, int, long. Sedangkan lainnya berhubungan dengan bilangan pecahan.

Contoh :

```
#include <iostream.h>
void main()
{
    int n;
    cout<<n<<endl;    // n sebagai variabel
}
}
```

Outputnya :

18125

Darimana angka 18125 diperoleh ?

Jika variable tidak diinisialisai, namun nilai keluarannya diminta, maka compiler dengan bijak akan menampilkan nilai acak yang nilainya tergantung dari jenis compilernya.

3.3.1.1 KARAKTER & STRING LITERAL

String adalah gabungan dari karakter

Contoh : “ Belajar “ à Literal String

“ B “ à Karakter

Panjang String

strlen() à nama fungsi untuk menghitung panjang string

Fungsi strlen() dideklarasikan dalam file string.h. Jadi bila anda ingin menggunakan fungsi strlen(), maka preprocessor directive #include<string.h> harus dimasukkan dalam program diatas main().

Contoh :

```
#include <iostream.h>
#include <string.h>
void main()
{
    cout<<strlen("Selamat Pagi.\n")<<endl;
    cout<<strlen("Selamat Pagi.")<<endl;
    cout<<strlen("Selamat")<<endl;
    cout<<strlen("S")<<endl;
    cout<<strlen("");
}
}
```

Outputnya:

```
14
13
7
1
0
```

Perhatikan, bahwa disetiap akhir baris pernyataan diakhiri dengan tanda titik – koma (semicolon) “;”.

Perhatikan, bahwa :

- ‘\n ‘ dihitung satu karakter. \n disebut newline karakter
- Endl juga merupakan newline karakter (sama kegunaannya seperti \n).

Dalam C++, selain \n terdapat juga beberapa karakter khusus yang biasa disebut *escape sequence characters*, yaitu

Karakter	Keterangan
\0	Karakter ber-ASCII nol (karakter null)
\a	Karakter bell
\b	Karakter backspace
\f	Karakter ganti halaman (formfeed)
\n	Karakter baris baru (newline)
\r	Karakter carriage return (ke awal baris)
\t	Karakter tab horizontal
\v	Karakter tab vertika
\\	Karakter \
\'	Karakter ‘
\”	Karakter “
\?	Karakter ?
\ooo	Karakter yang nilai oktalnya adalah ooo (3 digit octal)
\xhh	Karakter yang nilai heksadesimalnya adalah hh (2 digit heksadesimal)

3.3.1.2 KEYWORD & IDENTIFIER

Dalam bahasa pemrograman, suatu program dibuat dari elemen-elemen sintaks individual yang disebut token, yang memuat nama variable, konstanta, keyword, operator dan tanda baca.

Contoh :

```
# include <iostream.h>
void main()
{
    int n=66;
    cout<<n<<endl;    // n sebagai variabel
}
```

Output :

66

Program diatas memperlihatkan 15 token, yaitu
main, (,), {, int, n, =, 66, :, cout, <<, endl, return, 0 dan }

Token n adalah suatu variable

Token 66,0 adalah suatu konstanta

Token int, return dan endl adalah suatu keyword

Token = dan << adalah operator

Token(,), {, :, dan } adalah tanda baca

Baris pertama berisi suatu preprocessor directive yang bukan bagian sebenarnya dari program

3.3.2 TIPE BENTUKAN

Merupakan tipe yang dibentuk dari tipe dasar. Seperti Tipe Struktur.

3.3.2.1 TIPE STRUKTUR

Suatu tipe data yang merupakan kumpulan dari tipe data lainnya. Struktur terdiri dari data yang disebut field. Field – field tersebut digabungkan menjadi satu tujuan untuk kemudahan dalam operasi.

Bentuk umumnya :

```
typedef struct{ tipe nama_field1;
                tipe nama_field2;
                tipe nama_field3;
                ....
                }nama_variabel;
```

Contoh :

```
# include <iostream.h>

typedef struct { int tahun;
                int bulan;
                int tanggal;
            } data_tunggal;

data_tunggal tanggal_lahir;
void main ()
{
    tanggal_lahir.tanggal=13;
    tanggal_lahir.bulan=1;
    tanggal_lahir.tahun=1982;

    cout << tanggal_lahir.tanggal << '/'
         << tanggal_lahir.bulan << '/'
         << tanggal_lahir.tahun << endl;
}
```

Perhatikan bahwa pada akhir dari **typedef struct** diberi tanda semicolon.

Latihan :

1. Buatlah program dengan menggunakan define untuk menghitung volume Tabung (Rumus Volume Tabung : $\text{phi} \times \text{jari-jari} \times \text{jari-jari} \times \text{tinggi}$) dan Luas Tabung (Rumus Luas tabung : $2 \times \text{phi} \times \text{jari-jari} \times \text{tinggi}$) dimana jari-jari 7 dan tinggi 24.
2. Buatlah program untuk mencatat data mahasiswa yang terdiri dari field nama, nim dan nilai.

IV. OPERATOR DAN STATEMEN I/O

4.1 OPERATOR

Operator adalah symbol yang biasa dilibatkan dalam program untuk melakukan sesuatu operasi atau manipulasi.

4.1.1 OPERATOR PENUGASAN

Operator Penugasan (*Assignment operator*) dalam bahasa C++ berupa tanda sama dengan (“=”).

Contoh :

```
nilai = 80;
```

```
A = x * y;
```

Penjelasan :

variable “nilai” diisi dengan 80 dan
variable “A” diisi dengan hasil perkalian antara x dan y.

4.1.2 OPERATOR ARITMATIKA

Operator	Deskripsi	Contoh
+	Penjumlahan (Add)	$m + n$
-	Pengurangan (Subtract)	$m - n$
*	Perkalian (Multiply)	$m * n$
/	Pembagian (Divide)	m / n
%	Sisa Pembagian Integer (Modulus)	$m \% n$
-	Negasi (Negate)	-m

NB : Operator seperti operator negasi (-) disebut unary operator, karena membutuhkan hanya satu buah operand

Operator % (modulus) digunakan untuk mencari sisa pembagian antara dua bilangan. Misalnya : $9 \% 2 = 1$, $9 \% 3 = 0$

Contoh :

```
#include <iostream.h>
void main()
{
    int m = 82, n = 26;
    cout<<m<<" + "<<n<<" = "<<m+n<<endl;
    cout<<m<<" - "<<n<<" = "<<m-n<<endl;
    cout<<m<<" * "<<n<<" = "<<m*n<<endl;
    cout<<m<<" / "<<n<<" = "<<m/n<<endl;
    cout<<m<<" % "<<n<<" = "<<m%n<<endl;
    cout<<"- "<<m<<" = "<<-m<<endl;
}
```

Output :

```
82 + 26 = 108
82 - 26 = 56
82 * 26 = 2132
82 / 26 = 3
82 % 26 = 4
-82 = -82
```

Karena tipe datanya adalah int, maka $82/26=3$, supaya dapat merepresentasikan nilai yang sebenarnya, gunakan tipe data float.

Cara lain penulisan dengan menggunakan operator aritmatika :

```
m = m + n   ○   m += n
m = m - n   ○   m -= n
m = m * n   ○   m *= n
m = m / n   ○   m /= n
m = m % n   ○   m %= n
```

4.1.3 OPERATOR HUBUNGAN (PERBANDINGAN)

Operator Hubungan digunakan untuk membandingkan hubungan antara dua buah operand (sebuah nilai atau variable). Operator hubungan dalam bahasa C++

Operator	Arti	Contoh	
==	Sama dengan (bukan assignment)	$x = y$	Apakah x sama dengan y
!=	Tidak sama dengan	$x != y$	Apakah x tidak sama dengan y
>	Lebih besar	$x > y$	Apakah x lebih besar dari y
<	Lebih kecil	$x < y$	Apakah x lebih kecil dari y
>=	Lebih besar atau sama dengan	$x >= y$	Apakah x lebih dari sama dengan y
<=	Lebih kecil atau sama dengan	$x <= y$	Apakah x kurang dari sama dengan y

Contoh:

```
#include <iostream.h>

void main()
{
    int m = 5, n =7;
    if (m == n) cout<<m<<" sama dengan "<<n<<endl;
    else if (m != n) cout<<m<<" tidak sama dengan "<<n<<endl;
    else if (m > n) cout<<m<<" lebih besar dari "<<n<<endl;
    else if (m < n) cout<<m<<" lebih kecil dari "<<n<<endl;
}
```

Outputnya :

```
5 tidak sama dengan 7
```

4.1.4 OPERATOR NAIK DAN TURUN (INCREMENT DAN DECREMENT)

Operator increment à ++

Operator decrement à --

Contoh :

```
#include <iostream.h>
void main()
{
    int m = 44, n = 66;

    cout<<"m = "<<m<<", n = "<<n<<endl;
    ++m; --n;
    cout<<"m = "<<m<<", n = "<<n<<endl;
    m++; n--;
    cout<<"m = "<<m<<", n = "<<n<<endl;
}
```

Outputnya :

```
m = 44, n = 66
m = 45, n = 65
m = 46, n = 64
```

Terlihat bahwa operator pre-increment dan post-increment memiliki akibat yang sama, yaitu menambah nilai satu pada m dan memasukkan nilai tersebut kembali ke m ($m=m+1$). Hal yang sama juga terjadi pada operator pre-decrement dan post-decrement yang memberikan akibat yang sama, yaitu mengurangi nilai satu dari n ($n = n - 1$).

Tetapi bila digunakan sebagai sub-ekspresi, operator post-increment dan pre-increment menunjukkan hasil yang berbeda

Contoh :

```
# include <iostream.h>
void main()
{
    int m = 66, n ;
    n = ++m;
    cout<<"m = "<<m<<" , n = "<<n<<endl;
    n = m++;
    cout<<"m = "<<m<<" , n = "<<n<<endl;
    cout<<"m = "<<m++<<endl;
    cout<<"m = "<<m<<endl;
    cout<<"m = "<<++m<<endl;
}
```

Outputnya :

```
m = 67, n = 67
m = 68, n = 67
m = 68
m = 69
m = 70
```

Penjelasan :

Dalam penugasan yang pertama, m adalah pre-increment, menaikkan nilainya menjadi 67, yang selanjutnya dimasukkan ke n.

Dalam penugasan kedua, m adalah post-increment, sehingga 67 dimasukkan dahulu ke n baru kemudian nilai m-nya dinaikkan, itu sebabnya mengapa nilai m = 68 dan n = 67.

Dalam penugasan ketiga, m adalah post-increment, sehingga nilai m (= 68) ditampilkan dahulu (ke layar) baru kemudian nilai m dinaikkan menjadi 69.

Dalam penugasan keempat, m adalah pre-increment, sehingga nilai m dinaikkan dahulu menjadi 70 baru kemudian ditampilkan ke layar.

Supaya lebih paham, perhatikan pula contoh dibawah.

Contoh :

```
# include <iostream.h>

void main ()
{
    int m=5, n;
    n=++m * --m;
    cout << "m=" << m << " n=" << n << endl;
    cout << ++m << " " << ++m << " " << ++m << endl;
}
```


Penjelasan :

Dalam penugasan untuk n, pertama kali m dinaikkan (++m) menjadi 6, kemudian m diturunkan kembali menjadi 5, karena adanya --m. Sehingga nilai m sekarang adalah 5 dan nilai m = 5 inilah yang dievaluasi pada saat penugasan perkalian dilakukan. Pada baris terakhir, ketiga sub-ekspresi dievaluasi dari kanan ke kiri.

4.1.5 OPERATOR BITWISE

Operator	Deskripsi	Contoh
<<	Geser n bit ke kiri (left shift)	m << n
>>	Geser n bit ke kanan (right shift)	m >> n
&	Bitwise AND	m & n
	Bitwise OR	m n
^	Bitwise XOR	m ^ n
~	Bitwise NOT	~m

NB : Seluruh operator bitwise hanya bisa dikenakan pada operand bertipe data int atau char

Berikut ini diberikan tabel kebenaran untuk operator logika

P = A operator B

AND

A	B	P
0	0	0
0	1	0
1	0	0
1	1	1

OR

A	B	P
0	0	0
0	1	1
1	0	1
1	1	1

XOR

A	B	P
0	0	0
0	1	1
1	0	1
1	1	0

Contoh :

```
# include <iostream.h>
void main()
{
    int m = 82, n = 26;
    cout<<m<<" << 2"<<" = "<<(m<<2)<<endl;
    cout<<m<<" >> 2"<<" = "<<(m>>2)<<endl;
    cout<<m<<" & "<<n<<" = "<<(m&n)<<endl;
    cout<<m<<" | "<<n<<" = "<<(m|n)<<endl;
    cout<<m<<" ^ "<<n<<" = "<<(m^n)<<endl;
    cout<<"~"<<m<<" = "<<~m<<endl;
}
```

Output :

```
82 << 2 = 328
82 >> 2 = 20
82 & 26 = 18
82 | 26 = 90
82 ^ 26 = 72
~82 = -83
```

Penjelasan :

Nilai keluaran diatas, tergantung dari jenis compiler yang digunakan. Hasil diatas merupakan keluaran dari compiler Turbo C++.

Pada Turbo C++ besar dari integer adalah 2 byte atau sama dengan 16 bit, untuk mengetahuinya digunakan perintah

```
cout<<sizeof(int)<<endl; // Untuk mengetahui besar dari int
```

Maka :

$82_{10} = 0000000001010010_2$ dan

$26_{10} = 0000000000011010_2$

Sehingga :

$82 \ll 2 \rightarrow 0000000101001000_2 = 328_{10}$

$82 \gg 2 \rightarrow 000000000010100_2 = 20_{10}$

$82 \& 26 \rightarrow 0000000001010010_2$

0000000000011010_2

----- &

$000000000010010_2 = 18_{10}$

dan begitu juga untuk operasi OR dan XOR.

$\sim 82 \rightarrow$ digunakan 2's complement, yaitu

$82_{10} = 0000000001010010_2$ lalu dinegasikan tiap bitnya menjadi
 1111111110101101_2 kemudian LSB ditambah 1 menjadi
 $1111111110101110 = 65454_{10}$ nilai ini melebihi jangkauan maksimum int
 yang berkisar di -32768 sampai 32767, sehingga nilai yang keluar yaitu
 83.

Cara lain penulisan dengan menggunakan operator bitwise :

$m = m \ll n$ **⓪** $m \ll= n$

$m = m \gg n$ **⓪** $m \gg= n$

$m = m \& n$ **⓪** $m \&= n$

$m = m | n$ **⓪** $m |= n$

$m = m \wedge n$ **⓪** $m \wedge= n$

3.1.6 OPERATOR LOGIKA

Operator logika digunakan untuk menghubungkan dua atau lebih ungkapan menjadi sebuah ungkapan berkondisi.

Operator	Deskripsi	Contoh
&&	logic AND	$m \&\& n$
	logic OR	$m n$
!	logic NOT	$!m$

Contoh :

```

#include <iostream.h>
void main()
{
  int m = 166;
  cout<<"(m)>=0 && m<=150) -> "<<(m)>=0 && m<=150)<<endl;
  cout<<"(m)>=0 || m<=150) -> "<<(m)>=0 || m<=150)<<endl;
}
  
```

Outputnya :

```

(m)>=0 && m<=150) -> 0
(m)>=0 || m<=150) -> 1
  
```

Penjelasan :

Hasil keluaran dari operator logika adalah 0 dan 1.

0 jika keluarannya salah dan 1 jika keluarannya benar.

3.17 OPERATOR KONDISI

Operator kondisi digunakan untuk memperoleh nilai dari dua kemungkinan

ungkapan1 ? ungkapan2 : ungkapan3

Bila nilai *ungkapan1* benar, maka nilainya sama dengan *ungkapan2*, bila tidak maka nilainya sama dengan *ungkapan3*

Contoh :

```
#include <iostream.h>
void main()
{
    int m = 26, n = 82;
    int min = m < n ? m : n;
    cout<<"Bilangan terkecil adalah "<<min<<endl;
}
```

Outputnya :

Bilangan terkecil adalah 26

3.2 STATEMEN I/O

Pada C++ terdapat 2 jenis I/O dasar, yaitu:

- Statemen Input adalah Statemen / fungsi yang digunakan untuk membaca data dari inputing device (keyboard/mouse), contoh : cin (character in)
- Statemen Output adalah Statemen yang digunakan untuk menuliskan data ke layar monitor, contoh : cout (character out)

Contoh 1:

```
# include <iostream.h>
void main()
{
    char nama[100];    // Dekalarasi variable nama
    cout<<"Masukkan nama Anda : ";
    cin>>nama;    // Meminta user untuk menginisialisasi variable nama
    cout<<"Nama anda adalah "<<nama;
}
```

Contoh 2:

```
#include <iostream.h>
#include <conio.h>
void main()
{
    int nilai;
    clrscr();
    cout << "Masukkan nilai Anda : " ;
    cin >> nilai;
    cout << "Anda memperoleh nilai " << nilai << endl;
    cout << "Apakah Anda telah puas mendapat nilai ? "<< nilai;
    getch();
}
```

Outputnya :

```
Masukkan nilai Anda : 45
Anda memperoleh nilai 45
Apakah Anda telah puas mendapat nilai ? 45
```

Contoh 3:

```
// Program untuk mempertukarkan nilai A dengan nilai B
# include <iostream.h>

void main ()
{
    int A, B, temp;
    cout << "A= ";
    cin >> A;

    cout << "B= ";
    cin >> B;

    temp =A;
    A=B;
    B=temp ;

    cout << "Jadi sekarang : " << endl;
    cout << "A= " << A << endl;
    cout << "B= " << B;
}
```

Output :

```
A= 5
B= 3
Jadi sekarang :
A= 3
B= 5
```

Latihan

1. Buatlah program untuk menghitung luas bangunan geometri (bujursangkar, lingkaran, segitiga dan trapesium). Data masukkan dibaca dari piranti masukkan dan luas bangun ditampilkan sebagai keluaran.
2. Buatlah program untuk menghitung harga total suatu barang, dimana jumlah barangnya 5, harga perunit 5203.02.
3. Buatlah program untuk penggunaan operasi aritmatika yaitu penjumlahan, pembagian, perkalian, dan pengurangan dengan variabel yang diinputkan.
4. Mencetak sejumlah deret bilangan ganjil antara 1 sampai N, dimana N dimasukkan oleh user.

IV. PEMILIHAN

Suatu Struktur dasar algoritma yang memiliki satu atau lebih kondisi tertentu dimana sebuah instruksi dilaksanakan jika sebuah kondisi/persyaratan terpenuhi. Ada beberapa bentuk struktur dasar pemilihan ini :

4.1 PERNYATAAN *if*

Sebuah pernyataan yang dapat dipakai untuk mengambil keputusan berdasarkan suatu kondisi. Bentuk pernyataan ini ada dua macam :

- **if** saja dan
- **else**

Bentuk Umumnya Satu Kasus:

<pre>if (kondisi) pernyataan ;</pre>
--

Pernyataan dilaksanakan jika dan hanya jika kondisi yang diinginkan terpenuhi, jika tidak program tidak memberikan hasil apa-apa.

Contoh :

```
// Contoh Penggunaan if
#include <iostream.h>

void main ()
{
    int usia;

    cout << "Berapa usia Anda : ";
    cin >> usia;

    if (usia < 17)
        cout << "Anda tidak diperkenankan menonton" << endl;
}
```

Outputnya :

```
Berapa usia Anda : 12
Anda tidak diperkenankan menonton
```

Penjelasan :

Tampak bahwa kalau usia yang dimasukkan lebih dari atau sama dengan 17, program tidak menghasilkan apa-apa.

Bentuk Umumnya Dua Kasus :

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

Pernyataan1 dilaksanakan jika dan hanya jika kondisi yang diinginkan terpenuhi, jika tidak, lakukan pernyataan2.

Jika Anda tidak mempergunakan pernyataan *else* program tidak akan error, namun jika anda mempergunakan pernyataan *else* tanpa didahului pernyataan *if*, maka program akan error.

Jika pernyataan1 atau pernyataan2 hanya terdiri dari satu baris, maka tanda { } tidak diperlukan, namun jika lebih maka diperlukan.

Bentuk Umumnya Banyak Kasus :

```
if (kondisi)
{
    pernyataan1;
    pernyataan1a;
    pernyataan1b;
}
else
{
    pernyataan2;
    pernyataan2a;
    pernyataan2b;
}
```


Contoh 1:

```
void main()
{
    int m = 166;
    if(m == 0)
    {
        cout<<"Nilainya sama dengan nol\n";
    }
    else
    {
        cout<<"Nilainya tidak sama dengan nol\n";
        cout<<"Nilainya sama dengan "<<m<<endl;
    }
}
```

Contoh 2:

```
# include <iostream.h>

void main ()
{
    int usia;

    cout << "Berapa usia Anda ? ";
    cin >> usia;

    if (usia < 17)
        cout << "Anda tidak diperkenankan menonton" << endl;
    else
        cout << "Selamat Menonton" << endl;
}
```

Outputnya :

```
Berapa usia Anda ? 16
Anda tidak diperkenankan menonton
```

Penjelasan :

Terlihat bahwa kalau usia yang dimasukkan lebih dari 17, program akan memberi pesan Selamat Menonton.

Selain dari **if ... else**, juga dikenal bentuk **if ... else if**. Adapun perbedaannya diilustrasikan oleh dua contoh dibawah ini.

```

# include <iostream.h>

void main ()
{
    int m =166;
    if (m > 1000)
        cout << m << " Lebih Besar dari 1000 " << endl;
    if (m > 100)
        cout << m << " Lebih Besar dari 100 " << endl;
    if (m >10)
        cout << m << " lebih besar dari 10 " << endl;
}

```

Contoh 3 :

```

# include <iostream.h>

void main ()
{
    int m = 166;
    if (m > 1000)
        cout << m << " lebih besar dari 1000 " << endl;
    else if (m > 100)
        cout << m << " Lebih besar dari 100 " << endl;
    else if (m >10)
        cout << m << " Lebih besar dari 10 " << endl ;
}

```

Outputnya:

```

166 Lebih Besar dari 100
166 lebih besar dari 10

```

Mengapa ? Karena contoh 2 sama saja jika ditulis seperti dibawah ini

Contoh 4 :

```

#include <iostream.h>
void main()
{
    int m = 166;
    if(m > 1000)
        cout<<m<<" lebih besar dari 1000\n";
    else
    {
        if(m > 100)
            cout<<m<<" lebih besar dari 100\n";
        else if(m > 10)
            cout<<m<<" lebih besar dari 10\n";
    }
}

```

Contoh diatas disebut juga *nested conditional*

4.2 PERNYATAAN *Switch*

Pernyataan **switch** adalah pernyataan yang digunakan untuk menjalankan salah satu pernyataan dari beberapa kemungkinan pernyataan, berdasarkan nilai dari sebuah ungkapan dan nilai penyeleksian.

Pernyataan **if...else if** jamak dapat dibangun dengan pernyataan **switch**.

Bentuk Umumnya :

```
switch (ekspresi)
{
  case konstanta1 :
    pernyataan1 ;
    break ;
  case konstanta2 :
    pernyataan2 ;
    break ;
  case konstanta3 :
    pernyataan3 ;
    break ;
  :
  :
  case konstantaN :
    pernyataanN ;
    break ;
  default :
    pernyataanlain;
}
```

Hal – hal yang perlu diperhatikan adalah :

1. Dibelakang keyword **case** harus diikuti oleh sebuah konstanta, tidak boleh diikuti oleh ekspresi ataupun variable.
2. Konstanta yang digunakan bertipe **int** atau **char**
3. Jika bentuknya seperti diatas maka apabila *ekspresi* sesuai dengan konstanta2 maka pernyataan2, pernyataan3 sampai dengan pernyataanlain dieksekusi. Untuk mencegah hal tersebut, gunakan keyword **break**; Jika keyword **break** digunakan maka setelah pernyataan2 dieksekusi program langsung keluar dari pernyataan **switch**. Selain digunakan dalam **switch**, keyword *break* banyak digunakan untuk keluar dari pernyataan yang berulang (*looping*).

4. pernyataan lain dieksekusi jika konstanta1 sampai konstantaN tidak ada yang memenuhi *ekspresi*.

Contoh :

```
// Program untuk melihat nilai akhir test
// Nilai A jika nilai diatas 80, B jika 70 <= nilai < 80
// C jika 50 <= nilai < 70 , D jika 30 <= nilai <50
// E jika nilai < 30

# include <iostream.h>
void main ()
{
    int nilai;
    cout << "Masukkan nilai test : ";
    cin >> nilai;
    switch (nilai/10)
    {
        case 10:
        case 9:
        case 8:
            cout << 'A' <<endl; break;
        case 7:
            cout << 'B' <<endl; break;
        case 6:
        case 5:
            cout << 'C' <<endl; break;
        case 4:
        case 3:
            cout << 'D' <<endl; break;
        case 2:
        case 1:
        case 0:
            cout << 'E' <<endl; break;
        default:
            cout << " Salah, nilai diluar jangkauan. " << endl;
    }
}
```

Output :

```
Masukkan nilai test : 45
D
Masukkan nilai test : 450
Salah, nilai diluar jangkauan.
```

Ket : 45, dan 450 adalah input dari user

Latihan :

1. Buatlah program untuk mencari apakah bilangan tersebut ganjil atau genap, dimana bilangan merupakan piranti masukan
2. Buatlah program untuk menseleksi suatu bilangan dengan ketentuan sebagai berikut :
 $0 \leq \text{nilai} < 30$: Nilai rendah
 $30 \leq \text{nilai} < 60$: Nilai sedang
 $60 \leq \text{nilai} \leq 100$: Nilai tinggi
3. Buatlah program dalam bentuk menu yang mampu menghitung :
 - a. Luas dan Keliling Bujur sangkar
 - b. Luas dan Keliling persegi panjang
 - c. Luas dan keliling lingkaran

V. PENGULANGAN

Sebuah / kelompok instruksi diulang untuk jumlah pengulangan tertentu. Baik yang terdefiniskan sebelumnya ataupun tidak.

Struktur pengulangan terdiri atas dua bagian :

1. Kondisi pengulangan yaitu ekspresi boolean yang harus dipenuhi untuk melaksanakan pengulangan
2. Isi atau badan pengulangan yaitu satu atau lebih pernyataan (aksi) yang akan diulang.

Perintah atau notasi dalam struktur pengulangan adalah :

1. Pernyataan **while**
2. Pernyataan **do..while**
3. Pernyataan **for**
4. Pernyataan **continue dan break**
5. Pernyataan **go to**

5.1 PERNYATAAN *while*

Pernyataan **while** merupakan salah satu pernyataan yang berguna untuk memproses suatu pernyataan atau beberapa pernyataan beberapa kali. Pernyataan **while** memungkinkan statemen-stemen yang ada didalamnya tidak diakukan sama sekali.

Bentuk Umumnya :

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

Contoh:

```
# include <iostream.h>

void main ()
{
    int i;

    i=0;
    while (i<10)
    {
        cout << "C++" << endl;
        i++;
    }
}
```

Output :

```
C++
C++
C++
C++
C++
C++
C++
C++
C++
C++
```

Penjelasan :

Program diatas digunakan untuk mengulangan tulisan sebanyak 10 kali

5.2 PERNYATAAN *do...while*

Pernyataan **do...while** mirip seperti pernyataan **while**, hanya saja pada **do...while** pernyataan yang terdapat didalamnya minimal akan sekali dieksekusi.

Bentuk Umumnya :

<pre>do { <i>pernyataan</i> ; } while(kondisi);</pre>

Contoh :

```
# include <iostream.h>

void main ()
{
    int i;
    i=0;

    do
    {
        cout << "C++" << endl;
        i++;
    }
    while (i<10);
}
```

5.3 PERNYATAAN *for*

Pernyataan **for** digunakan untuk menghasilkan pengulangan(looping) beberapa kali tanpa penggunaan kondisi apapun. Pada umumnya looping yang dilakukan oleh **for** telah diketahui batas awal, syarat looping dan perubahannya.

Pernyataan **for** digunakan untuk melakukan looping. Pada umumnya looping yang dilakukan oleh **for** telah diketahui batas awal, syarat looping dan perubahannya. Selama *kondisi* terpenuhi, maka pernyataan akan terus dieksekusi.

Bentuk Umumnya :

<pre>for (inisialisasi ; kondisi ; perubahan) { Statement; }</pre>
--

Contoh :

```
// Program mencetak angka 1-100
# include <iostream.h>

void main ()
{
    int i;

    for (i=1; i<=100; i++)
        cout << i << " " <<endl;
}
```


Bagaimana jika program diatas diubah menjadi

```
# include <iostream.h>

void main()
{
    int i;

    for (i=1; ;i++)
        cout << i << endl;
}
```

Program diatas akan menampilkan bilangan yang banyaknya tak terhingga sehingga dapat membuat komputer anda berhenti bekerja. Contoh diatas juga merupakan prinsip membuat bom program (contohnya : bom mail)

Pernyataan *for* dapat berada di dalam pernyataan *for* lainnya yang biasa disebut *nested for*

Contoh :

```
// Program membentuk segitiga
# include <iostream.h>

void main ()
{
    int tinggi,
        baris,
        kolom;

    cout << "Tinggi segitiga : ";
    cin >> tinggi;

    cout << endl;
    for (baris=1; baris <= tinggi; baris++)
    {
        for (kolom = 1; kolom <= baris; kolom++)
            cout << '*';
        cout << endl;
    }
}
```

5.3 PERNYATAAN *continue* dan *break*

Pernyataan *break* akan selalu terlihat digunakan bila menggunakan pernyataan *switch*. Pernyataan ini juga digunakan dalam loop. Bila pernyataan ini dieksekusi, maka akan mengakhiri loop dan akan menghentikan itrasi pada saat tersebut.

Pernyataan *continue* digunakan untuk pergi ke bagian awal dari blok loop untuk memulai iterasi berikutnya.

Contoh :

```
// Program Continue dan break

# include <iostream.h>
void main ()
{
    int i;
    for (i=0; i<10; i++)
    {
        if (i==4) continue;
        cout << " Bilangan " << i <<endl;
        if (i==6) break;
    }
}
```

Output :

```
Bilangan 0
Bilangan 1
Bilangan 2
Bilangan 3
Bilangan 5
Bilangan 6
```

Penjelasan :

Dari program diatas, dapat dilihat perulangan dari suatu bilangan sebanyak 10 kali. Tetapi, pada perulangan $i=4$, ada perintah *continue*. Dengan perintah ini, maka program langsung meloncat ke loop berikutnya dan ketika sampai perulangan $i = 6$, ada perintah *break*. Otomatis program akan berhenti dan tidak sampai ke $i=10$. Dan program akan mencetak bilangan 0, bilangan 1, bilangan 2, bilangan 3, bilangan 5, bilangan 6.

5.5 PERNYATAAN *go to*

Pernyataan **goto**, diperlukan untuk melakukan suatu lompatan ke suatu pernyataan berlabel yang ditandai dengan tanda “ : “.

Bentuk Umumnya :

```
goto bawah;
    pernyataan1;
    pernyataan2;
bawah : pernyataan 3;
```

Pada contoh diatas, pada saat goto ditemukan maka program akan melompat pernyataan berlabel bawah dan melakukan pernyataan 3.

Contoh :

```
# include <iostream.h>

void main ()
{
    cout << "Tes go to " <<endl;
    goto selesai;

    cout << "Hai, saya kok tidak disapa" << endl;

    selesai :
    cout << "Selesai... " << endl;
}
```

Outputnya :

```
Tes go to
Selesai...
```

Latihan :

1. Buatlah program untuk mencetak deret 10 9 8 7 6 5 4 3 2 1
2. Buatlah program untuk mencetak (gunakan perulangan while atau for)
* * * *
* * *
* *
*
3. Buatlah program yang menampilkan 5 buah bilangan, yaitu mulai dari bilangan ke 5 sampai bilangan ke 1 dengan nilai awal bilangan 8. Tampilan bilangan tersebut adalah menurun dan contohnya adalah : bilangan ke 5, $i=3$ (diperoleh dari $8-5$) dan seterusnya sampai bilangan 1, $i=7$ (diperoleh dari $8-1=7$)

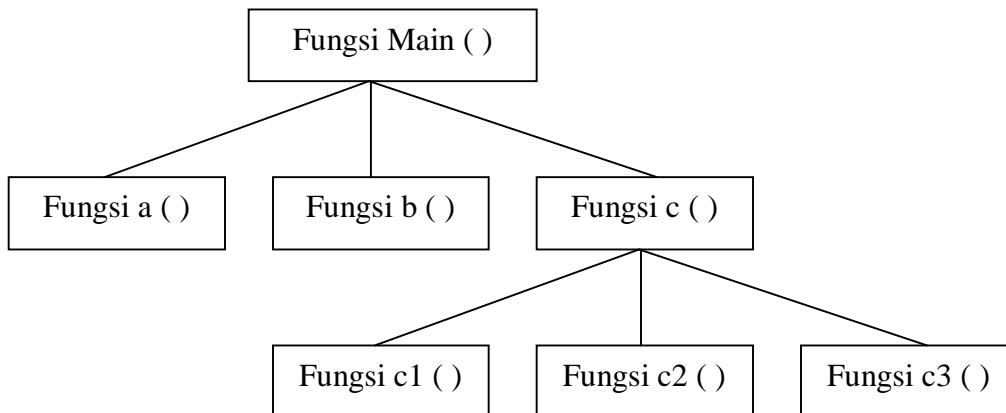
VI. FUNGSI

6.1. FUNGSI

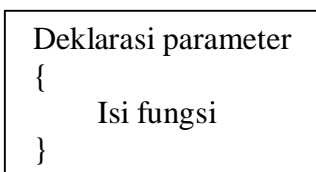
Fungsi adalah sekumpulan perintah operasi program yang dapat menerima argumen input dan dapat memberikan hasil output yang dapat berupa nilai ataupun sebuah hasil operasi. Hasil akhir fungsi akan berupa sebuah nilai balik (return)

Nama fungsi yang didefinisikan sendiri oleh pemrogram tidak boleh sama dengan nama build-in function pada compiler C++.

Fungsi digunakan agar pemrogram dapat menghindari penulisan bagian program (kode) berulang-ulang, dapat menyusun kode program agar terlihat lebih rapi dan kemudahan dalam debugging program. Parameter adalah nama-nama peubah yang dideklarsikan pada bagian header fungsi. Pemrogram dapat membuat fungsi yang didefinisikan sendiri olehnya.



Bentuk umumnya :



6.1.1 PROTITE FUNGSI

Sebuah fungsi tidak dapat dipanggil kecuali sudah dideklaraikan, deklarasi fungsi dikenal dengan sebutan prototipe fungsi. Prototipe fungsi berupa :

1. Nama Fungsi
2. Tipe nilai fungsi
3. Jumlah dan tipe argumen

Dan diakhiri dengan titik koma, sebagaimana pada pendeklarasian variabel. Sebagai contoh:

1. **long** kuadrat (**long l**) ;

Pada contoh pertama, fungsi kuadrat () mempunyai argumen bertipe long dan nilai balik bertipe **long**.

2. **void** garis () ;

Pada contoh kedua, fungsi garis () tidakmemiliki argumen dan nilai baliknya tidak ada (**void**).

3. **double** maks (**double x, double y**)

Pada contoh ketiga, fungsi maks() mempunyai dua buah argumen, dengan masing-masing argumen bertipe double.

Contoh :

```
# include <iostream.h>

double hasil (int A, int B) ;

void main()
{
    int x, y;
    double z;

    cout << "Masukkan Nilai x : ";
    cin >> x;

    cout << "Masukkan Nilai y : ";
    cin >> y;

    z = hasil (x, y) ;

    cout << "Hasil perkaliannya = ";
    cout << x << " x " << y << " = " << z;
}

double hasil (int A, int B)
{
    return (A * B) ;
} // Statement Mengembalikan Nilai
```

Penjelasan :

```
// Fungsi Perkalian
// tipe_return nama_fungsi (tipe_argument argumen)
double hasil (int A, int B)
{
    return (A * B);
} // Statement Mengembalikan Nilai
```

Fungsi yang didefinisikan oleh pemrogram terdiri atas dua bagian, yaitu judul (*header*) dan isi (*body*). Judul dari sebuah fungsi terdiri dari tipe return (**double**), nama fungsi (hasil) dan list parameter (**int A, int B**).

Jadi, judul untuk fungsi hasil adalah

```
double hasil (int A, int B)
```

Isi dari sebuah fungsi adalah blok kode yang mengikuti judulnya. Berisi kode yang menjalankan aksi dari fungsi, termasuk pernyataan *return* yang memuat nilai fungsi yang akan dikembalikan ke yang memanggilnya, Isi dari fungsi hasil () adalah

```
{
    return A * B;
}
```

Biasanya isi dari fungsi cukup besar. Meskipun demikian, judulnya tetap hanya berada dalam satu baris. Isi dari sebuah fungsi dapat memanggil fungsi itu sendiri (*disebut rekursif*) atau memanggil fungsi lainnya.

Pernyataan *return* dari sebuah fungsi mempunyai dua manfaat, yaitu akan mengakhiri fungsi dan mengembalikan nilainya ke program pemanggil.

Bentuk umum pernyataan return adalah :

```
return ekspresi;
```

Dengan *ekspresi* adalah sebuah ekspresi yang nilainya dinyatakan untuk sebuah variable yang tipenya sama seperti tipe **return**. Terdapat juga fungsi yang tidak memberikan nilai balik atau tipe **return**nya void.

Contoh :

```
#include <iostream.h>

void tampilkan_judul();
void main ()
{
    tampilkan_judul();
}

void tampilkan_judul()
{
    cout << "Universitas Sriwijaya" << endl;
    cout << "Kampus Diploma" << endl;
    cout << "Jalan Sriwijaya Negara Bukit Besar" << endl;
    cout << "Palembang 30139" << endl;
}
```

6.1.2 VARIABEL LOKAL DAN VARIABEL EKSTERNAL

Variabel lokal adalah variabel yang didefinisikan dalam suatu fungsi tertentu, sehingga hanya dikenal dalam fungsi tersebut. Dalam hal ini artinya suatu fungsi tidak akan mengenal variabel lokal dan fungsi lain. Suatu fungsi hanya akan mengenal variabel yang didefinisikan dalam fungsi yang bersangkutan.

Variabel eksternal adalah variabel yang bersifat global yang dapat dikenali oleh seluruh fungsi yang terdapat dalam program tersebut. Seluruh fungsi akan mengenal variabel yang bersifat eksternal ini. Variabel eksternal dideklarasikan diluar fungsi dan sejajar dengan prototipe fungsi serta pengarah kompiler.

Contoh :

```

// nama program : clokak_eksternal.cpp
// contoh program variabel lokal dan eksternal

# include <iostream.h>

int data = 100;      // variabel eksternal
void fungsi_satu (); // prototipe fungsi
void fungsi_dua ();
void main ()
{
    int data = 200; // variabel lokal main
    fungsi_satu();
    fungsi_dua();
    cout << "Nilai data lokal main = " << data << endl;
}

void fungsi_satu()
{
    int data = 300; // variabel lokal fungsi_satu
    cout << "Nilai data lokal satu = " << data << endl;
}

void fungsi_dua()
{
    cout << "Nilai data eksternal = " << data << endl;
}

```

Outputnya :

```

Nilai data lokal satu = 300
Nilai data eksternal = 100
Nilai data lokal main = 200

```

Penjelasan :

Dalam pemrograman tersebut terdapat variabel lokal dan variabel eksternal yang namanya sama yaitu data. Dalam fungsi main () dan fungsi_satu () terdapat variabel lokal dengan nama sama tetapi sebetulnya lokasi penyimpanannya dalam memori berbeda, sehingga dua variabel itu berbeda dan tidak saling mengenal. Fungsi_satu () sebetulnya mengenal variabel eksternal data yang nilainya 100, tetapi karena dalam fungsi terdapat variabel lokal data yang bernilai 300, maka diprioritaskan untuk diproses dalam fungsi tersebut adalah variabel lokalnya. Jika dalam fungsi terdapat variabel lokal dan variabel eksternal yang sama, maka diprioritaskan untuk diproses variabel lokal. Dalam fungsi_dua() tidak terdapat variabel lokal sehingga yang diproses pada fungsi tersebut adalah variabel eksternalnya.

6.1.3 PARAMETER


Parameter adalah sarana komunikasi antar fungsi. Pengertian antar fungsi adalah antara fungsi dengan fungsi lain termasuk antara fungsi dengan fungsi utama. Dalam pemrograman yang melibatkan fungsi, diusahakan agar fungsi bersifat independen artinya tidak tergantung pada fungsi lain. Setiap fungsi hanya mengerjakan satu tugas tertentu. Antar fungsi saling berkomunikasi menggunakan parameter.

Terdapat dua macam bentuk parameter dalam hubungannya dengan penggunaan fungsi dalam program yaitu :

- Parameter Formal : parameter yang diberi nilai. Parameter formal merupakan parameter yang terdapat dalam daftar parameter fungsi.
- Parameter Aktual : parameter yang memberi nilai. Parameter fungsi dan digunakan untuk memberi nilai pada parameter formal.


Dalam contoh program perkalian di atas parameter formal terdapat pada pendefinisian fungsi :

```
double hasil(int A, int B) // parameter formal
{
    return (A * B);
}
```



Sedangkan parameter aktual terdapat pada pemanggilan fungsi :

```
void main()
{ .....
.....
z = hasil(x,y); // parameter aktual
.....
}
```



6.1.3.1 Cara Melewatkan Parameter

Cara melewati suatu parameter dalam Bahasa C++ ada dua cara yaitu :

1. Pemanggilan Secara Nilai (*Call by Value*)
 - a) Call by value akan menyalin nilai dari parameter aktual ke parameter formal.

- b) Yang dikirimkan ke fungsi adalah nilai dari datanya, bukan alamat memori letak dari datanya.
- c) Fungsi yang menerima kiriman nilai akan menyimpannya di alamat terpisah dari nilai aslinya yang digunakan oleh bagian program yang memanggil fungsi.
- d) Perubahan nilai di fungsi (parameter formal) tidak akan merubah nilai asli di bagian program yang memanggilnya.
- e) Pengiriman parameter secara nilai adalah pengiriman searah, yaitu dari bagian program yang memanggil fungsi ke fungsi yang dipanggil.
- f) Pengiriman suatu nilai dapat dilakukan untuk suatu ungkapan, tidak hanya untuk sebuah variabel, elemen array atau konstanta saja.

Contoh :

```
#include<iostream.h>
/*Contoh program transfer by value*/

int Tambah(int x);
void main()
{
    int a,hasil;

    cout<<"Masukkan Bilangan: ";
    cin>>a;

    cout<<"a awal= "<<a<<endl;

    hasil = Tambah(a);

    cout<<"a akhir= "<<a<<endl;

    cout<<"Hasil : "<<hasil;
}

int Tambah(int x)
{
    cout<<"x awal = "<<x<<endl;

    x = x + 2;

    cout<<"x akhir = "<<x<<endl;

    return x;
}
```

Outputnya :

```
Masukkan Bilangan: 8
a awal= 8
x awal = 8
x akhir = 10
a akhir= 8
Hasil : 10
```

2. Pemanggilan Secara Referensi (*Call by Reference*)

- a) Pemanggilan secara Referensi merupakan upaya untuk melewatkan alamat dari suatu variabel ke dalam fungsi.
- b) Yang dikirimkan ke fungsi adalah alamat letak dari nilai datanya, bukan nilai datanya.
- c) Fungsi yang menerima kiriman alamat ini maka menggunakan alamat yang sama untuk mendapatkan nilai datanya.
- d) Perubahan nilai di fungsi akan merubah nilai asli di bagian program yang memanggil fungsi.
- e) Pengiriman parameter secara referensi adalah pengiriman dua arah, yaitu dari fungsi pemanggil ke fungsi yang dipanggil dan juga sebaliknya.
- f) Pengiriman secara acuan tidak dapat dilakukan untuk suatu ungkapan.

Contoh :

```

#include<iostream.h>
/*Contoh program transfer by referensi*/

int Tambah(int &x); //x diberi & tuk referensi
                  //sehingga nilai a akan sll mengikuti nilai x
                  //krn var a dan x berada dlm satu alamat memori

void main()
{
    int a,hasil;

    cout<<"Masukkan Bilangan: ";
    cin>>a;

    cout<<"nilai a awal= "<<a<<endl;
    hasil = Tambah(a);
    cout<<"nilai a akhir= "<<a<<endl;
    cout<<"Hasil : "<<hasil;
}

int Tambah(int &x)
{
    cout<<"nilai x awal = "<<x<<endl;
    x = x + 2;
    cout<<"nilai x akhir = "<<x<<endl;
    return x;
}

```

Outputnya :

```

Masukkan Bilangan: 8
nilai a awal= 8
nilai x awal = 8
nilai x akhir = 10
nilai a akhir= 10
Hasil : 10

```

6.1.4 NILAI BAWAAN UNTUK ARGUMEN FUNGSI

Salah satu keistimewaan C++ yang sangat bermanfaat dalam pemrograman adalah adanya kemampuan untuk menyetel nilai *default*

Argumen fungsi. Argumen-argumen yang mempunyai nilai bawaan nantinya dapat tidak disertakan di dalam pemanggilan fungsi dan dengan sendirinya C++ akan menggunakan nilai bawaan dari argumen yang tidak disertakan.

Contoh :

```
#include <iostream.h>
# include <conio.h>

void sayHello(int n);

void main()
{
    sayHello(1);
}

void sayHello(int n)
{
    for (int m=0;m<n;m++)
        cout<<"Halloo :\n";
}
```

Penjelasan :

Jika pada program, argumen sayHello tidak diberikan, maka program akan menampilkan

Halloo :

Sebanyak satu kali, namun jika argumen pada fungsi sayHello diberikan, misalkan *sayHello(4)*, maka program akan menampilkan 4 kali. Itulah yang disebut dengan nilai default pada fungsi.

6.1.4 REKURSI

Merupakan suatu fungsi dapat memanggil fungsi yang merupakan dirinya sendiri. Penerapan rekursi diantaranya untuk menghitung nilai : x^n

Contohnya :

```

// Operasi pangkat secara rekursi

# include <iostream.h>

long int pangkat (int x, int n);
void main ()
{
    int x, y;

(   cout << " Menghitung x ^ y " << endl;
    cout << " x = ";
    cin >> x;
    cout << " y = ";
    cin >> y;

    cout << x << " ^ " << y << " = " << pangkat(x,y)<< endl;
}

long int pangkat (int x, int n)
{
    if (n==1)
        return (x);
    else
        return ( x* pangkat (x, n-1));
}

```

6.1.5 FUNGSI-FUNGSI BAWAAN C++

Anda dapat menggunakan fungsi-fungsi bawaan C++, misalkan fungsi-fungsi matematika, pengolah kata dan banyak lagi. Sebenarnya (mungkin tidak terasa bagi anda) main juga adalah fungsi, jadi tanpa anda sadari sebenarnya anda telah menggunakan fungsi.

Untuk dapat menggunakan fungsi-fungsi tersebut anda harus meng-include file dimana fungsi tersebut didefinisikan

Misalkan :

- Fungsi – fungsi matematika, anda harus meng-include file math.h
- Fungsi – fungsi pengolah string dan karakter, anda harus meng-include file string.h
- Fungsi clrscr(), getch(), getche() dalam file conio.h

LATIHAN

1. Buatlah fungsi untuk menghitung luas segitiga?
2. Buatlah program rekursi untuk mencari Nilai n faktorial
3. Buatlah program dengan cara rekursi untuk menampilkan perkalian 3 buah bilangan tersebut nilainya diinputkan

VII. LARIK (ARRAY)

7.1. LARIK

Larik merupakan sekumpulan data yang mempunyai nama dan tipe yang sama. Larik sering disebut juga variabel berindeks. Nilai suatu data dalam larik ditentukan oleh nama dan indeks. Larik banyak digunakan pada operasi yang melibatkan indeks seperti pada statistik dan matriks.

Tipe data larik dapat berupa larik satu dimensi, dua dimensi, tiga dimensi atau banyak dimensi.

Bentuk Umum Larik Satu Dimensi :

```
tipe_larik nama_larik [ukuran]
```

Bentuk Umum Larik Dua Dimensi :

```
tipe_larik nama_larik [ukuran1][ukuran2]
```

Perhatikan :

- Tanda kurung [] digunakan untuk menunjukkan elemen larik
- Perhitungan elemen larik dimulai dari 0, bukan 1

C++ tidak mengecek larik. Bila anda menyatakan `int x[10]`, ini artinya 10 elemen yang dimulai dari 0. Karena itu elemen terakhir larik adalah `x[9]`. Bila anda salah mereferensikannya dengan `x[10]`, anda akan mendapatkan harga yang tidak terpakai. Akan lebih buruk lagi jika anda memberikan harga ke `x[10]`, yang tidak dapat diterima.

7.2 REPRESENTASI LARIK

Misalkan kita memiliki sekumpulan data *ujian* seorang siswa, *ujian* pertama bernilai 90, kemudian 95,78,85. Sekarang kita ingin menyusunnya sebagai suatu data kumpulan *ujian* seorang siswa. Dalam array kita menyusunnya sebagai berikut

```
ujian[0] = 90;
ujian[1] = 95;
ujian[2] = 78;
ujian[3] = 85;
```

Empat pernyataan diatas memberikan nilai kepada array *ujian*. Tetapi sebelum kita memberikan nilai kepada array, kita harus mendeklarasikannya terlebih dahulu, yaitu :

```
int ujian[4];
```

Perhatikan bahwa nilai 4 yang berada didalam tanda kurung menunjukkan jumlah elemen larik, bukan menunjukkan elemen larik yang ke-4. Jadi elemen larik *ujian* dimulai dari angka 0 sampai 3.

Pemrogram juga dapat menginisialisasi larik sekaligus mendeklarasikannya, sebagai contoh :

```
int ujian[4] = {90,95,78,85};
```

Elemen terakhir dari larik diisi dengan karakter '\0'. Karakter ini memberitahu kompiler bahwa akhir dari elemen larik telah dicapai. Walaupun pemrogram tidak dapat melihat karakter ini secara eksplisit, namun kompiler mengetahui dan membutuhkannya.

Sekarang kita akan membuat daftar beberapa nama pahlawan di Indonesia

```
char pahlawan[3][15] ;
char pahlawan[0][15] = "Soekarno";
char pahlawan[1][15] = "Diponegoro";
char pahlawan[2][15] = "Soedirman";
```

Larik diatas terlihat berbeda dengan contoh larik pertama kita. Perhatikan bahwa pada larik *pahlawan* memilih dua buah tanda kurung [][]. Larik seperti itu disebut larik dua

dimensi. Tanda kurung pertama menyatakan total elemen yang dapat dimiliki oleh larik pahlawan dan tanda kurung kedua menyatakan total elemen yang dapat dimiliki setiap elemen larik pahlawan. Dalam contoh di atas, tanda kurung kedua menyatakan karakter yang menyatakan nama pahlawan.

7.3 MENGHITUNG JUMLAH ELEMEN ARRAY

Karena fungsi `sizeof()` mengembalikan jumlah byte yang sesuai dengan argumennya, maka operator tersebut dapat digunakan untuk menemukan jumlah elemen array, misalnya

```
int array[ ] = {26,7,82,166};  
cout<<sizeof(array)/sizeof(int);
```

akan mengembalikan nilai 4, yaitu sama dengan jumlah elemen yang dimiliki larik.

7.4 MELEWATKAN ARRAY SEBAGAI ARGUMEN FUNGSI

Larik dapat dikirim dan dikembalikan oleh fungsi. Pada saat larik dikirim ke dalam fungsi, nilai aktualnya dapat dimanipulasi

Contoh :

```
#include <iostream.h>  
void ubah(int x[]);  
void main()  
{  
    int ujian[] = {90,95,78,85};  
    ubah(ujian);  
    cout<<" Elemen kedua dari array ujian adalah "<<ujian[1]<<endl;  
}  
  
void ubah(int x[])  
{  
    x[1] = 100;  
}
```

Output :

Elemen kedua dari array ujian adalah 100

Latihan :

Buatlah program yang menghitung jumlah elemen dalam suatu array(larik) dengan array(larik) 1 dimensi { 1,3,5,4,7,2,99,16,45,67,89,45}