# MODUL PRAKTIKUM
# ALGORITMA PEMROGRAMAN


**TIM PENYUSUN:**

**Dr. Andi Tenriawaru, S.Si., M.Si.**

**Perdiansyah (F1G121081)**

**Kalingga Sakti Satriantara (F1G122020)**

**Nazwah Thalbiatul Ilmi Rahman (F1G122057)**



**FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM**

**PROGRAM STUDI ILMU KOMPUTER**

**UNIVERSITAS HALU OLEO**

**KENDARI**

**2023**

# KATA PENGANTAR

Assalamu’alaikum Warahmatullahi Wabarakatuh

Alhamdulillah kami panjatkan puja dan puji syukur kehadirat Allah swt yang senantiasa melimpahkan segala rahmat, taufik dan hidayah-Nya sehingga penyusun dapat menyelesaikan modul ini.

Pengembangan teknologi informasi dan komputasi telah mengubah wajah dunia secara mendasar. Di balik perkembangan teknologi tersebut, terdapat fondasi yang kokoh yang dikenal sebagai algoritma pemrograman. Algoritma adalah panduan langkah demi langkah untuk menyelesaikan masalah, dan pemrograman adalah implementasi dari algoritma tersebut ke dalam bahasa komputer. Dalam dunia yang semakin terkoneksi dan dijejali dengan data, pemahaman yang kuat tentang algoritma pemrograman adalah suatu keharusan.

Dalam modul ini, Anda akan diajak untuk memahami konsep-konsep dasar algoritma, serta mengimplementasikannya dalam bahasa pemrograman tertentu. Kami telah merancang materi ini agar dapat diikuti dengan mudah, bahkan jika Anda baru memulai perjalanan dalam dunia pemrograman. Modul ini akan membantu Anda membangun fondasi yang kuat dalam pemahaman algoritma dan pemrograman.

Penyusun menyadari bahwa di dalam pembuatan modul masih banyak kekurangan, untuk itu penyusun sangat membuka saran dan kritik yang sifatnya membangun. Mudah-mudahan modul ini memberikan manfaat.

Kendari, 13 September 2023

Penyusun

# DAFTAR ISI

# BAB I
# PENDAHULUAN

## 1.1 Algoritma

Algoritma biasanya didefinisikan sebagai rangkaian terurut langkah-langkah yang logis dan sistematis yang disusun untuk menyelesaikan suatu masalah. Tujuan algoritma adalah memberikan petunjuk tentang langkah-langkah logika penyelesaian masalah dalam bentuk yang mudah dipahami nalar manusia sebagai acuan yang membantu dalam mengembangkan program komputer. Pemahaman terhadap algoritma akan mencegah sejak dini kemungkinan terjadinya kesalahan logika pada program komputer yang dikembangkan.

**Penulisan (Notasi) Algoritma :**

Ada tiga macam bentuk notasi algoritma antara lain :

**a. Kalimat Dekript*if***

Notasi algoritma dengan menggunakan kalimat deskript*if* disebut juga notasi alami. otasi algoritma deskript*if* dilakukan dengan cara menuliskan intruksi-intruksi yang musti dilaksanakan dalam bentuk untaian kalimat deskript*if* dengan menggunakan bahasa yang jelas. Contoh penulisan algoritma dengan notasi deskript*if*, sebagai berikut :

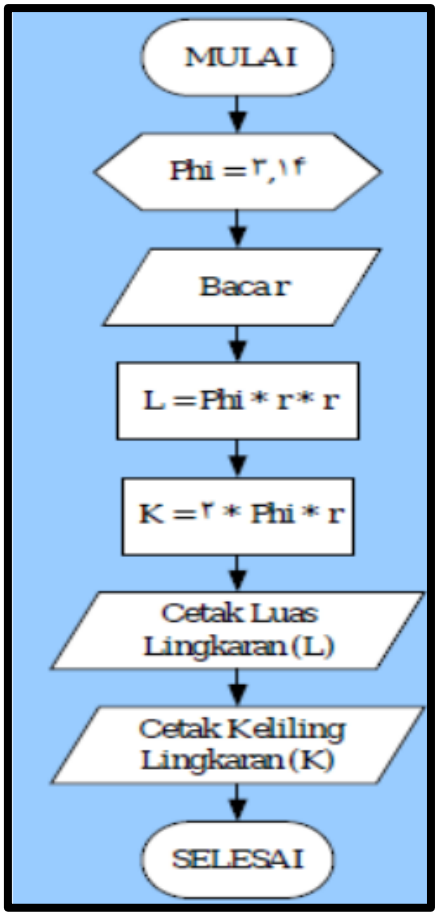
Algoritma_Hitung_Luas_dan_Keliling_Lingkaran

- Masukkan jari-jari lingkaran ®
- Hitung luas lingkaran dengan rumus L = p* $r^2$
- Hitung keliling dengan rumus K = 2 * p * r
- Tampilkan luas lingkaran
- Tampilkan keliling lingkaran

**b. *Flowchart***

*Flowchart* adalah gambaran dalam bentuk diagram alir dari algoritma-algoritma dalam suatu program, yang menyatakan arah alur program tersebut.

Contoh : menghitung luas dan kelliling lingkaran yang algoritmanya dinotasikan dalam bentuk *flowchart*.




**c. Kode Semu / *Pseudocode***

Notasi yang menyerupai notasi Bahsa pemrograman tingkat tinggi, misalnya Bahasa Pascal dan C.

Struktur algoritma dibagi ke dalam beberapa bagian di antaranya :

1. Bagian kepala (*Header*)
2. Bagian deklarasi (definisi *variable*)
3. Bagian deskripsi (rincian langkah)

**Contoh :**

**Algoritma Luas_persegi_panjang**

{Menghitung sebuah luas persegipanjang apabila panjang dan lebar persegipanjang tersebut diberikan}

**Deklarasi**

{Definisi nama peubah/variabel}

float panjang, lebar, luas

**Deskripsi**

read(panjang,lebar) // bisa juga : input / baca

luas -> panjang * lebar

write(Luas) // bisa juga : *ouput* / tulis

## 1.2 Pemrograman

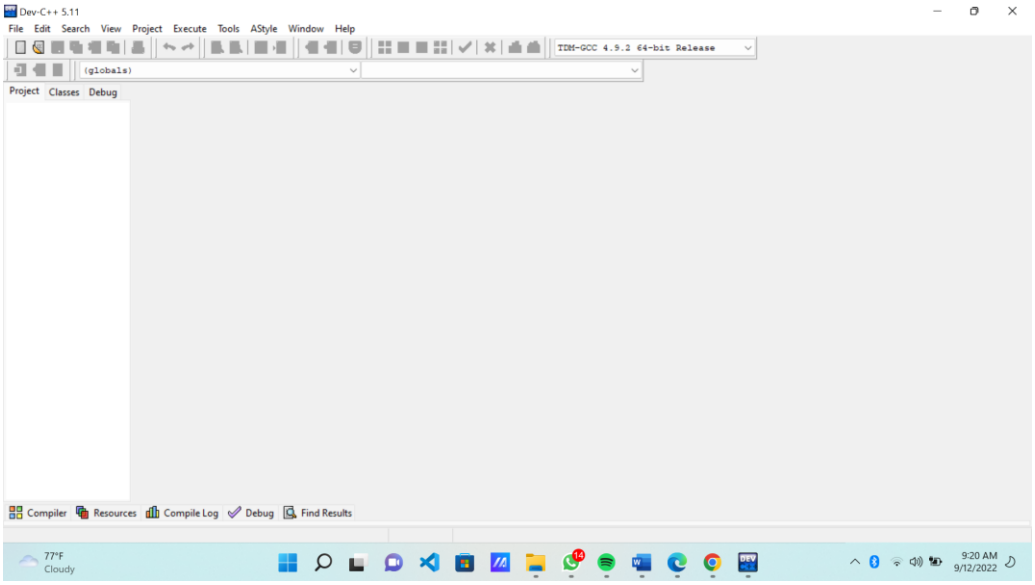
Program secara umum didefinisikan sebagai kumpulan instruksi atau perintah yang disusun sedemikian rupa sehingga mempunyai urutan nalar yang logis untuk menyelesaikan suatu persoalan yang dimengerti oleh komputer. Pemrograman adalah aktivitas yang berhubungan dengan pembuatan program dengan mengikuti kaidah bahasa pemrograman tertentu. Dalam konteks pemrograman terdapat sejumlah bahasa pemrograman seperti Pascal, Delphi, C, C++, C#, Java, dll. Dari contoh algoritma mencari luas persegi panjang bisa dihasilkan program dengan menggunakan beberapa Bahasa pemrograman seperti C++, sebagai berikut.

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

        main() {
           float   panjang,   lebar,
luas;
           cout<<"Panjang : ";
           cin>>panjang;
           cout>>"Lebar : ";
           cin>>lebar;

           luas = panjang * lebar;
           cout<<"Luas              :
"<<luas<<endl;
        }
```

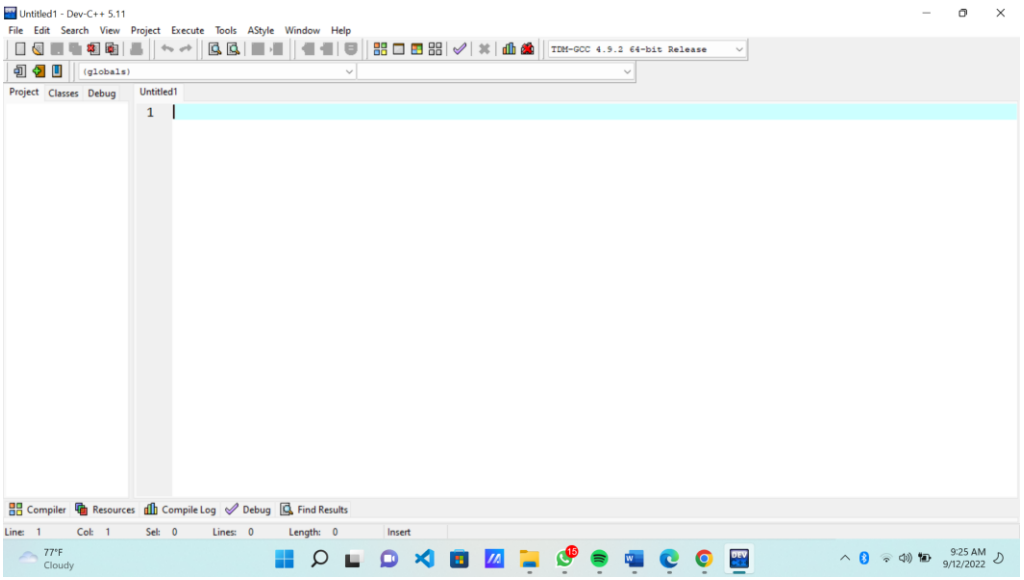
## 1.3 Menuliskan program C++ dengan Dev-C++

Percobaan 1.1. Mencoba *compiler* dan membuat program luas segitiga, Adapun langkah-langkahnya adalah sebagai berikut :

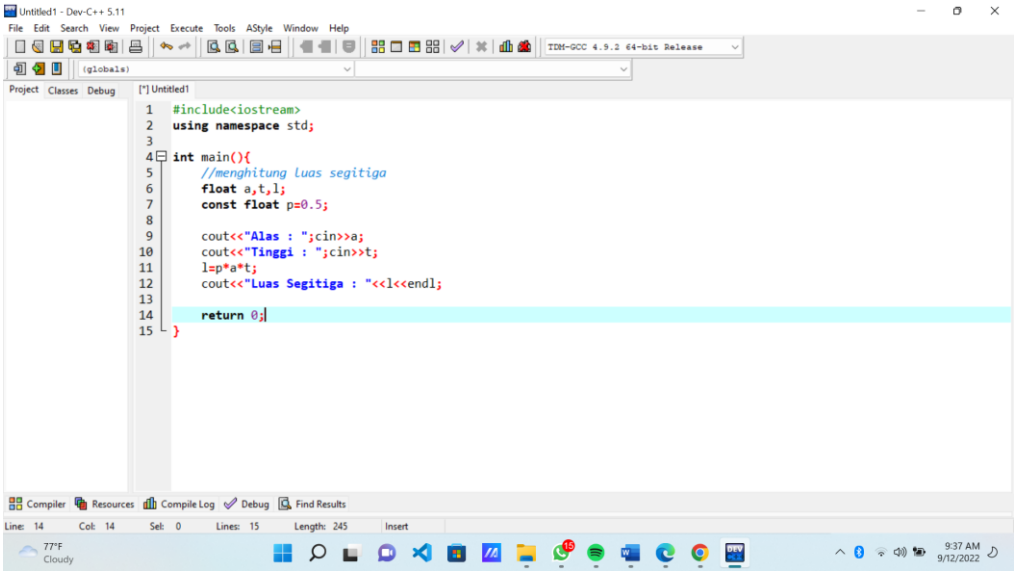
1. Download dan Instal aplikasi Dev-C++ di link berikut https://sourceforge.net/projects/orwelldevcpp/
2. Bukalah software Dev-C++ jika sudah diinstal, gambarannya adalah seperti gambar di bawah ini :



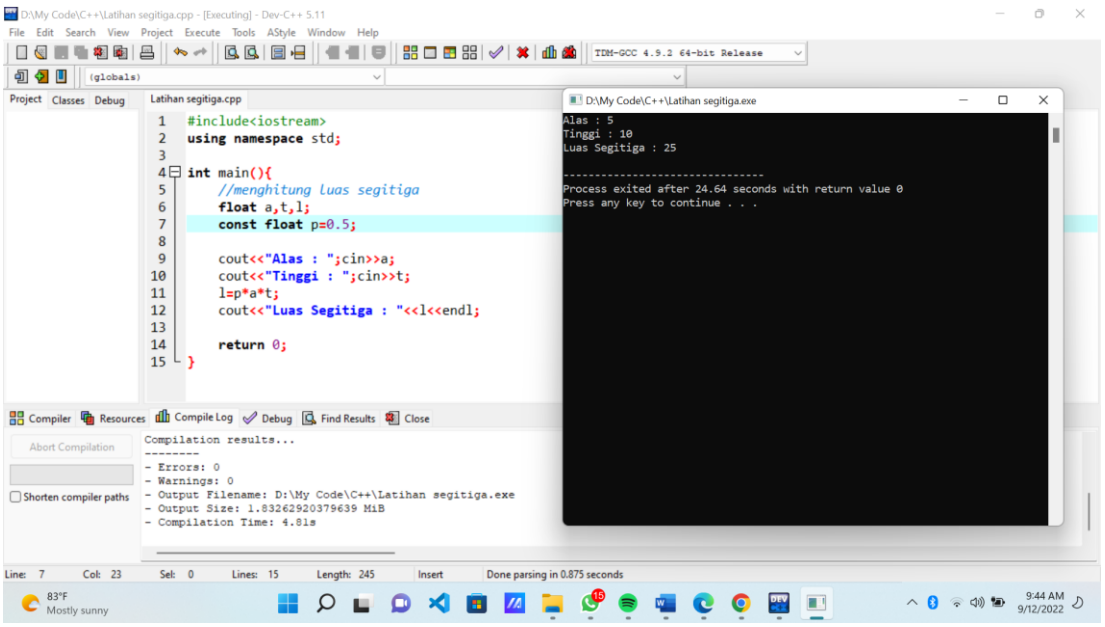

3. Klik **File**>>**New, Pilih Source File** atau lebih singkatnya bisa diklik pada tombol keyboard ctrl+n untuk membuat halaman baru di tampilan aplikasi **Dev-C++**, tampilan gambarnya adalah berikut ini :




4. Tulis kode program yang ingin Anda buat, misalnya seperti gambar di bawah ini :




5. Simpan file yang Anda buat (Ctrl+S), lalu compile program Anda dengan mengklik **Execute>>Compile&Run** atau **tekan F11,** hasil tampilannya adalah sebagai berikut :



# BAB II
# PENGENALAN BAHASA C++

## 2.1. Dasar Bahasa Pemrograman C++

C++ adalah bahasa pemrograman komputer yang dibuat oleh Bjarne Stroustrup, yang merupakan perkembangan dari bahasa C dikembangkan di Bell Labs (Dennis Ritchie). Pada awal tahun 1970-an, bahasa itu merupakan peningkatan dari bahasa sebelumnya, yaitu B. Bahasa pemrograman C++ adalah bahasa yang bisa digunakan untuk membuat berbagai aplikasi. Misalnya, aplikasi pengolah gambar, *software gadget*, *game*, hingga sistem operasi baru.

Dalam mempelajari bahasa C++ ada beberapa konsep dasar yang perlu dipahami, yaitu sebagai berikut :

a. *Header*

*File Header* pada C++ adalah pernyataan pendeklarasian utama pada sebuah program C++ yang berfungsi untuk memanggil dan menjalan fungsi-fungsi yang terdapat dalam *library file header* C++ agar kita dapat menggunakan fungsi pada file tersebut. Pada bagian #include memberitahuakan preprosesor untuk menyertakan kode dari kode didalam kurung <.....> yang berisi deklarasi/perintah untuk berbagai fungsi yang dibutuhkan oleh perangkat lunak, atau class-class yang dibutuhkan.

b. Penulisan *Variable*

Variabel adalah penanda identitas yang digunakan untuk menampung suatu nilai. Artinya, variabel akan menunjukkan suatu lokasi yang ada di memori komputer atau RAM. Jadi, saat Anda membuat satu variabel, akan ada satu slot memori untuk menampung nilai tersebut. Terdapat beberapa aturan penamaan *variable* yang harus diperhatikan untuk menghindari terjadinya eror pada program, yaitu adalah sebagai berikut :

- Variabel bisa terdiri dari huruf, angka dan karakter *underscore* / garis bawah ( _ ).
- Karakter pertama dari variabel hanya boleh berupa huruf dan *underscore* ( _ ), tidak bisa berupa angka. Meskipun dibolehkan, sebaiknya tidak menggunakan karakter *underscore* sebagai awal dari variabel karena bisa bentrok dengan beberapa variabel settingan program.

- Variabel harus selain dari *keyword*. Sebagai contoh, kita tidak bisa memakai kata **int** sebagai nama variabel, karena **int** merupakan *keyword* untuk menandakan tipe data **integer**.
- Beberapa *compiler* bahasa C++ ada yang membatasi panjang variabel maksimal 31 karakter. Agar lebih aman, sebaiknya tidak menulis nama variabel yang lebih dari 31 karakter.

c. Tipe Data

Secara sederhana, tipe data adalah cara kita memberitahu komputer untuk mengelompokkan data berdasarkan apa yang dipahami oleh komputer. Tiap *variable* yang dituliskan harus kita deklarasikan terlebih dahulu tipe data yang akan digunakan oleh *variable* tersebut. Adapun contoh *table* tipe data pada bahasa pemrograman c++ adalah sebagai berikut :

| Tipe Data | Byte | Batasan |
|---|---|---|
| char | 1 | Bilangan bulat / ASCII antara -128 s.d. 127 |
| unsigned char | 1 | Bilangan bulat antara 0 s.d. 255 |
| short | 2 | Bilangan bulat antara -32.768 s.d. 32.767 ($-2^{15}$ s.d. $2^{15}-1$) |
| unsigned short | 2 | Bilangan bulat antara 0 s.d. 65.535 (0 s.d. $2^{16}-1$) |
| int | 4 | Bilangan bulat antara -2.147.483.648 s.d. 2.147.483.647 ($-2^{31}$ s.d. $2^{31}-1$) |
| unsigned int / unsigned | 4 | Bilangan bulat antara 0 s.d. $2^{32}-1$ |
| long int | 4 | Bilangan bulat antara $-2^{31}$ s.d. $2^{31}-1$ |
| unsigned long int | 4 | Bilangan bulat antara 0 s.d. $2^{32}-1$ |
| float | 4 | Bilangan real antara -3.4E+38 s.d.3.4E+38 (7 digit presisi) |
| double | 8 | Bilangan real antara 1.7E+308 s.d. 1.7E+308 (15 digit presisi) |

## 2.2. Syntax Dasar C++

Bentuk atau struktur dasar program yang dibuat dengan C++ terdiri dari tiga bagian :

a. Bagian *Include*

Pada bagian ini, kita mendefinisikan *library* (pustaka) apa saja yang akan kita gunakan di dalam program. *Library* yang biasa digunakan yaitu *library* iostream yang terdapat fungsi input dan *ouput* yang bisa dipakai contohnya *cin* dan *cout*. Selain library iostream terdapat beberapa library yang bis akita gunakan tergantung kebutuhan yang diinginkan, contohnya sebagai berikut :

a. #include<stdio.h>

b. #include<conio.h>
c. #include<math.h>
d. #include<stdlib.h>
e. #include<string.h>
f. #include<iomanip.h>
g. Dan lain sebagainya.

b. Bagian namespace

Using namespace std, perintah ini digunakan untuk mendeklarasikan / memberitahukan kepada *compiler* bahwa kita akan menggunakan semua fungsi / *class* / *file* yang terdapat dalam namespace std. Namespace sendiri memiliki kesamaan dengan paket pada Bahasa java yang berisi pengelompokan fungsi, *class* dan yang sejenis. Pada *library-library* umumnya disimpan dalam namespace std, seperti cin dan cout. Bisa dilihat sebuah perbedaan penulisan apabila kita menggunakan using namespace std dan tidak :

- Tanpa menggunakan using namespace std

```
std::cout<<"tanpa menggunakan using namespace std";
std::cin>>a;
```

- Menggunakan using namespace std

```
#include<iostream>
using namespace std;
int main(){
        cout<<"menggunakan using namespace std";
        cin>>a;
}
```

c. Bagian fungsi main

Fungsi main merupakan fungsi utama sebuah program. Fungsi ini juga akan dieksekusi untuk pertama kalinya. Oleh karena itu, kamu harus selalu menyertakan fungsi main() dalam sebuah program, contohnya seperti ini :

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

int main(){
    //kode program
}
```

Berikut adalah *syntax* dasar Bahasa pemrograman c++.

**Percobaan 2.2.1** Membuat program sederhana *hello world* dan komentarnya

```
#include<iostream>      //header program
using namespace std;

int main(){        //fungsi utama bahasa c++
     cout<<”Hello World”;
     /*ini adalah komentar*/
 return 0;
}
```

# BAB III
# PENDEKLARASIAN DAN PENCETAKAN VARIABEL

## 3.1. Mendeklarasikan Variabel

Deklarasi itu sangat diperlukan ketika kita akan menggunakan pengenalan (atau bisa disebut *identifier*) dalam pemrograman, pengenalan atau *identifier* bisa berupa *variable* konstanta dan fungsi. Nah kali ini yang sering digunakan adalah DEKLARASI VARIABEL sendiri itu kita mendeklarasikan sebuah *variable* yang sudah kita buat yaitu sebuah nama. Deklarasi *variable* terdiri dari *variable* yang kita buat dengan menyertakan tipe data yang tentunya tipe data tersebut berada di awal, berikut contohnya :

```
TipeData Variabel;
```

Contoh penulisannya :

- int x; = huruf x sebagai variable dan int sebagai tipe data integer
- char x; = huruf x sebagai variable dan char sebagai tipe data berupa char
- float x; = huruf x sebagai variable dan tipe datanya adalalah float

## 3.2. Menginisialisasi Variabel

Dalam konteks ini, inisialisasi dapat didefinisikan sebagai proses pengisian nilai awal (nilai *default*) ke dalam sebuah variable. Dalam C++, pengisian nilai dilakukan dengan menggunakan operator sama dengan (=). Bentuk umum yang digunakan untuk melakukan inisialisasi *variable* adalah sebagai berikut :

```
Tipe_data nama_variabel = nilai_awal;
```

ATAU

```
Tipe_data nama_variabel = nilai_awal1, nama_variabel2 = nilai_awal2,
                                    nama_variabel3 = nilai_awal3
```

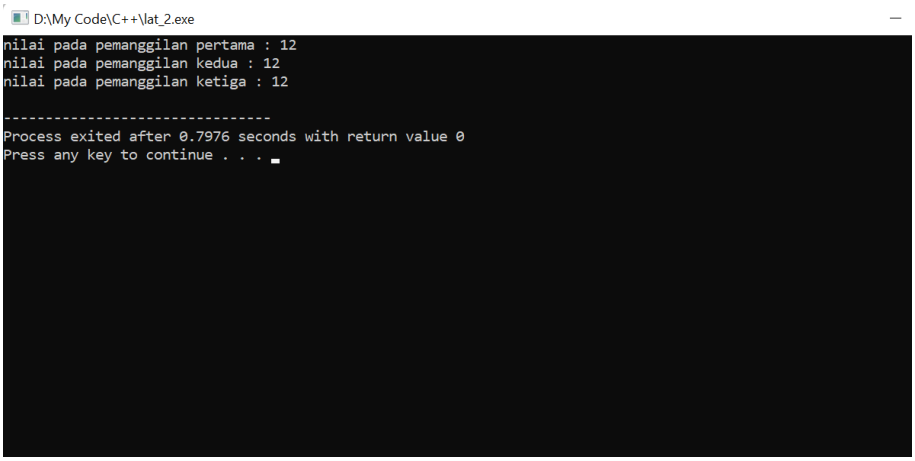
## 3.3. Variabel Statis Lokal

Variabel statis local ini diterapkan di dalam suatu fungsi/prosedur sehingga nama variabelnya hanya akan dikenali di dalam fungsi tempat pendeklarasiannya. Namun, perlu diperhatikan bahwa nilai terakhir yang dihasilkan akan terus disimpan. Dengan demikian, setiap pemanggilan fungsi yang sama pasti akan memberikan hasil yang berbeda.

Untuk mengetahui lebih jauh mengenai variable statis llokal, perhatikan terlebih dahulu dua contoh program di bawah ini :

```
#include<iostream>
using namespace std;
//membuat fungsi dengan nama latihan
int latihan(){
   //mendefinisikan variabel biasa
   int m = 0;
   m=m+12;
   return m;
}
int main(){
   //mendefinisikan variabel x,y,z;
   int x,y,z;
   //memanggil fungsi latihan() dengan variabel yang berbeda
   x=latihan();
   y=latihan();
   z=latihan();
   //menampilkan nilai dari variabel
   cout<<"nilai pada pemanggilan pertama : "<<x<<endl;
   cout<<"nilai pada pemanggilan kedua : "<<y<<endl;
   cout<<"nilai pada pemanggilan ketiga : "<<z<<endl;

   return 0;
}
```

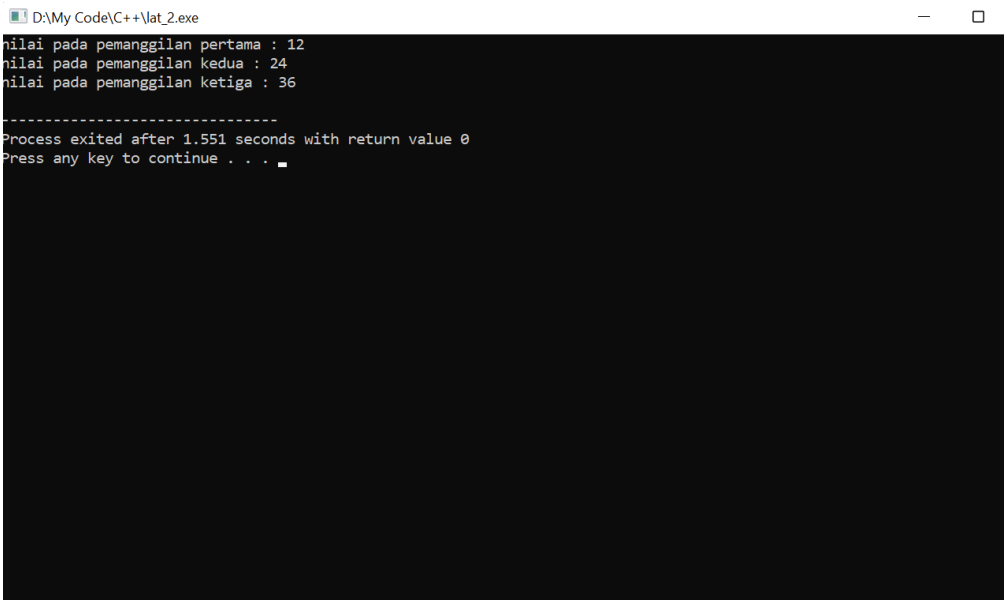
Hasil *ouput*nya akan menjadi seperti di bawah ini




Jika variabel m diubah menjadi variabel statis, seperti program yang ditunjukkan di bawah ini :

```
#include<iostream>
using namespace std;
//membuat fungsi dengan nama latihan
int latihan(){
   //mendefinisikan variabel biasa
   static int m = 0;
   m=m+12;
   return m;
}
int main(){
   //mendefinisikan variabel x,y,z;
   int x,y,z;
   //memanggil fungsi latihan() dengan variabel yang berbeda
   x=latihan();
   y=latihan();
   z=latihan();
   //menampilkan nilai dari variabel
   cout<<"nilai pada pemanggilan pertama : "<<x<<endl;
   cout<<"nilai pada pemanggilan kedua : "<<y<<endl;
   cout<<"nilai pada pemanggilan ketiga : "<<z<<endl;

   return 0;
}
```

Hasil *ouput*nya akan menjadi seperti di bawah ini :

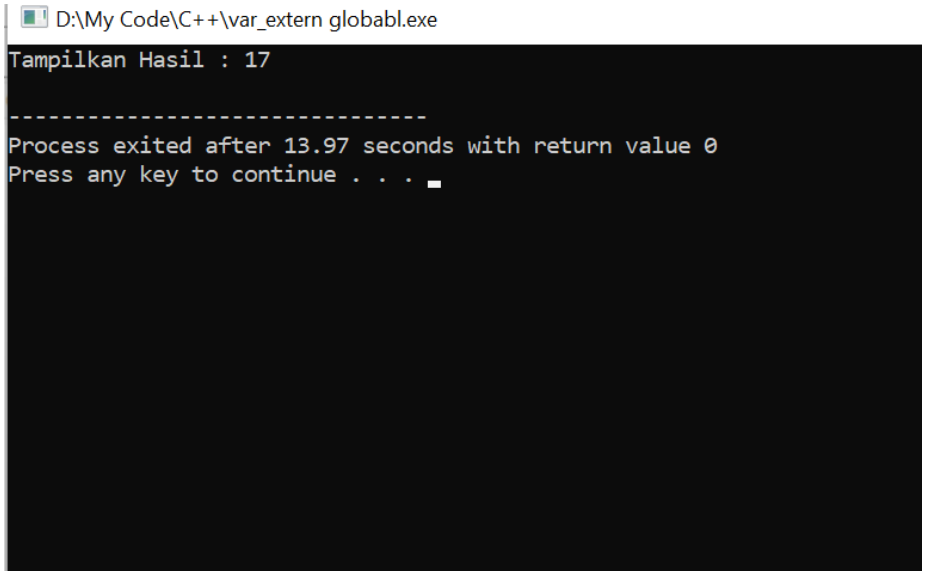



### 3.4. Variabel *Global*

Variabel *global* adalah variabel yang memiliki ruang lingkup *file*. Variabel *global* dideklarasikan di luar fungsi apapun dalam *file*. Walaupun variabel *global* diletakkan di luar fungsi, namun variabel ini dapat digunakan di dalam fungsi manapun dalam file program yang dibuat. Sebagai contohnya dapat dilihat sebagai berikut :

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

int a,b=5;    //variabel global
int jumlah(int bil1,int bil2){
      a=bil1+bil2;    //mengisi nilai var global a
      b+=a;      //mengisi nilai var global b
      return b;  //mengembalikan nilai var b ke fungsi
}
int main(){
      int x=2,y;
      x+=b;
      y=jumlah(x,b);
      cout<<"Tampilkan Hasil : "<<y<<endl;
      return 0;
}
```

Tampilan *ouput*nya akan menjadi seperti ini :



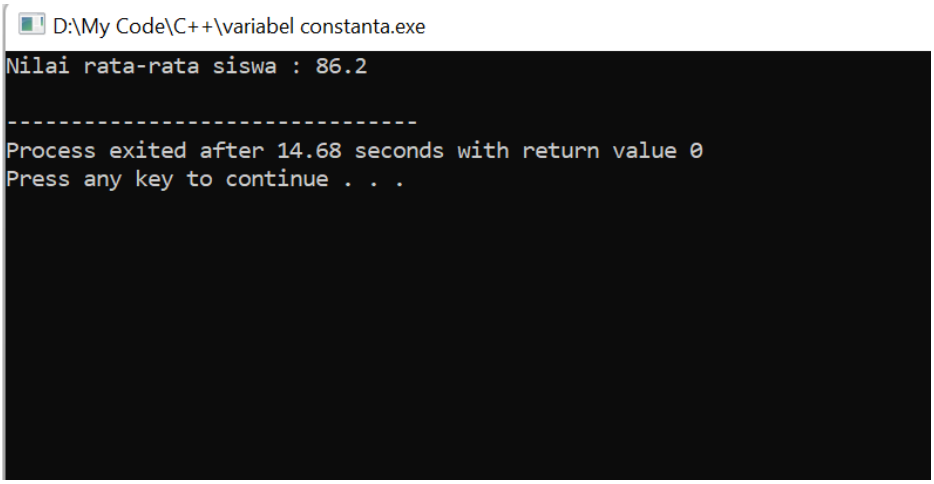

### 3.5. Variabel Konstanta

Konstanta adalah sebuah tempat atau *container* dari suatu nilai. Sesuai dengan namanya, nilai dari konstanta ber*sif*at tetap (konstan) dan tidak bisa diubah sepanjang program berjalan. Inilah yang menjadi pembeda antara konstanta dengan variabel.

Aturan penulisan variabel konstanta dapat di tulis dengan => **const** tipe_data nama_variabel = nilai_variabel;. Untuk contoh yang lebih akurat dapat dilihat pada penulisan kode program di bawah ini :

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

int main(){
	const int ttp= 5;
	int nilai_jmlh = 431;
	float rata_rata=(float)nilai_jmlh/(float)ttp;
	cout<<"Nilai rata-rata siswa : "<<rata_rata<<endl;
	return 0;
}
```

*Ouput* yang dihasilkan seperti pada gambar di bawah ini :




**Percobaan 2.2.2** Konstanta, Variabel dan pencetakan

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

int main(){
     float jari_jari, luas, keliling;
     const float phi =3.14;
     jari_jari=5.5;
     luas = phi * jari_jari * jari_jari;
     keliling = 2 * phi * jari_jari;
     cout<<"Luas Lingkaran : "<<luas<<endl;
     cout<<"Keliling Lingkaran : "<<keliling<<endl;
return 0;
}
```

### 3.6. Swapping Variable / Menukar Nilai Variabel

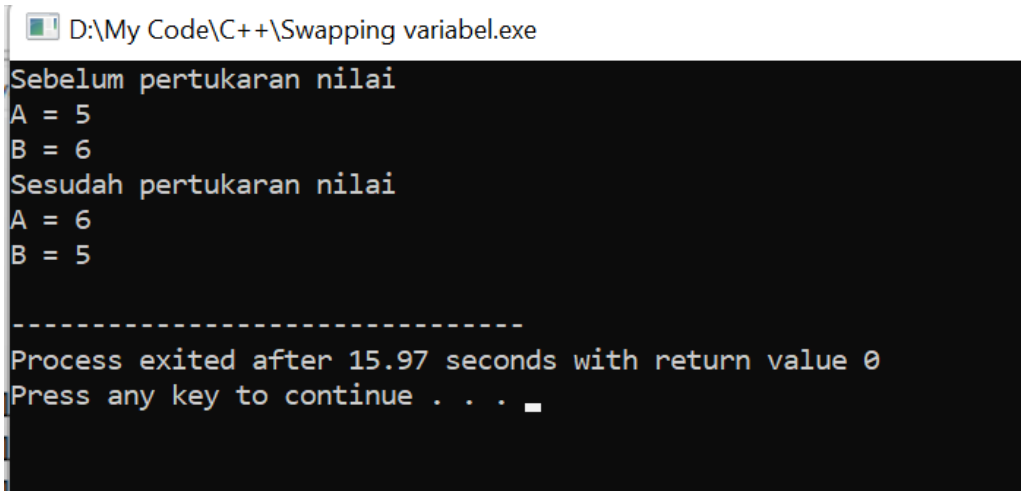
*Swapping variable* atau menukar nilai variabel adalah suatu proses dimana dua atau lebih variabel yang memiliki nilai berbeda yang dapat ditukar. Contohnya variabel a= 2 dan variabel b=4, ditukar menjadi variabel a=4 dan b=2. Logika dari program ini adalah Ketika kita mempunyai 2 gelas yang sudah terisi dengan minuman, dimana gelas A berisi

jus dan gelas B berisi susu, karena suatu hal kita akan menukarnya sehingga gelas A berisi susu dan gelas b berisi jus. Untuk menuntaskan proses pertukaran tersebut, tentunya kita akan mengambil satu buah gelas kosong sebut saja gelas c.

Sebagai contoh yang lebih jelas dapat kita lihat pada program di bawah ini :

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

int main(){
	//menukar nilai dari dua variabel
	int a=5,b=6,c;
	cout<<"Sebelum pertukaran nilai "<<endl;
	cout<<"A = "<<a<<endl;
	cout<<"B = "<<b<<endl;
	cout<<"Sesudah pertukaran nilai "<<endl;
	//mengambil var c sebagai penampungan sementara
	c=a;
	a=b;
	b=c;
	cout<<"A = "<<a<<endl;
	cout<<"B = "<<b<<endl;
	return 0;

}
```

Hasil *ouput* pada program di atas dapat dilihat pada gambar di bawah ini :



# BAB IV
# OPERATOR

## 4.1. Pengertian Operator dan Operand

Sebelum masuk ke jenis-jenis operator di dalam bahasa C++, terdapat istilah *operand* dan operator. *Operand* adalah nilai asal yang dipakai dalam sebuah proses operasi. Sedangkan Operator adalah instruksi yang diberikan untuk mendapatkan hasil dari proses tersebut.

Biasanya operator berbentuk karakter matematis atau perintah singkat sederhana. Sebagai contoh, pada operasi: 10 + 2. Angka 10 dan 2 disebut sebagai *operand*, sedangkan tanda tambah (karakter +) adalah operator.

Berdasarkan jumlah *operand*-nya, operator C++ dibagi menjadi 3 jenis : Operator *Unary***,** Operator *Binary* dan Operator *Ternary***.**

- Operator *Unary* adalah operator yang hanya terdiri dari 1 *operand*. Contohnya adalah operator posit*if* (plus): +7, +9, +10.111.
- Operator *Binary* adalah operator yang terdiri dari 2 *operand*. Sebagian besar operator di dalam bahasa C++ termasuk ke dalam operator *binary*. Contohnya seperti operator aritmatika: 4 + 8, 9 * 2, 8 % 2, dll.
- Operator *Ternary* adalah operator yang terdiri dari 3 *operand*. Bahasa C++ memiliki 1 operator *ternary*, yakni ” ? : ” seperti (a == 1) ? 20: 30.

## 4.2. Operator Aritmatika

Aritmatika adalah cabang ilmu matematika yang membahas perhitungan dasar “kabataku”, yakni operasi perkalian, pembagian, penambahan dan pengurangan. Operator aritmatika termasuk operator yang paling sering digunakan dalam dunia programming. Dalam menyelesaikan beberapa kasus pengoperasian bilangan yang kita buat, operator aritmatika sangat cocok atau pas dalam menyelesaikan kasus tersebut. Dalam Bahasa C++ ada 5 operator aritmatika yang bisa kita gunakan, dapat dilihat pada tabel berikut :

| Operator | Penjelasan | Contoh |
|---|---|---|
| + | Penambahan | A=5+5; |
| - | Pengurangan | A=5-5; |
| * | Perkalian | A=5*5; |
| / | Pembagian | A=5/5; |
| % | Modulus | A=5%5; |

Berikut adalah contoh program yang menggunakan operator aritmatika :

**Percobaan 4.1.** Operator aritmatika

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

int main(){
      int bil1,bil2;
      bil1=5;bil2=25;
      cout<<bil1<<" + "<<bil2<<" = "<<bil1+bil2<<endl;
      cout<<bil1<<" - "<<bil2<<" = "<<bil1-bil2<<endl;
      cout<<bil1<<" * "<<bil2<<" = "<<bil1*bil2<<endl;
      cout<<bil1<<" / "<<bil2<<" = "<<bil1/bil2<<endl;
      cout<<bil1<<" % "<<bil2<<" = "<<bil1%bil2<<endl;

      return 0;
}
```

## 4.3. Operator Increment dan Decrement

Operator *increment* dan *decrement* adalah sebutan untuk operasi seperti a++, dan a--. Ini sebenarnya penulisan singkat dari operasi a = a + 1 serta a = a – 1**.**

*Increment* digunakan untuk menambah variabel sebanyak 1 angka, sedangkan *decrement* digunakan untuk mengurangi variabel sebanyak 1 angka.

Penulisannya menggunakan tanda tambah 2 kali untuk *increment*, dan tanda kurang 2 kali untuk *decrement***.** Penempatan tanda tambah atau kurang ini boleh di awal seperti ++a dan --a, atau di akhir variabel seperti a++ dan a--.

Dengan demikian terdapat 4 jenis *increment* dan *decrement* dalam bahasa C++:

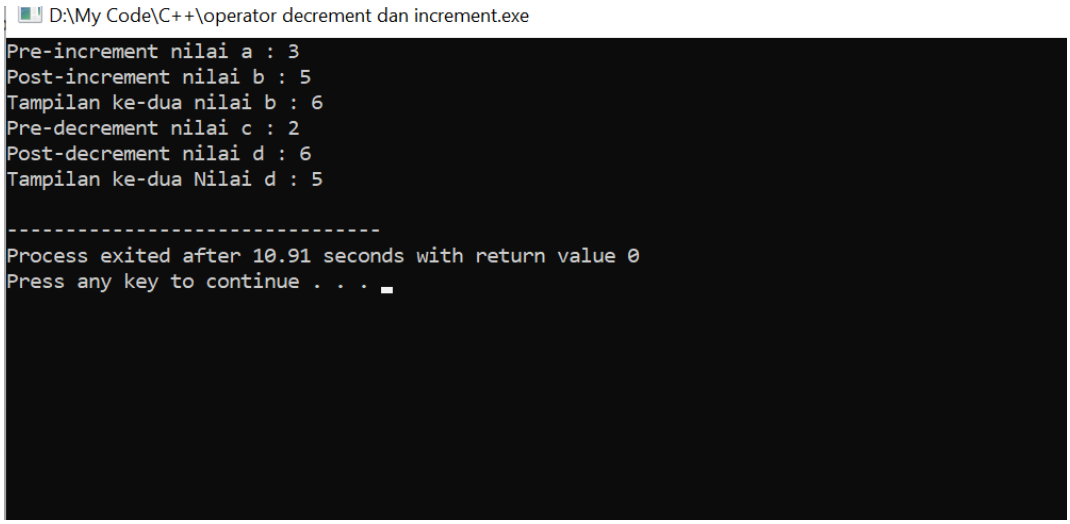
| Oprator | Contoh | Penjelasan |
|---|---|---|
| Pre-increment | ++a | Tambah a sebanyak 1 angka, lalu tampilkan hasilnya |
| Post-increment | a++ | Tampilkan nilai a, lalu tambah a sebanyak 1 angka |
| Pre-decrement | --a | Kurangi a sebanyak 1 angka, lalu tampilkan hasilnya |
| Post-decrement | a-- | Tampilkan nilai a, lalu kurangi a sebanyak 1 angka |

Berikut contoh program operator *decrement* dan *increment* :

**Percobaan 4.2.** Operator *decrement* dan *inrement*

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

int main(){
      //operator decrement dan increment
      int a=2,b=5,c=3,d=6;
      cout<<"Pre-increment nilai a : "<<++a<<endl;
      cout<<"Post-increment nilai b : "<<b++<<endl;
      cout<<"Tampilan ke-dua nilai b : "<<b<<endl;
      cout<<"Pre-decrement nilai c : "<<--c<<endl;
      cout<<"Post-decrement nilai d : "<<d--<<endl;
      cout<<"Tampilan ke-dua Nilai d : "<<d<<endl;

return 0;
}
```

Hasil *ouput* dari program di atas dapat di lihat pada gambar di bawah ini ;




## 4.4. Operator Perbandingan / Relasional

Operator perbandingan dipakai untuk membandingkan 2 buah nilai, apakah nilai tersebut sama besar, lebih kecil, lebih besar, dll. Hasil dari operator perbandingan ini adalah boolean *true* atau *false*.

Ketika ditampilkan dengan perintah cout, *true* dan *false* ini akan ditampilkan C++ sebagai *integer* 1 atau 0.

| Operator | Penjelasan | Contoh | Hasil |
|---|---|---|---|
| == | Sama dengan | 5 == 5 | 1 (true) |
| != | Tidak sama dengan | 5 != 5 | 0 (false) |
| > | Lebih besar | 6 > 4 | 1 (true) |

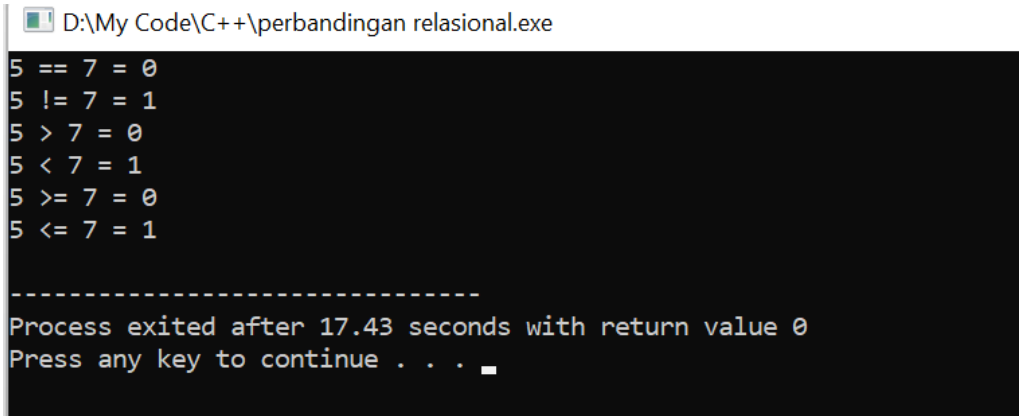
| | | | |
|---|---|---|---|
| < | Lebih kecil | 4 < 3 | 0 (false) |
| >= | Lebih besar / sama dengan | 4 >= 8 | 0 (false) |
| <= | Lebih kecil / sama dengan | 4 <= 4 | 1 (true) |

Berikut contoh program operator perbandingan atau relasional :

**Percobaan 4.3.** Membuat program operator perbandingan dengan dua bilangan

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

int a=5,b=7;
int main(){
	bool hasil;
	hasil=a==b;
	cout<<a<<" == "<<b<<" = "<<hasil<<endl;
	hasil=a!=b;
	cout<<a<<" != "<<b<<" = "<<hasil<<endl;
	hasil=a>b;
	cout<<a<<" > "<<b<<" = "<<hasil<<endl;
	hasil=a<b;
	cout<<a<<" < "<<b<<" = "<<hasil<<endl;
	hasil=a>=b;
	cout<<a<<" >= "<<b<<" = "<<hasil<<endl;
	hasil=a<=b;
	cout<<a<<" <= "<<b<<" = "<<hasil<<endl;
}
```

Hasil *Ouput*nya dapat dilihat pada gambar di bawah ini :




## 4.5. Operator Logika

Operator logika dipakai untuk menghasilkan nilai boolean *true* atau *false* dari 2 kondisi atau lebih. Jika operator logika bernilai *true* maka angka 1 yang akan tampil di program dan apabila bernilai *false* maka angka 0 yang akan keluar. Operator logika terdiri dari 3 operator. Tabel berikut merangkum hasil dari operator logika dalam bahasa C++ :

| Operator | Nama | Penjelasan | Contoh |
|---|---|---|---|
| && | And | Menghasilkan true jika kedua operand true | true && false, hasilnya: false |
| \|\| | Or | Menghasilkan true jika salah satu operand true | true \|\| false, hasilnya: false |
| ! | Not | Menghasilkan true jika operand false | !false, hasilnya: true |

Berikut contoh program operator logika :

**Percobaan 4.4.** Membuat program operator logika

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

int main(){
      bool a=true, b=false,hasil;
      hasil=a&&a;
      cout<<"Hasil a && a : "<<hasil<<endl;
      hasil=a&&b;
      cout<<"Hasil a && b  "<<hasil<<endl;
      hasil=a||b;
      cout<<"Hasil a || b : "<<hasil<<endl;
      hasil=b||b;
      cout<<"Hasil b || b : "<<hasil<<endl;
      hasil=!a;
      cout<<"Hasil !a : "<<hasil<<endl;
      hasil=!b;
      cout<<"Hasil !b : "<<hasil<<endl;
}
```

Berikut adalah tampilan *ouput* dari program di atas :

```
D:\My Code\C++\operator perbandingan.exe
Hasil a && a : 1
Hasil a && b  0
Hasil a || b : 1
Hasil b || b : 0
Hasil !a : 0
Hasil !b : 1

--------------------------------
Process exited after 15.06 seconds with return value 0
Press any key to continue . . .
```

## 4.6. Operator Penugasan / *Assignment*

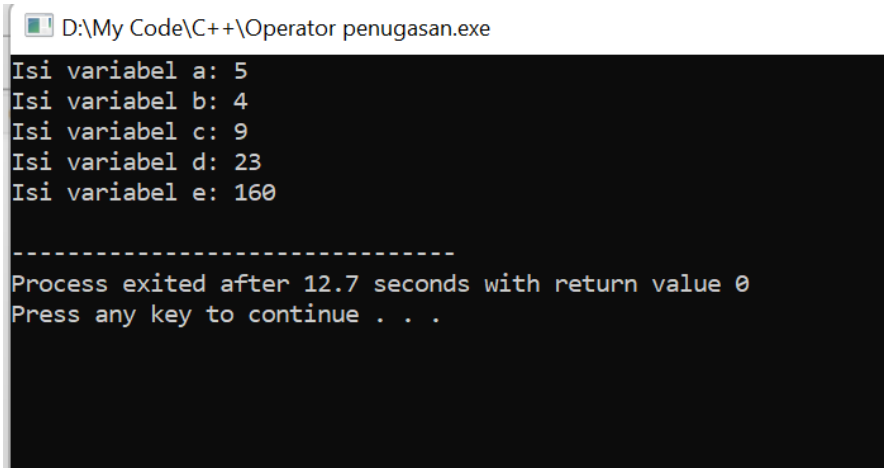
Operator *assignment* adalah operator untuk memasukkan suatu nilai ke dalam variabel. Dalam bahasa C++, operator assignment menggunakan tanda sama dengan ( = ). Pembacaan operasi assignment dilakukan dari kanan ke kiri, bukan dari kiri ke kanan seperti yang biasa kita pahami dalam matematika.

Contoh program operator penugasan adalah sebagai berikut :

**Percobaan 4.5.** Membuat program operator penugasan / *assignment*

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

int main()
{
  int a, b, c, d, e;
  a = 5;
  b = 3;
  b = b + 1;
  c = a + b;
  d = c + c + a;
  e = (c + d)* a;

  cout << "Isi variabel a: " << a << endl;
  cout << "Isi variabel b: " << b << endl;
  cout << "Isi variabel c: " << c << endl;
  cout << "Isi variabel d: " << d << endl;
  cout << "Isi variabel e: " << e << endl;

  return 0;
}
```

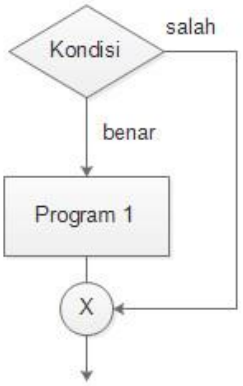
Tampilan dari *ouput* pada program di atas dapat dilihat pada gambar di bawah ini :



# BAB V
# PEMILIHAN/SELEKSI

## 5.1. *If Statements*

*If* adalah sebuah struktur pemilihan yang digunakan untuk mengeksekusi sebuah kondisi. Dalam kaidahnya sederhananya, *if* berarti sebuah kondisi "jika" sebuah kondisi bernilai benar, maka program1 yang akan mengeksekusinya. Namun jika salah, kondisi akan diabaikan.



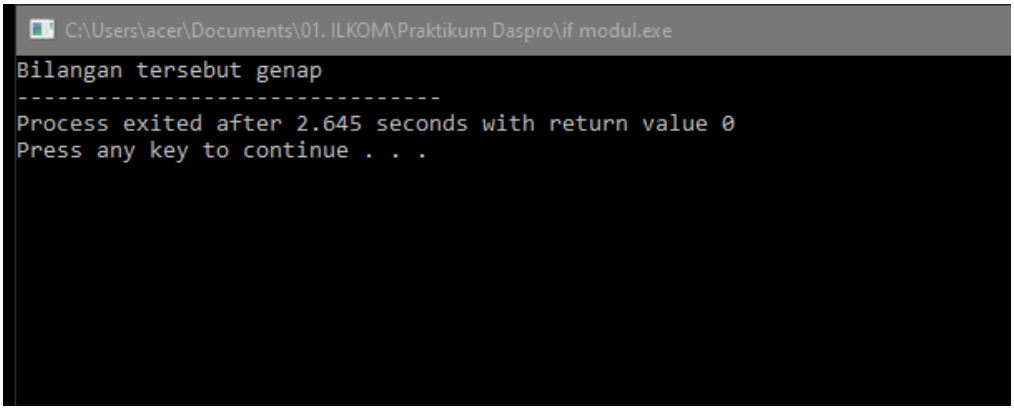

```
if (kondisi=benar){
    \\program 1 dijalankan
}
```

Contohnya program memunculkan pesan "bilangan genap" jika kondisi yang kita berikan pada inputan *keyboard* yang merupakan bilangan genap atau tidaknya. Untuk algoritmanya, kita bisa menggunakan operator sisa bagi=2 sehingga kondisinya adalah hasil dari sisa bagi tersebut. Berikut codingannya :

**Percobaan 5.1** Membuat program menentukan bilangan genap

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

int main (){
    int x = 8, s;

    s= x % 2;

    if (s==0){
        cout << "Bilangan tersebut genap";
    }
}
```

Berikut tampilan *ouput* program diatas:




## 5.2. *If Else Statements*

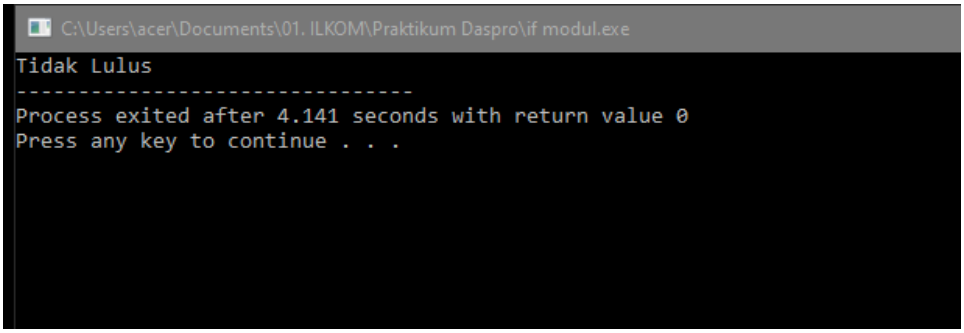
*If Else* adalah menjalankan kondisi dengan 2 pernyataan yang berbeda. Jika pada *if* sebelumnya hanya melakukan pernyataan jika kondisi benar namun akan diabaikan jika kondisi salah. Namun pada *if else*, kondisi salah tidak diabaikan tapi di berikan pernyataan juga.


Berikut contoh program yang menggunakan *if else*, mengident*if*ikasi nilai inputan suatu nilai ujian. Jika nilai ujian lebih besar atau sama dengan 60 maka akan diberikan pesan "lulus". Namun, jika tidak akan akan diberikan pesan "tidak lulus".

**Percobaan 5.2** Menentukan kelulusan

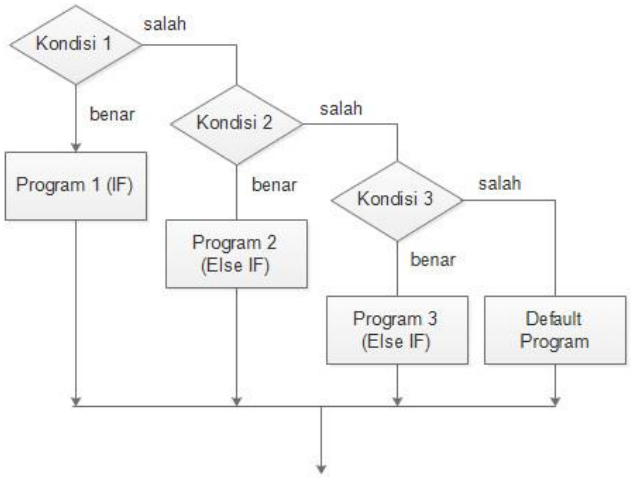
```
#include <iostream>
using namespace std;

int main (){
    int nilai = 59;


    if (nilai >= 60){
        cout << "Lulus";
    }
    else {
        cout << "Tidak Lulus";
    }
}
```

Berikut hasil dari program di atas :




## 5.3. *If Else If Statements*

*If else if* digunakan untuk melakukan pengecekkan kondisi lebih dari satu. Hal ini biasanya digunakan untuk melakukan listing kondisi lainnya jika ingin melibatkan lebih dari satu kondisi. Penggunaan *if else if* ini digunakan jika kondisi satu salah, maka akan di lemparkan ke kondisi kedua (*else if*), jika salah lagi ke kondisi ketiga (*else if*) dan seterusnya sampai ke kondisi default (*else*).



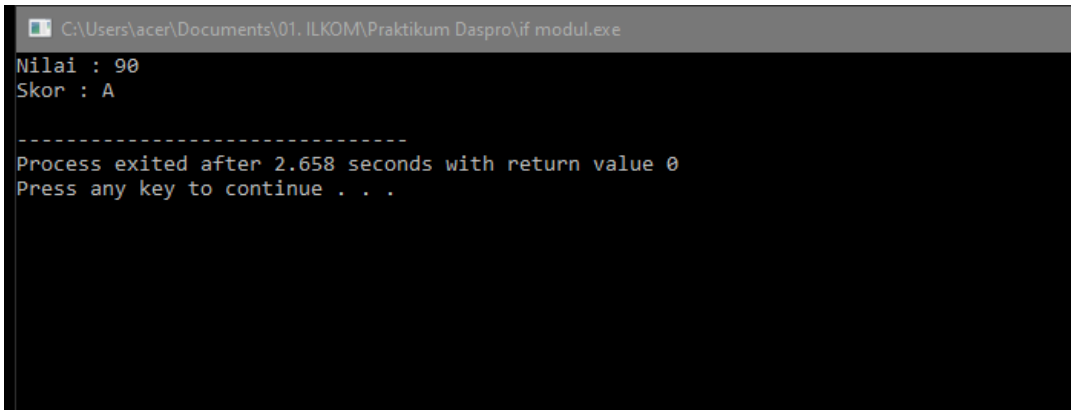
Contoh penggunaan jenis lebih dari 1 kondisi ini adalah pemberian nilai huruf di setiap nilai angka yang didapatkan. Semisal :

- jika mendapat nilai lebih dari atau sama dengan 90 maka skor A
- jika mendapat nilai lebih dari atau sama dengan 70 maka skor B
- jika mendapat nilai lebih dari atau sama dengan 60 maka skor C
- jika mendapat nilai lebih dari atau sama dengan 50 maka skor D

**Percobaan 5.3** Menentukan nilai ujian

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

int main (){
    int NilaiUjian = 90;
    char skor;


    if (NilaiUjian >= 90){
         skor='A';
    }
    else if (NilaiUjian >= 70){
        skor='B';
    }
    else if (NilaiUjian >= 60){
        skor='C';
    }
    else if (NilaiUjian >= 50){
        skor='D';
    }
    else{
        skor='E';
    }

    cout<<"Nilai : " << NilaiUjian << endl;
    cout<<"Skor : " << skor << endl;
}
```

Berikut tampilan *ouput* dari prigram diatas :



Program *if else* lebih dari satu kondisi s*if*atnya adalah sequence artinya secara listing program per kondisi (kondisi1, kondisi2, kondisi3 dst) akan membaca kondisi terlebih dahulu lalu akan langsung mengeksekusi jika sesuai dengan kondisinya. Karena hal tersebut, hal yang sama berlaku untuk program nilai diatas. Terlebih dahulu harus memposisikan kondisi nilai lebih besar atau sama dengan 90 di kondisi pertama agar sequencenya sesuai.

## 5.4. *Swith Case*

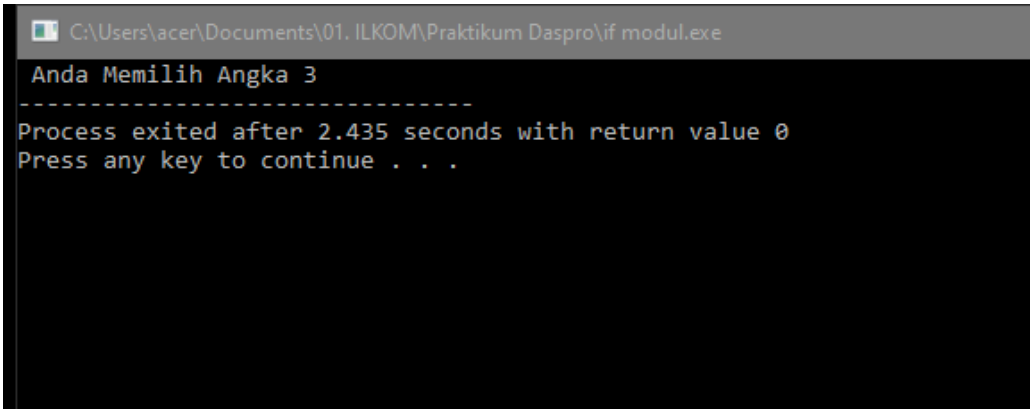
*Switch case* merupakan salah satu jenis percabangan (selain *IF ELSE*) yang dapat digunakan di bahasa **pemrograman C++.** Cara kerjanya sederhana sebuah nilai akan dibandingkan dengan setiap nilai pada ***case*** yang ada. Jika sebuah *case* mempunyai nilai yang sama (bernilai *true*) maka pernyataan pada case tersebut yang akan dijalankan. Apabila setiap *case* bernilai *false* maka pernyataan *default* yang akan dijalankan.

Berikut contoh program yang menggunakan switch case :

**Percobaan 5.4** Memilih angka

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

int main() {
   int nomer = 3 ;

      switch ( nomer ) {
         case 1 :
            cout << " Anda Memilih Angka 1 " ;
         break ;

         case 2 :
            cout << " Anda Memilih Angka 2 " ;
         break ;

         case 3 :
            cout << " Anda Memilih Angka 3 " ;
         break ;

         default :
            cout << " Pilihan default Terpilih " ;
         break ;
   }
    return 0 ;
}
```

Hasil *ouput* dari program diatas yaitu sebagai berikut:




## 5.5. *Nested if*

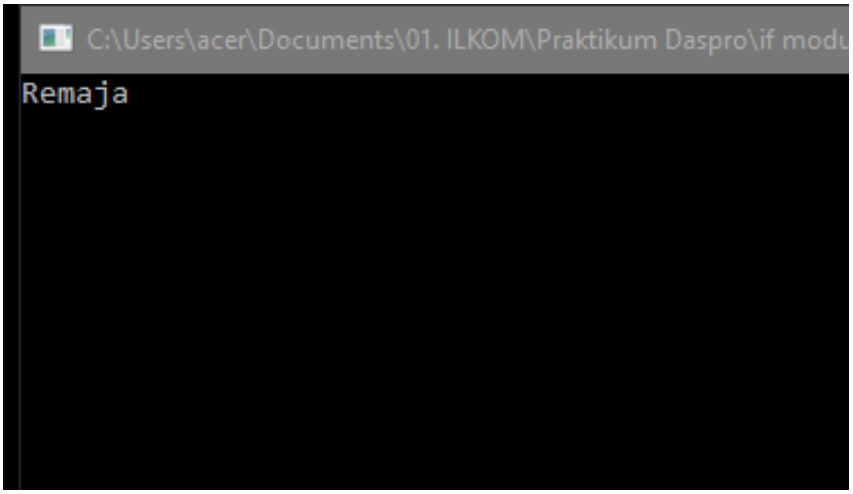
*IF* Bersarang (*Nested If*) merupakan percabangan *IF* dengan struktur yang lebih kompleks. Dimana didalam sebuah pernyataan *IF* terdapat pernyataan *IF* lainnya, Dengan kata lain terdapat sebuah kondisi *IF* didalam *IF*. Penggunaan struktur *IF* Bercabang biasa digunakan untuk pemilihan beberapa pernyataan bertingkat, Ketika sebuah pernyataan *IF* dijalankan dan bernilai *true* maka akan terdapat pernyataan *if* lainnya pada blok tersebut.

Berikut contoh program penggunaan nested *if* :

**Percobaan 5.5** Menentukan usia

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

using namespace std;
int main()
{
	int umur=14;

	if (umur<=18)
	{
		if (umur<=10){
			cout<<"Anak-anak"<<endl;
		}else {
			cout<<"Remaja"<<endl;
		}
	}else {
		cout<<"Bukan usia anak maupun remaja"<<endl;
	}

getch();
}
```

Hasil *ouput* dari program di atas sebagai berikut:



# BAB VI
# PENGULANGAN/*LOOPING*

Pengulangan digunakan untuk menjalankan satu atau beberapa pernyataan sebanyak beberapa kali. Proses yang berulang adalah suatu urut-urutan pernyataan yang akan dieksekusi terus menerus selama kondisi yang disyaratkan terpenuhi. Pengulangan proses merupakan suatu kemampuan yang dimiliki oleh semua compiler bahasa pemrograman. Terdapat banyak jenis pengulangan proses, tetapi paling tidak akan dibahas tiga buah bentuk jenis pengulangan yaitu *for*, *while* dan *do while*.

## 6.1. Pengulangan dengan Statements *For*

Penggunaan *for* dilakukan dengan cara mengulang instruksi berdasarkan ekspresi yang diberikan. Pengulangan jenis ini digunakan untuk melakukan pengulangan yang telah diketahui banyaknya. Jenis pengulangan ini lebih mudah dipahami dibanding jenis pengulangan lainnya. Tipe data yang digunakan untuk melakukan pengulangan menggunakan *for* hanya tipe data int dan char saja.

Adapun standar penulisan menggunakan *for* yaitu:

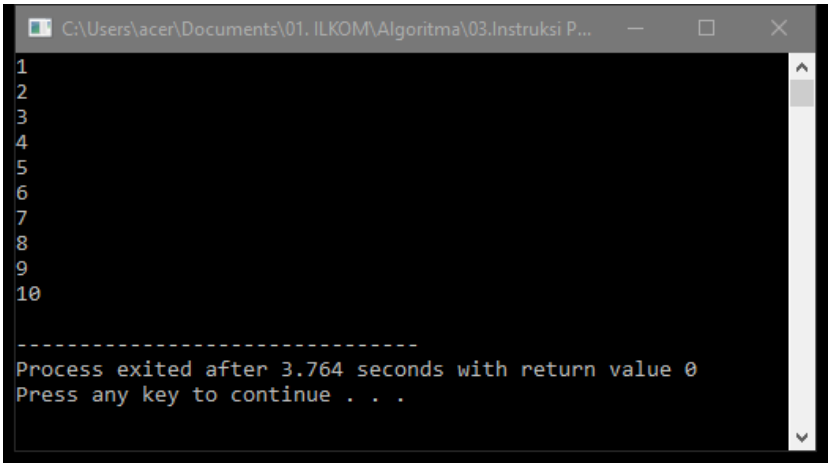
***for*** (inisialisasi nilai_awal; kondisiPembatas_perulangan; *increment*/*decrement*)
{
    instruksi-instruksi;
}

Berikut contoh program dari perulangan *for*:

**Percobaan 6.1** Mencetak angka 1 sampai 10

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

int main(){
      int i;

      for(i = 1; i <= 10; i++){
            cout<<i<<endl;
      }
      return 0;
}
```

Hasil *ouput* dari program diatas yaitu sebagai berikut :




## 6.2. Pengulangan dengan *Statements While*

Struktur pengulangan jenis *while* akan mengulang instruksi selama kondisi bernilai benar/terpenuhi dan akan berhenti ketika kondisi bernilai salah. *while* akan melakukan pengecekan kondisi diawal, apabila kondisi bernilai salah maka *statement* yang terdapat dalam blok pengulangan tidak akan di jalankan.

Bentuk penulisan dari perulangan *while* yaitu :

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

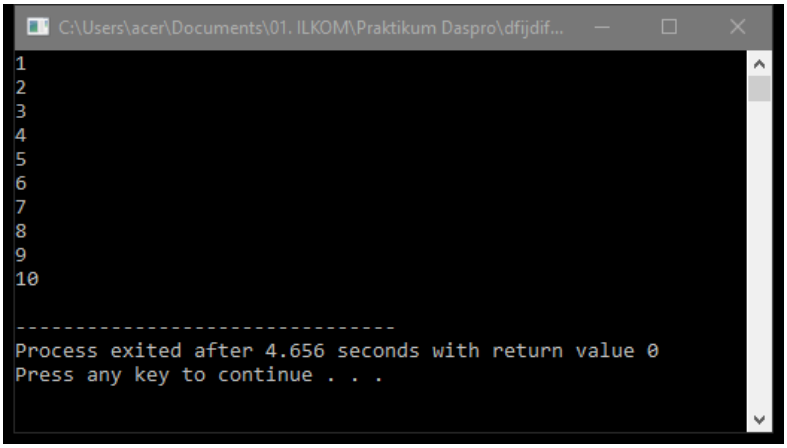
kondisi sendiri merupakan suatu ekspresi boolean, artinya hanya dapat bernilai benar (*true*) atau salah (*false*).

Contoh penggunaan *while*:

**Percobaan 6.2** Mencetak angka 1 sampai 10

```
# include < iostream >
using namespace std;
int main() {
  int i;
  i = 1; // inisialisasi nilai awal
  while (i <= 10) {
    cout << i << endl;
    i = i + 1;
  }
  return 0;
}
```

Berikut hasil *ouput* dari program di atas :




### 6.3. Pengulangan dengan Statements *Do-while*

*Do-while* adalah salah satu pernyataan pengulangan yang memungkinkan kita untuk membuat program berjalan secara fleksibel berdasarkan keinginan pengguna. *Do-while* berfungsi intuk mengulangi pengeksekusian beberapa substatement berdasarkan *conditional expression* yang ada. *Do-while* berbeda dengan pernyataan *while*. *Do-while* pertama kali akan mengeksekusi pernyataan terlebih dahulu, setelah itu baru akan memeriksa *conditional expression.*

Berikut penulisan perulangan *do-while:*

***do***{

pernyataan;

…

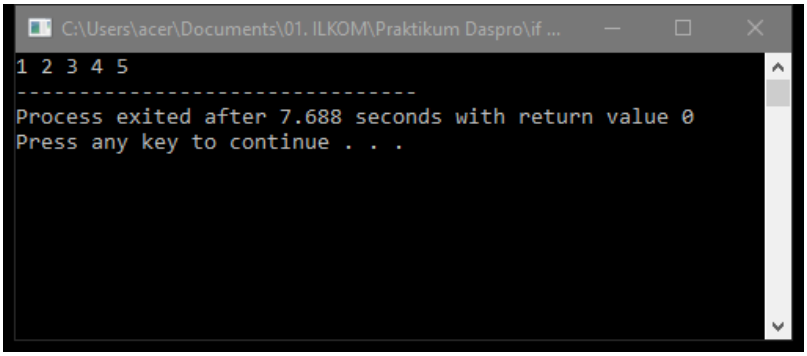
}***while***(kondisi);

Contoh program dari perulangan *Do-while:*

**Percobaan 6.3** Mencetak angka 1 sampai 5

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

int main() {
    int i = 1;

    do {
        cout << i << " ";
        ++i;
    }
    while (i <= 5);

    return 0;
}
```

Berikut hasil *ouput* program diatas :



# BAB VII
# PERULANGAN BERSARANG (*NESTED LOOP*)

**Perulangan bersarang** adalah sebutan untuk **perulangan di dalam perulangan.** Konsep seperti ini sering dipakai untuk memecahkan masalah programming yang cukup kompleks. Dalam suatu *loop* bisa terkandung *loop* yang lain, sehingga jika *loop* tersebut diulangi n kali dan *loop* yang didalamnya juga dilakukan m kali maka pernyataan yang berada didalam nested *loop* sebanyak n x n. *Loop* yang terletak dalam *loop* biasa di disebut dengan *loop* di dalam *loop* (*nested loop*).

## 7.1. *Nested Loop For*

Struktur penulisan *nested loop for* dapat dituliskan sebagai berikut :

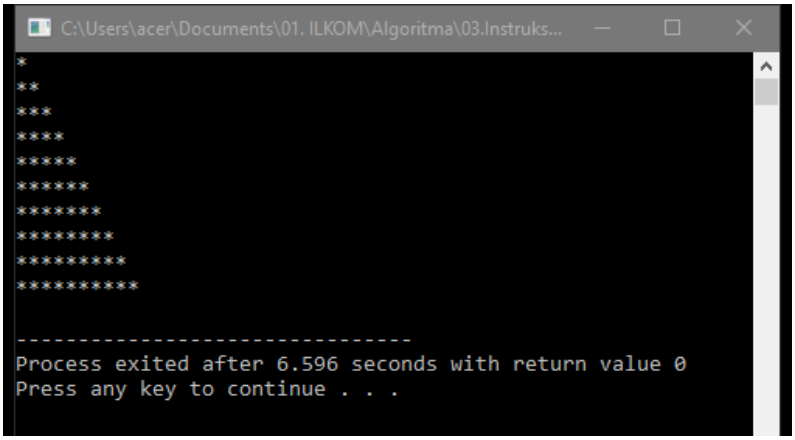
```
for (variabell=nilai_awal; kondisi; varibel1++){
    for (variabel2=nilai_awal; kondisi2; varibel2++){
            for (variabel3=nilai_awal; kondisi3; varibel3++){
            instruksi
        }
        }
}
```

Contoh program dari *nested for* :

**Percobaan 7.1** Membuat setengah piramida bintang

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

int main(){
      int n = 10;

      for(int i = 1; i <= n; i++){
            for (int j = 1; j<=i; j++){
                  cout<<"*";
            }
            cout<<endl;
      }
      return 0;
}
```

Hasil *ouput* dari program di atas :




### 7.2. *Nested Loop While*

Struktur penulisan *nested while* dapat ditulis seperti berikut :

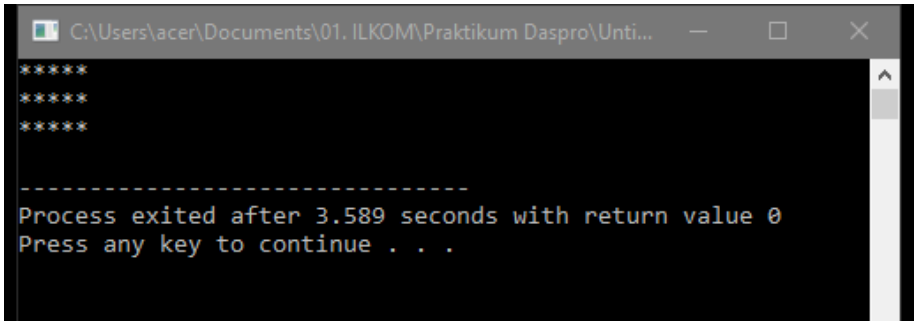
```
while(kondisi){
  while(kondisi){
      Pernyataan;
  }
  Pernyataan; //bisa ditambah pernyataan
}
```

Contoh program dari *nested while*:

**Percobaan 7.2** Membuat 3 baris 5 bintang

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

int main(){
      int i=1;
      while(i<=3) {
            int j=1;
            while(j<=5) {
                  cout<<"*";
                  j++;
            }
            cout<<endl;
            i++;
      }
}
```

Berikut hasil *ouput* dari program diatas :



# BAB VIII
# FUNGSI

## 8.1. Fungsi Di C++

Fungsi merupakan suatu bagian dari program c++ yang di maksudkan untuk mengerjakan suatu tugas tertentu dan letaknya terpisah dari program yang memanggilnya. Fungsi merupakan elemen utama dalam c++ karena bahasa c++ sendiri terbentuk dari kumpulan fungsi-fungsi. Dalam setiap program, setidaknya terdapat satu fungsi utama yang merupakan fungsi awal pemanggilan program yaitu fungsi main().

Sebelum di gunakan atau di panggil, suatu fungsi harus di deklarasikan dan di definisikan terlebih dahulu, bentuk umum pendeklarasiannya kurang lebih seperti ini :

TipeFungsi NamaFungsi (ParameterFungsi)

Tipe fungsi di deklarasikan apakah tipe int, string, float atau lainnya. Nama fungsi digunakan sebagai penamaan saja, merupakan nama suatu variabel. Parameter fungsi berisi kondisi dan status dari fungsi yang digunakan. Untuk pendefinisian fungsi kurang lebih seperti berikut ini :

```
TipeFungsi NamaFungsi (ParamenterFungsi){
    statement
    statement
    .....
}
```

Hal-hal yang perlu di perhatikan dalam penggunaan fungsi :

- Kalau tipe fungsi tidak di sebutkan, maka secara default fungsi akan di anggap bertipe integer.
- Untuk fungsi yang memiliki keluaran bertipe bukan integer, maka di perlukan pendefinisian penentu tipe fungsi.
- Untuk fungsi yang tidak memiliki nilai keluaran maka di masukan ke dalam tipe void.
- Pernyataan yang di berikan untuk memberikan nilai akhir fungsi berupa pernyataan return
- Suatu fungsi dapat menghasilkan nilai balik bagi fungsi pemanggilnya.

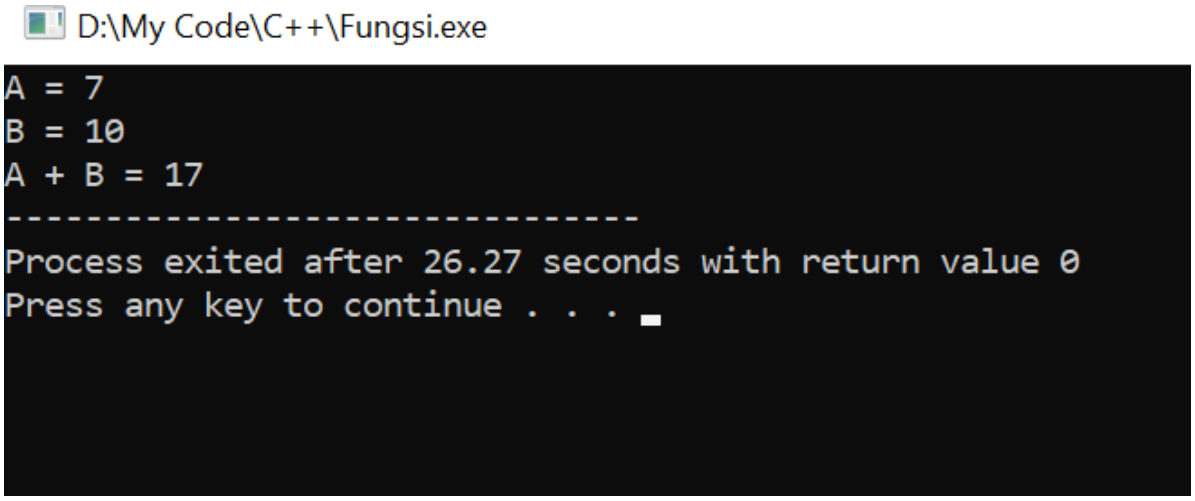
Contohnya kita akan membuat suatu fungsi di luar fungsi main. Contoh disini kita membuat fungsi penjumlahan bilangan yang di input. Proses instruksi penjumlahan di letakkan di fungsi tambah artinya di luar fungsi main. Untuk membuat

fungsi tambah terlebih dahulu kita deklarasikan tipe fungsi dan parameternya. Berikut program penjumlahan menggunakan fungsi pengembalian :

**Percobaan 8.1.** Membuat program penjumlahan dengan menggunakan fungsi

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

float tambah(float x, float y); /* deklarasi fungsi tambah */
int main(){

    float a, b, c;
    cout << "A = ";
    cin >> a;
    cout << "B = ";
    cin >> b;

    c = tambah(a, b); /* pemanggilan fungsi tambah() */
    cout << "A + B = " << c;
    getch();
}
```

Berikut adalah tampilan *ouput* dari program di atas :

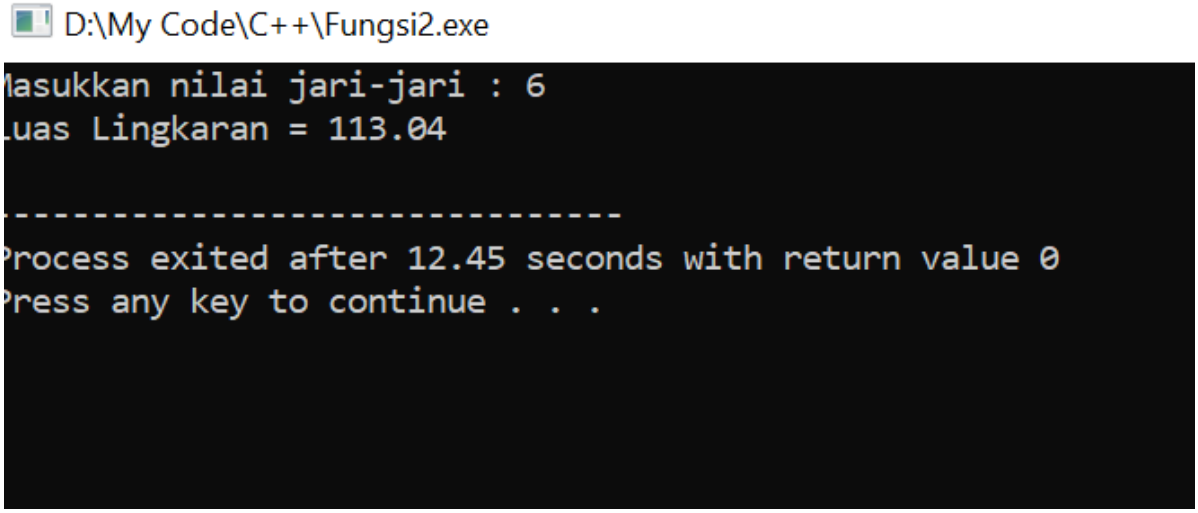



Contoh lain dari penggunaan fungsi, misalnya dengan menghitung luas suatu lingkaran. Gambaran programnya dapat di lihat pada contoh di bawah ini :

**Percobaan 8.2.** Membuat program menghitung luas lingkaran dengan menggunakan fungsi

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

const float phi=3.14;
float lingkaran(float r){
      float luas;
      luas=phi*r*r;
      return luas;
}
int main(){
      float jari_jari;
      cout<<"Masukkan nilai jari-jari : ";
      cin>>jari_jari;
      cout<<"Luas Lingkaran = "<<lingkaran(jari_jari)<<endl;

      return 0;
}
```

Tampilan *ouput* pada program di atas dapat dilihat pada gambar di bawah ini :




## 8.2. Fungsi Rekursif

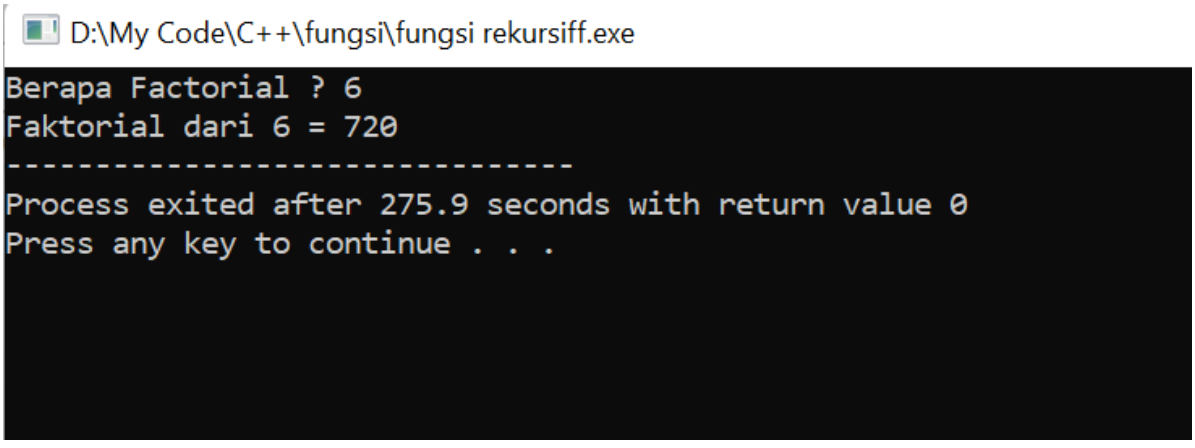
Fungsi rekursif adalah fungsi yang memanggil dirinya sendiri. Kita bisa menyebutnya loop namun tetap sebagai suatu fungsi. Perbedaannya dengan fungsi biasa adalah bahwa rekursif bisa memanggil dirinya sendiri, tetapi fungsi biasa harus di panggil lewat pemanggil prosedur atau fungsi. Fungsi rekursif jarang di temui tergantung kondisi dan pemakaian yang di perlukan untuk menjalankan instruksi sesuai fungsi tersebut.

Contoh dari program rekursif dengan menghitung nilai factorial dari suatu bilangan adalah sebagai berikut :

**Percobaan 8.3** Menghitung nilai faktorial dari suatu bilangan menggunakan fungsi rekursif

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

long int faktorial(int N); /* deklarasi fungsi faktorial */
int main(){
    int N;
    cout << "Berapa Factorial ? ";
    cin >> N;
    cout << "Faktorial dari " << N << " = " << faktorial(N);
    getch();
}

long int faktorial(int N) { /* eksekusi fungsi faktorial */
    if(N==0){
            return(1);
    }
    else{
            return(N * faktorial(N - 1)); /*fungsi faktorial()
memanggil fungsi factorial()*/
    }
}
```

Tampilan *ouput* dari program di atas dapat dilihat pada gambar di bawah ini :




## 8.3. Fungsi Prosedur / Fungsi Void

Prosedur adalah jenis fungsi yang berbeda lainnya dalam hal ini void. Prosedur dan fungsi biasa adalah 2 jenis instruksi yang berbeda. Dalam prosedur tidak ada pengembalian nilai seperti fungsi biasa. Namun prosedur dan fungsi sama sama terpisah dari program utama. Cara penulisan prosedur :

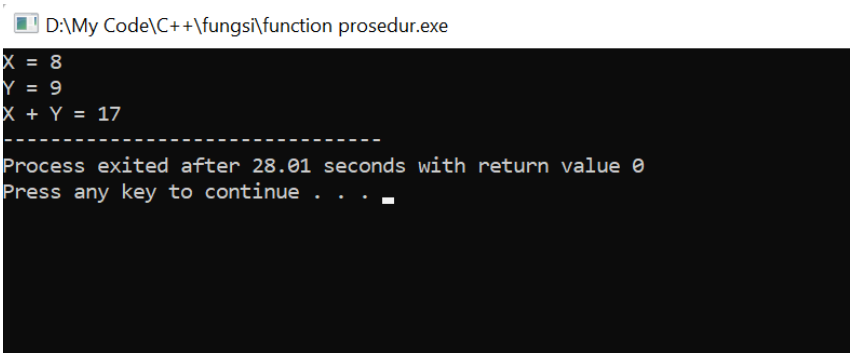
- Diberikan suatu nama yang berfungsi untuk penamaan saja. Merupakan nama suatu variabel.
- Parameter formal (jika ada), yaitu diberikan parameternya.

Berikut contoh fungsi prosedur(*void*) tanpa parameter.

**Percobaan 8.4** Membuat program fungsi prosedur tanpa parameter.

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

int a,b;

void tambah(){
    int p;
    p = a + b;
    cout << "X + Y = " << p;
}

int main(){
    cout << "X = ";
    cin >> a;
    cout << "Y = ";
    cin >> b;
    tambah();
}
```

Berikut hasil tampilan *ouput* dari program di atas :

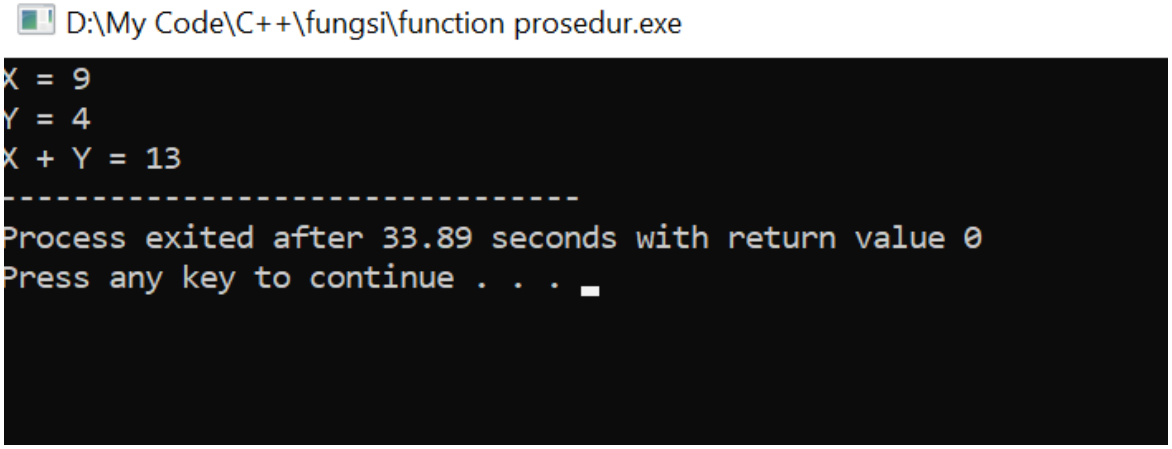


Berikut contoh prosedur (void) dengan parameter. Yaitu parameter untuk mengubah bentuk variabel x, y menjadi variabel a dan b.

**Percobaan 8.5** Membuat program fungsi prosedur dengan parameter.

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

int x,y;

void tambah(int a, int b){
    int p;
    p = a + b;
    cout << "X + Y = " << p;
}

int main(){
    cout << "X = ";
    cin >> x;
    cout << "Y = ";
    cin >> y;
    tambah(x, y);
}
```

Tampilan *ouput* pada program di atas dapat dilihat pada gambar di bawah ini :



# BAB IX
# ARRAY

## 9.1. Pengertian Array

Array merupakan kumpulan elemen yang bertipe sama dalam urutan tertentu yang menggunakan nama yang sama. Letak atau posisi dari elemen array ditunjukkan oleh index atau posisi. Dalam beberapa buku, array sering juga disebut dengan istilah Larik atau Tabel. Array termasuk dalam struktur data statis, artinya adalah lokasi memori untuk suatu array tidak dapat ditambah atau dikurangi selama program dijalankan. Maka untuk mengubah ukuran dari lokasi memori suatu array harus diperbaiki dalam listing programnya.

Array dapat berupa array 1 dimensi, 2 dimensi, bahkan n-dimensi. Suatu array dikatakan sebagai array satu dimensi, dua dimensi, atau n-dimensi berdasarkan banyaknya penuntuk indeks/posisi.

## 9.2. Array Satu Dimensi

Dikatakan array satu dimensi karena banyaknya penunjuk indeks hanya satu. Sebelum variabel array digunakan maka variabel array harus dideklarasikan terlebih dahulu. Pendeklarasian variabel array satu dimensi sebenarnya hampir sama dengan pendeklarasian variabel yang lain, hanya saja pendeklarasian variabel array diikuti dengan maksimum banyaknya elemen yang dapat disimpan dalam variabel array yang dituliskan dalam pasangan tanda siku pembuka dan siku penutup. Di dalam bahasa C++, harga awal indeks dimulai dari 0 (nol). Maka jika dituliskan banyaknya maksimum elemen adalah N, berarti indeks yang akan digunakan adalah 0, 1, 2, …, N-1.

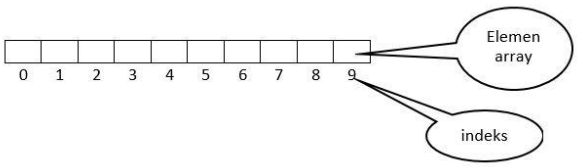
Bentuk umum pendeklarasian array :

Tipe_Data Nama_Var_Array[ukuran]

Sebagai contoh penulisannya dapat dilihat berikut ini :

int nilai[10]

dengan deklarasi di atas maka akan disiapkan lokasi memori untuk variabel array Nilai sebanyak 10, seperti di bawah ini.




Menginisialisasi array sama dengan memberikan nilai awal array pada saat dideklarasikan. Bentuk umum inisialisasi array satu dimensi:

```
tipe_data nama_var_array[ukuran]={elemen-0, elemen-1, ..., elemen-n-1}
```

Dimana elemen-0, elemen-1, …, elemen-n-1 merupakan elemen-elemen dari array sebanyak ukuran. Jika banyaknya elemen kurang dari ukuran, maka sisanya akan diberi harga 0 (nol). Tetapi jika banyaknya elemen lebih dari ukuran makan akan mengalami kesalahan "*Too many initializers*".

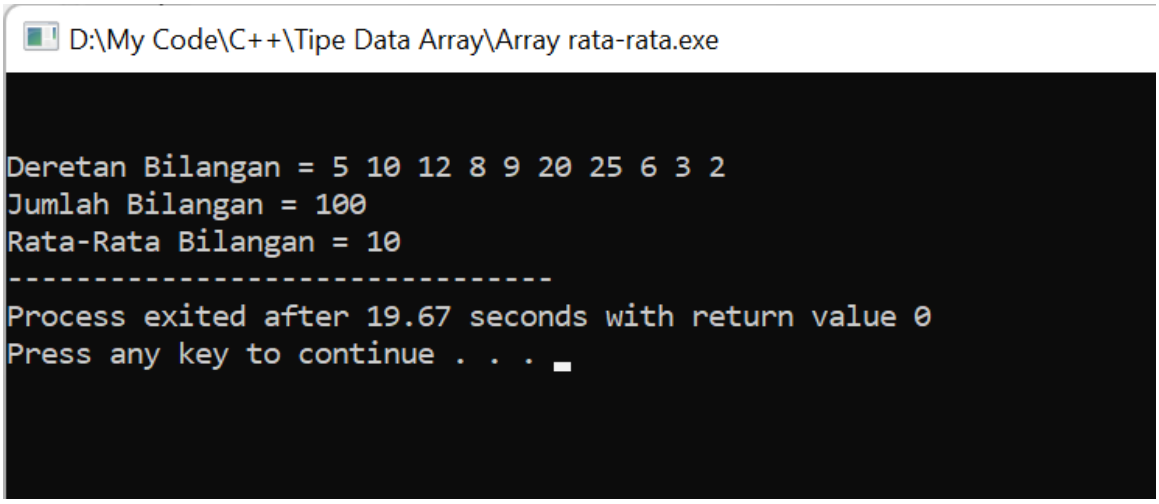
int nilai[10]={5,10,12,8,9,20,25,6,3,2};

Contoh di atas berarti kita memesan tempat di memori komputer sebanyak 10 dengan indeks dari 0 sampai 9, dimana elemen-elemen akan dimasukkan ke lokasi nilai secara berturut-turut mulai dari indeks ke-0 hingga indeks ke-9, seperti berikut ini.

**Percobaan 9.1** Membuat program menghitung nilai rata-rata menggunakan array

```
#include <iostream>
#include <conio.h>
using namespace std;
main()
{
    int Nilai[10]={5,10,12,8,9,20,25,6,3,2};
    int i, Jumlah=0;
    float Rata_Rata;
    // Menghitung jumlah
    for (i=0;i<10;i++)
        Jumlah+=Nilai[i];
        Rata_Rata=(float) Jumlah / 10;
    //Mencetak Elemen Array
    cout<<"\n\nDeretan Bilangan = ";
    for (i=0;i<10;i++)
        cout<<Nilai[i]<<" ";
    //Mencetak Harga Jumlah
    cout<<"\nJumlah Bilangan = "<<Jumlah;
    cout<<"\nRata-Rata Bilangan = "<<Rata_Rata;
    getch();
}
```

Berikut adalah tampilan *ouput* dari program di atas :




### 9.3. Mengakses Elemen Array Satu Dimensi

Elemen-elemen suatu array satu dimensi dapat diakses dengan menyebutkan nama variabel array yang diikuti dengan indeks/posisi elemen yang diakses. Pengaksesan elemen array dapat dilakukan berurutan atau random berdasarkan indeks tertentu secara langsung.

Nama_var_array[indeks];

Pengaksesan nilai pada indeks tertentu dapat dilakukan dengan mengeset nilai atau menampilkan nilai pada indeks yang dimaksud.

nilai[3]; → berarti mengakses/mengambil elemen dari variabel array yang nilainya ada pada posisi ke-4. nilai[6]; → berarti mengakses/mengambil elemen dari variabel array yang nilainya ada pada posisi ke-7.

Sebagai contoh membuat program menyeleksi data bilangan ganjil yang ada pada var array dan menentukan berapa data bilangan ganjil yang tersedia, jika tidak ada angka bilangan ganjil, maka akan tampil pernyataan tersebut.

**Percobaan 9.2.** Membuat program yang menentukan data bilangan ganjil

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

int main(){
	int bil[]={3,6,7,4,8,9,10,12,14,19};
	int batas=sizeof(bil)/sizeof(*bil);
	int ganjil[batas],g=0;
	cout<<"Deret bilangan bulat : { ";
	for(int i=0;i<batas;i++){
		cout<<bil[i]<<"  ";
	}
	cout<<"}"<<endl;
	/*mengeksekusi bilangan ganjil dari deret bilangan
tersebut*/
	cout<<"Deret bilangan ganjil : { ";
	for(int a=0;a<batas;a++){
		if(bil[a]%2!=0){
			cout<<bil[a]<<"  ";
			ganjil[a]=bil[a];
			g+=1;
		}
	}
	cout<<"}"<<endl;
	if(g<1){
		cout<<"Tidak ada bilangan ganjil yang
terinput\n";
	}
	else{
		cout<<"Total bilangan ganjil adalah :
"<<g<<endl;
	}
	getch();
}
```

Berikut adalah hasil *ouput* dari program di atas :

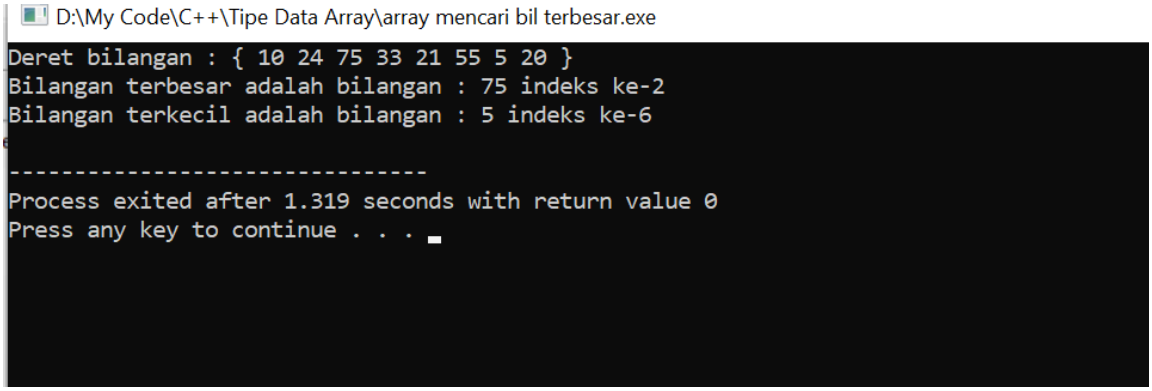
```
D:\My Code\C++\Tipe Data Array\Array2.exe
Deret bilangan bulat : { 3  6  7  4  8  9  10  12  14  19  }
Deret bilangan ganjil : { 3  7  9  19  }
Total bilangan ganjil adalah : 4

--------------------------------
Process exited after 53.65 seconds with return value 0
Press any key to continue . . .
```

Program selanjutnya adalah dengan membuat sekumpulan data bilangan dan mengambil bilangan terbesar dan bilangan terkecil dari data tersebut :

**Percobaan 9.3** Membuat program menentukan bilangan terkecil dan terbesar.

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

int main(){
      int data[]={10,24,75,33,21,55,5,20};
      int jmlh=sizeof(data)/sizeof(*data);
      int u=data[0];
      int a=data[0];
      int b,k;
      cout<<"Deret bilangan : { ";
      for(int i=0;i<jmlh;i++){
            cout<<data[i]<<" ";
            if(data[i]>=u){
                  u=data[i];
                  b=i;
            }
            else if(data[i]<=a){
                  a=data[i];
                  k=i;
            }
      }
      cout<<"}"<<endl;
      cout<<"Bilangan terbesar adalah bilangan : "<<u<<" indeks
ke-"<<b<<endl;
      cout<<"Bilangan terkecil adalah bilangan : "<<a<<" indeks
ke-"<<k<<endl;
      getch();
}
```

Tampilan *ouput* dari program di atas dapat dilihat pada gambar di bawah ini :



## 9.4. Array Dua Dimensi

Array dua dimensi sering digambarkan sebagai sebuah matriks Array berdimensi dua memberikan kita kesempatan untuk menyimpan data baik dalam bentuk baris maupun dalam bentuk kolom. Oleh karena itu dibutuhkan dua buah nilai indeks.

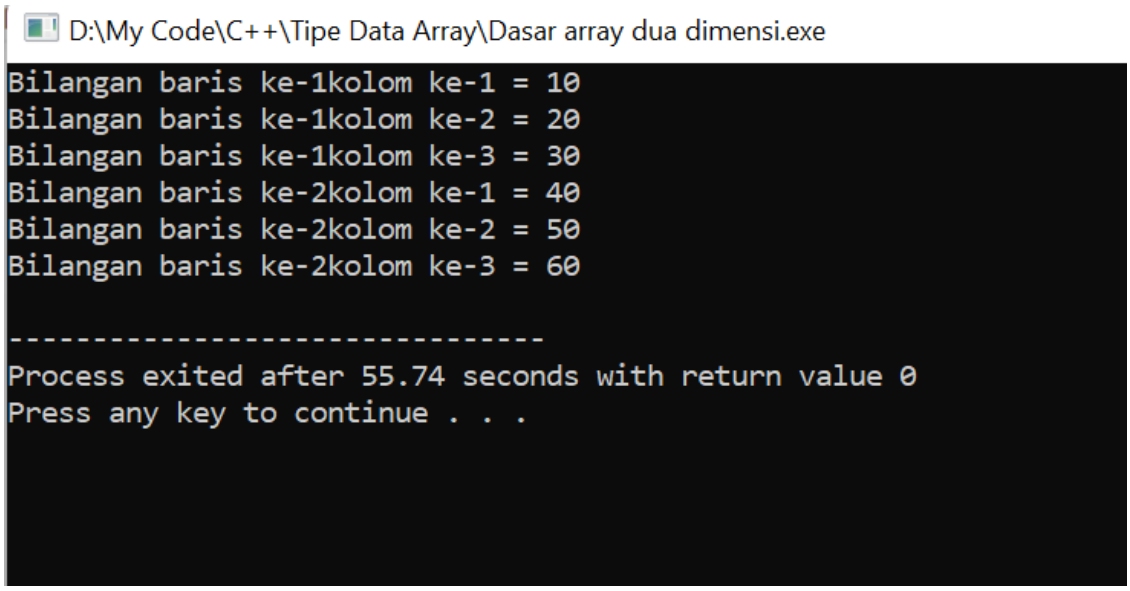
Bentuk umum dalam mendeklarasikan array berdimensi dua adalah sebagai berikut :

Tipe_data Var_array[jumlah_baris][jumlah_kolom];

Bentuk penulisan array dua dimensi adalah dengan mengident*if*ikasikan terlebih dahulu berapa jumlah baris yang diinginkan dan jumlah kolom yang ingin digunakan. Array dua dimensi lebih dulu membaca data baris yang dimasukkan kemudian membaca kolom yang dimasukkan. Syntax sederhana array dua dimensi adalah sebagai berikut :

```
#include<iostream>
#include<conio.h>
using namespace std;
int main(){
   int data_bil[2][3]={10,20,30,
                            40,50,60};
   for(int a=0;a<2;a++){
            for(int i=0;i<3;i++){
                 cout<<”Bilangan  baris  ke-“<<a+1<<”kolom  ke-
   “<<i+1<<” = “<<data_bil[a][i]<<endl;
            }
   }
   getch();
}
```

Tampilan *ouput*nya adalah sebagai berikut :

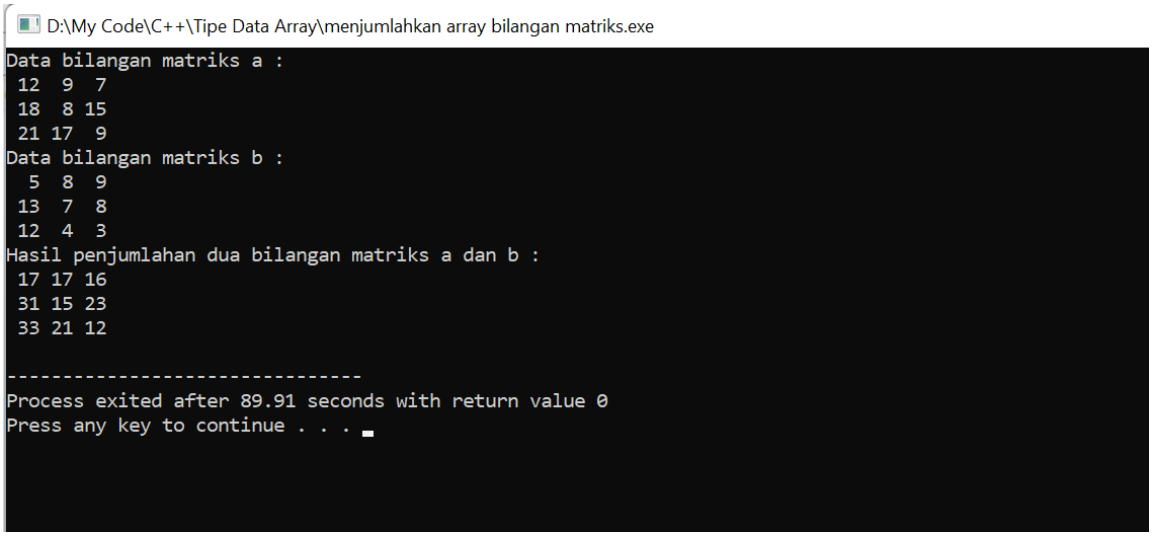



Pada program di atas sudah menjelaskan tentang bagaimana cara menginisialisasikan suatu variabel yang memiliki banyak data di dalamnya / *array*. Sebagai contoh selanjutnya adalah menjumlahkan dua data *array* yang berisikan beberapa angka/bilangan atau bisa dibilang mengoperasikan bilangan matriks. Untuk menjumlahkan kedua matriks, dibutuhkan ordo yang sama. Ordo adalah ukuran baris dan kolom. Kemudian, proses yang terjadi adalah setiap elemen pada matriks pertama dijumlahkan dengan elemen pada matriks kedua sesuai dengan alamat baris dan kolomnya.

**Percobaan 9.5** Menjumlahkan dua bilangan matriks

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

int main(){
	int matriks_a[3][3]={12,9,7,
	                     18,8,15,
	                     21,17,9};
	int matriks_b[3][3]={5,8,9,
	                     13,7,8,
	                     12,4,3},hasil[3][3];
	//menampilkan data matriks a
	cout<<"Data bilangan matriks a : \n";
	for(int a=0;a<3;a++){
		for(int b=0;b<3;b++){
			cout<<setw(3)<<matriks_a[a][b]<<setw(3);
		}
		cout<<endl;
	}
	cout<<"Data bilangan matriks b : \n";
	for(int i=0;i<3;i++){
		for(int n=0;n<3;n++){
			cout<<setw(3)<<matriks_b[i][n]<<setw(3);
		}
		cout<<endl;
	}
	//menjumlahkan kedua bilangan matriks a dan b
	cout<<"Hasil penjumlahan dua bilangan matriks a dan b : \n";
	for (int g=0;g<3;g++){
		for(int h=0;h<3;h++){
			hasil[g][h]=matriks_a[g][h]+matriks_b[g][h];
			cout<<setw(3)<<hasil[g][h]<<setw(3);
		}
		cout<<endl;
	}
	getch();
}
```

Hasil *ouput* pada program di atas, dapat dilihat pada gambar di bawah ini :



# BAB X
# RECORD/STRUCT

## 10.1. Record di C++

*Record* adalah suatu tipe data yang merupakan gabungan dari 2 (dua) atau lebih *field*/variabel. Dalam tabel database sebuah *record* akan mewakili sebuah baris (*row*) sedangkan *field*/variabel adalah merupakan kolom (*column*).

*Record* atau *Struct* dalam bahasa C++ banyak digunakan untuk menyimpan data dalam jenis yang banyak. Dalam pemrograman, *record* digunakan untuk membuat sebuah tipe data baru yang kita inginkan, dalam hal ini. *Record* mempunyai dua struktur, yaitu *Record* yang mewakili "baris" dan *field* yang mewakili "kolom".

Untuk memahami tipe data *record* perhatikan contoh tabel data mahasiswa dibawah ini:

| NIM | NAMA | USIA | JML_SAUDARA |
|---|---|---|---|
| 21081 | FERDI | 20 | 4 |
| 21035 | AGUNG | 21 | 5 |
| 21015 | AKBAR | 20 | 2 |
| 21001 | ILHAM | 21 | 3 |

1. Dalam 1 kolom, tipe data yang diisikan pasti sama (missal NIM dideklarasikan sebagai data numeric (integer misalnya) maka semua NIM harus berupa data angka).
2. Suatu obyek dapat dikenali secara tunggal menggunakan gabungan nilai data kolom – kolom dalam setiap barisnya. (misal : gabungan nilai NIM ‘21081’, NAMA ‘FERDI’, USIA ‘20’ dan JML_SAUDARA ‘4’ mengacu pada suatu obyek yang tertentu yaitu seseorang).

Di dalam konsep database, kolom dalam suatu tabel seperti di atas di sebut sebagai atribut atau *field*. Sedangkan gabungan *field – field* dalam suatu baris di sebut *tuple* atau *record*.

Dengan deskripsi di atas, dapat di katakan bahwa seorang mahasiswa dapat dinyatakan sebagai suatu *record* yang memiliki 4 data (elemen) yaitu field NIM,NAMA,USIA, dan JML_SAUDARA.

Bentuk umum :

```
struct namastruct
{
   <tipe data> field1;
   <tipe data> field2;
   <tipe data> field3;
};
```

Contoh deklarasi penggunaan :

```
 struct mahasiswa
{
    char nim[10];
    char nama[50];
    char alamat[100];
    float ipk;
};
```

untuk pemanggilan *field* pada struktur dengan menambahkan simbol titik (.) misal ingin menampilkan nim mahasiswa di layar :

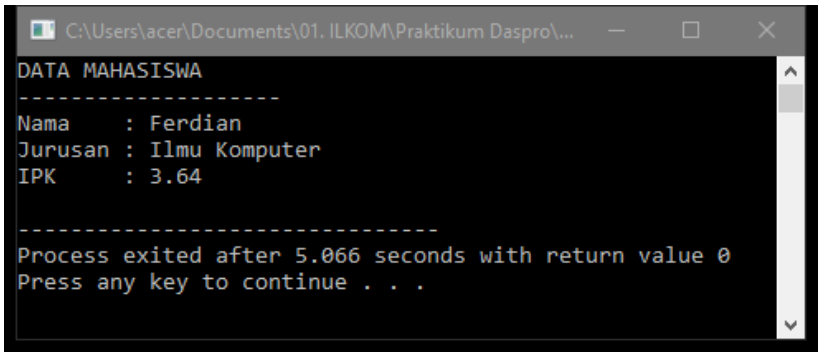
```
cout<<mahasiswa.nim;
```

Contoh program struct data mahasiswa :

**Percobaan 10.1** Menampilkan data mahasiwa

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

struct mahasiswa{
    string nama;
    string jurusan;
    float ipk;
};

int main(){

    mahasiswa mhs;

    mhs.nama="Ferdian";
    mhs.jurusan="Ilmu Komputer";
    mhs.ipk=3.64;
    cout<<"DATA MAHASISWA"<<endl;
    cout<<"--------------------"<<endl;
    cout<<"Nama    : "<<mhs.nama<<endl;
    cout<<"Jurusan : "<<mhs.jurusan<<endl;
    cout<<"IPK     : "<<mhs.ipk<<endl;

    return 0;
}
```

Hasil *ouput* programnya adalah:




**Percobaan 10.2** Konversi Waktu

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

struct waktu {
 int jam;
 int menit;
 int detik;
};
void getWaktu(waktu& wkt) {
 cout<<"Masukkan jam = ";cin >> wkt.jam;
 cout<<"Masukkan menit = ";cin >> wkt.menit;
 cout<<"Masukkan detik = ";cin >> wkt.detik;
}
void printWaktu( waktu wkt) {
 cout << wkt.jam << ":"<< wkt.menit << ":" << wkt.detik ;
}
int cariJumlahDetik(waktu wkt){
 return (wkt.jam * 3600 + wkt.menit*60 + wkt.detik);
}
void konversiWaktu(int jDetik, waktu& wkt){
int sisa;
wkt.jam = jDetik / 3600;
sisa = jDetik % 3600;
wkt.menit = sisa / 60;
wkt.detik = sisa % 60;
}
main() {
 waktu time1;
 int jDetik;
 cout << "Input Waktu " << endl; getWaktu(time1);
 cout << "Cetak Waktu " << endl; printWaktu(time1);
 jDetik = cariJumlahDetik(time1);
 cout << "\nJumlah detik adalah : " << jDetik << endl;

 konversiWaktu(jDetik, time1);
 cout << "Cetak Waktu " << endl; printWaktu(time1);
}
```

Hasil *output* program diatas dapat dilihat di bawah ini :