ALGORITMA
(LOGIKA PEMROGRAMAN)

Disusun oleh:
Drs. Suarga. M.Sc, M.Math, Ph.D

**(Setelah mengikuti kuliah ini mahasiswa diharapkan :
mampu menyusun suatu algoritma penyelesaiaan masalah
yang menjadi dasar pembuatan program komputer)**

Makassar , September 2002

1. Pendahuluan

1.1 Konsep Algoritma

- **Algoritma :** teknik penyusunan langkah-langkah penyelesaian masalah dalam bentuk kalimat dengan jumlah kata terbatas tetapi tersusun secara logis dan sistematis.
- **Abu Ja'far Muhammad Ibnu Musa Al-Kwarizmi :** penulis buku "Aljabar wal muqabala" beberapa abad yang lalu, dianggap sebagai pencetus pertama dari Algoritma, bahkan kata "Algoritma"diambil dari kata "Al-Kwarizmi" yang kemudian berubah menjadi "Algorism", selanjutnya menjadi "Algorithm".
- **Donald E.Knuth :** seorang penulis beberapa buku algoritma abad XX, menyatakan bahwa ada beberapa ciri algoritma, yaitu:
    - Algoritma mempunyai awal dan akhir, suatu algoritma harus berhenti setelah mengerjakan serangkaian tugas atau langkahnya terbatas.
    - Setiap langkah harus didefinisikan dengan tepat sehingga tidak memiliki arti ganda (not ambiguous).
    - Memiliki masukan (input) atau tidak sama sekali.
    - Memiliki keluaran (output) atau tidak sama sekali.
    - Algoritma harus effektif.
- Algoritma bisa ditemukan dalam kehidupan sehari-hari, misalnya sbb:

Proses	**Algoritma**	**Contoh**
1. Membuat Kue	Resep Kue	Campurkan 2 butir telur kedalam adonan, kemudian kocok hingga mengembang
2. Membuat Pakaian	Pola pakaian	Gunting kain dari pinggir kiri bawah ke arah kanan atas sepanjang 15 cm.
3. Praktikum Kimia	Petunjuk Praktikum	Campurkan 10 ml Asam Sulfat ke dalam 15 ml Natrium hidroksida.

- **Struktur Algoritma :** agar algoritma dapat ditulis lebih teratur maka sebaiknya dibagi ke dalam beberapa bagian. Salah struktur yang sering dijadikan patokan adalah sebagai berikut:
    - **Bagian Kepala (Header) :** memuat nama algoritma serta informasi atau keterangan tentang algoritma yang ditulis.
    - **Bagian Deklarasi :** memuat defenisi tentang nama variable, nama tetapan, nama prosedur, nama fungsi, tipe data.
    - **Bagian Deskripsi :** memuat langkah-langkah penyelesaian masalah, termasuk beberapa perintah seperti baca data, tampilkan, ulangi, dsb.

- Berikut ini adalah contoh sebuah algoritma yang mengikuti struktur tersebut diatas:

Algoritma Luas_lingkaran
{ menghitung luas sebuah lingkaran apabila jari-jari lingkaran tersebut diberikan }

Deklarasi
{ Defenisi nama tetapan }
const N = 10;
const phi = 3.14;

{ defenisi nama peubah / variable }
real jarijari, luas;

Deskripsi
read(jarijari**)**;
luas = phi * jarijari * jarijari;
write(luas**);**

Contoh berikut ini adalah algoritma untuk menghitung nilai rata sejumlah angka yang dimasukkan lewat keyboard.

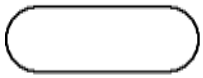
Algoritma Nilai_Rata
{menghitung nilai rata sejumlah bilangan yang dimasukkan lewat keyboard }

Deklarasi
integer x, N, k, jumlah;
real nilai_rata;

Deskripsi
{ masukkan jumlah data }
read(N**);**
k ← 1;
jumlah ← 0;
while (k <= N**) do**
{ baca data }
read(x);
jumlah ← jumlah + x;
k ← k + 1;
endwhile
{ hitung nilai rata }
nilai_rata ← jumlah / N;
write(nilai_rata);

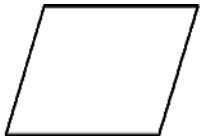
1.2 Flowcharting

- Suatu teknik untuk menyusun rencana program telah diperkenalkan dan telah dipergunakan oleh kalangan pemrogram komputer sebelum algoritma menjadi populer, yaitu ***flowcharting***. Flowchart adalah untaian simbol gambar (chart) yang menunjukkan aliran (flow) dari proses terhadap data.
- Simbol-simbol flowchart dapat diklassifikasikan menjadi: simbol untuk program dan simbol untuk sistem (peralatan hardware).
- Program Flowchart :

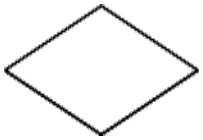


Terminator
untuk Mulai atau Selesai

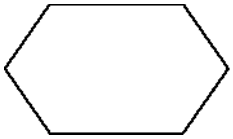
Proses
menyatakan proses terhadap data



Input / Output
menerima input atau menampilkan output

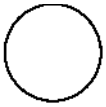


Seleksi/Pilihan
memilih aliran berdasarkan syarat

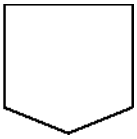


Predefined-Data
definisi awal dari variabel atau data

Predefined-Process
lambang fungsi atau sub-program



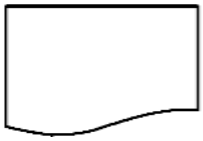
Connector
penghubung pada halaman



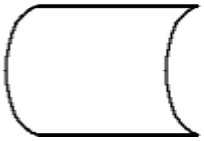
Off-page Connector
penghubung halaman yang berbeda.

- System Flowchart :

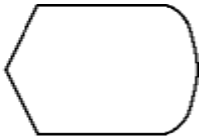
Keyboard



Printer



File / Storage



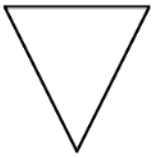
Display / Monitor

Magnetic Tape

Magnetic Disk

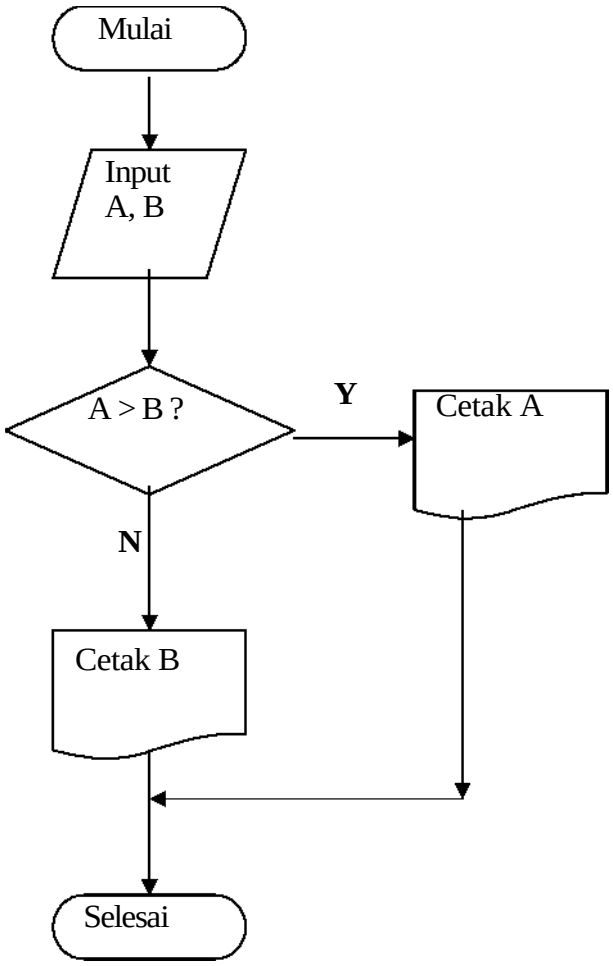
Sorting

Extract

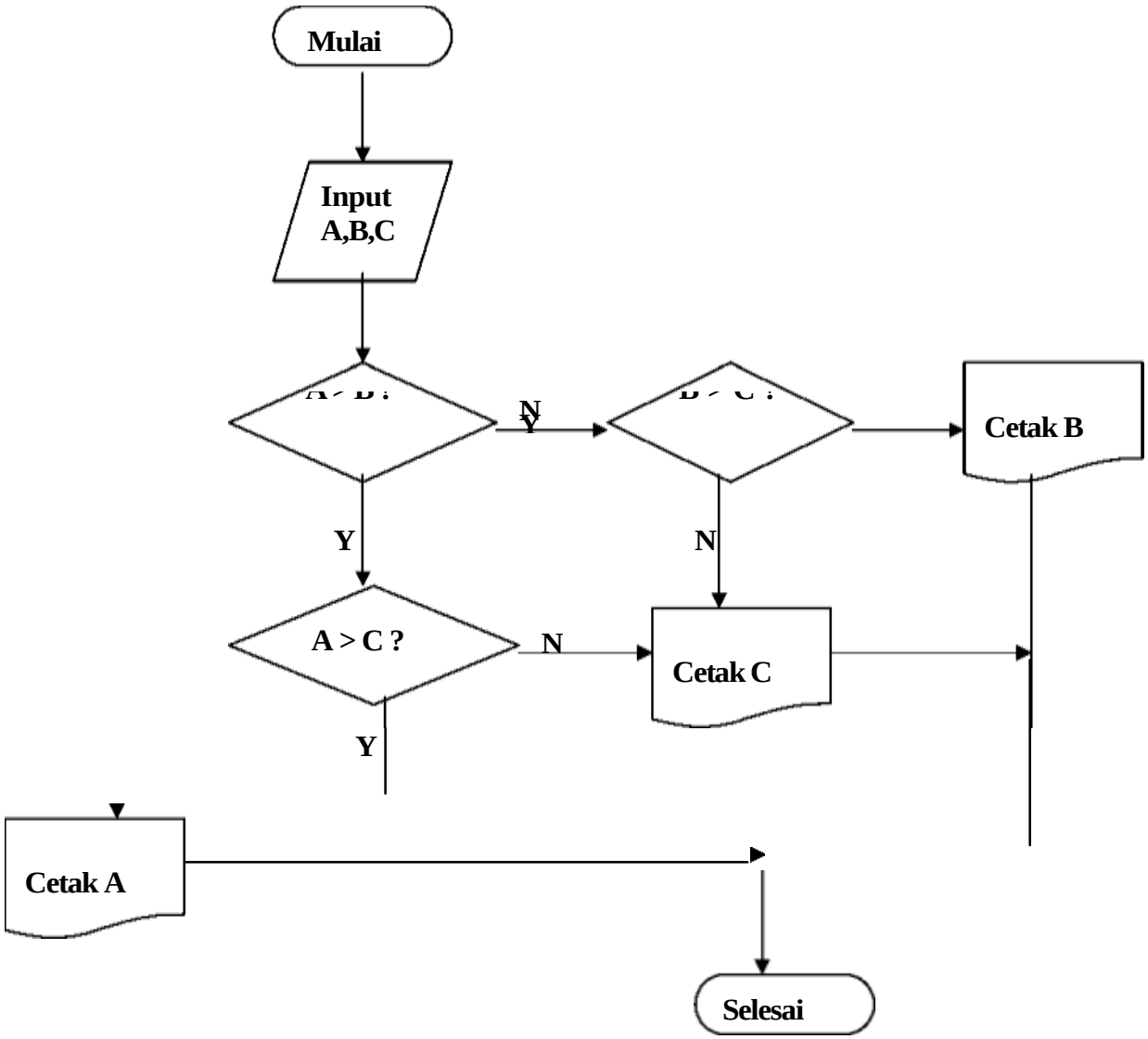


Merge

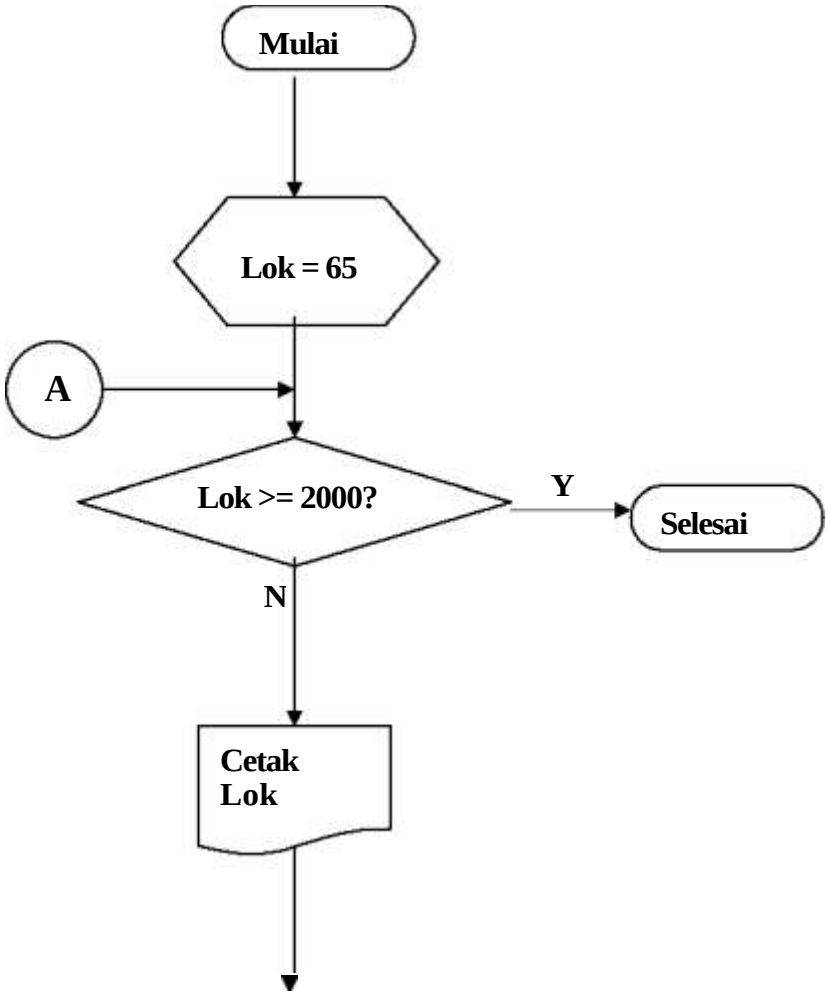
Contoh 1 : Proses memilih satu bilangan yang lebih besar dari yang lainnya.



Contoh 2 : Proses memilih satu bilangan terbesar dari tiga bilangan.



Contoh 3 : Jalan raya trans Sulawesi sepanjang 2000 Km akan segera ditingkatkan, pada setiap jarak 65 Km akan dibangun fasilitas berupa pompa bensin, kafe, tempat istirahat, dan tempat ibadah. Tampilkan lokasi pada jarak kilometer berapa fasilitas tersebut akan dibangun.



LATIHAN

1. Gambar flowchart untuk mengganti ban bocor dengan ban reserve.

2. Gambar flowchart untuk meyiapkan secangkir kopi manis dipagi hari (dimulai dari memasak air hingga menghidangkan kopi).

3. Gambar flowchart untuk memilih satu bilangan terbesar dari empat bilangan.

4. Gambar flowchart untuk memilih satu bilangan terbesar dari N buah bilangan.

2. Tipe Data, Variable, dan Nilai

. Pada prinsipnya suatu program komputer memanipulasi data untuk menjadi informasi yang berguna. Ada tiga hal yang berkaitan dengan data, yaitu:

- ***Tipe data :*** setiap data memiliki tipe data, apakah data itu merupakan angka bulat (integer), angka biasa (real), atau berupa karakter (char), dsb.
- ***Variabel :*** setiap data diwakili oleh suatu variable, dan variable ini diberi nama agar bisa dibedakan terhadap variable lainnya.
- ***Nilai :*** setiap data memiliki harga atau nilai, misalnya umur seseorang diwakili oleh variabel UMUR yang bertipe bilangan, dan memiliki nilai 20 tahun. Perlu diketahui bahwa dalam representasi nilai data dalam komputer setiap tipe data memiliki batasan nilai masing-masing.

2.1 Tipe Data

- Ada dua kategori dari tipe data, yaitu: ***tipe dasar*** dan ***tipe bentukan***.
- **Tipe dasar :** adalah tipe data yang selalu tersedia pada setiap bahasa pemrograman, antara lain: *bilangan bulat* (**integer**), *bilangan biasa* (**real**), bilangan tetap (**const**), *karakter* (**character** atau **char**), *logik* (**logic** atau **boolean**).
- **Tipe bentukan :** adalah tipe data yang dibentuk dari kombinasi tipe dasar, antara lain: *larik* (**array**), *rekaman* (**record**), *string* (**string**).

Bilangan bulat (integer)

- Bilangan atau angka yang tidak memiliki titik desimal atau pecahan, seperti: 10, +255, -1024, +32767.
- Tipe dituliskan sebagai : **integer** atau **int**
- Jangkauan nilai : bergantung pada implementasi perangkat keras komputer, misalnya dari -32768 s/d +32767, untuk algoritma tidak kita batasi.
- Operasi aritmetik : tambah **+** , kurang **-** , kali ***** , bagi /, sisa hasil bagi **%**
- Operasi pembandingan : lebih kecil < , lebih kecil atau sama < =
  lebih besar > , lebih besar atau sama > =
  sama = , tidak sama ><

Bilangan biasa (real)

- Bilangan atau angka yang bisa memiliki titik desimal atau pecahan, dan ditulis sebagai: 235.45, +1023.55, -987.3456 atau dalam notasi ilmiah seperti: 1 .245E+03, 7.45E-02, +2.34E-04, -5.43E+04, dsb.
- Tipe dituliskan sebagai : **real**
- Jangkauan nilai : bergantung pada implementasi perangkat keras komputer, misalnya dari -2.9E-39 s/d +1.7E+38, untuk algoritma tidak kita batasi.
- Operasi aritmatik dan pembandingan juga berlaku bagi bilangan biasa.

Bilangan tetap (const)

- Bilangan tetap **(const)** adalah tipe bilangan baik bernilai bulat maupun tidak yang nilainya tidak berubah selama algoritma dilaksanakan.
- Tipe dituliskan sebagai : **const**
- Jangkauan nilai meliputi semua bilangan yang mungkin

Karakter (character)

- Karakter adalah data tunggal yang mewakili semua huruf, simbol baca, dan juga simbol angka yang tidak dapat dioperasikan secara matematis, misalnya: 'A', 'B', ... , 'Z', 'a', 'b', ..., 'z', '?', '!', ':', ';' dst.
- Tipe dituliskan sebagai : **char**
- Jangkauan nilai meliputi semua karakter dalam kode ASCII, atau yang tertera pada setiap tombol keyboard.
- Operasi pembandingan dapat dilakukan dan dievaluasi menurut urutan kode ASCII, sehingga huruf 'A' (Hex 41) sebenarnya lebih kecil dari huruf 'a' (Hex 61).

Logik (Logical)

- Tipe data logik adalah tipe data yang digunakan untuk memberi nilai pada hasil pembandingan, atau kombinasi pembandingan.
- Tipe dituliskan sebagai : **boolean**
- Jangkauan nilai ada dua: **true** dan **false**
- Contoh: 45 > 56 hasilnya **false**, Amir < Husni hasilnya **true**
- Ada beberapa operasi untuk data jenis logik, antara lain: **and, or,** dan **not**

A	B	A and B	A or B	Not A
False	**False**	**False**	**False**	**True**
False	**True**	**False**	**True**	**True**
True	**False**	**False**	**True**	**False**
True	**True**	**True**	**True**	**False**

Array (larik)

- Array adalah tipe data bentukan, yang merupakan wadah untuk menampung beberapa nilai data yang sejenis. Kumpulan bilangan bulat adalah array integer, kumpulan bilangan tidak bulat adalah array real.
- Cara mendefinisikan ada dua macam, yaitu:
  - ***Nilai_ujian : array [1 .. 10] of integer***; atau
  - ***int nilai_ujian[10]***;
- Kedua definisi diatas menunjukkan bahwa nilai_ujian adalah kumpulan dari 10 nilai bertipe bilangan bulat.

String

- String adalah tipe data bentukan yang merupakan deretan karakter yang membentuk satu kata atau satu kalimat, yang biasanya diapit oleh dua tanda kutip.
- Sebagai contoh : nama, alamat, dan judul adalah tipe string.
- Cara mendefinisikannya adalah:
  - ***String Nama, Alamat;*** atau
  - ***Nama, Alamat :*** **String;**

Record (rekaman)

- Record adalah tipe data bentukan yang merupakan wadah untuk menampung elemen data yang tipe-nya tidak perlu sama dengan tujuan untuk mewakili satu jenis objek.
- Sebagai contoh, mahasiswa sebagai satu jenis objek memiliki beberapa elemen data seperti: nomer_stb, nama, umur, t4lahir, jenkel.
- Cara mendefinisikan record mahasiswa tersebut adalah sbb:

```
Type DataMhs : record
      < nomer_stb : integer,
        nama_mhs : string,
        umur : integer,
        t4lahir : string,
        jenkel : char;
      >
```

2.2 Variable

- Variable adalah nama yang mewakili suatu elemen data seperti: jenkel untuk jenis kelamin, t4lahir untuk tempat lahir, alamat untuk alamat, dsb.
- Ada aturan tertentu yang wajib diikuti dalam pemberian nama variable antara lain:
  - Harus dimulai dengan abjad, tidak boleh dengan angka atau simbol
  - Tidak boleh ada spasi diantaranya
  - Jangan menggunakan simbol-simbol yang bisa membingungkan seperti titik dua, titik koma, koma, dsb.
  - Sebaiknya memiliki arti yang sesuai dengan elemen data
  - Sebaiknya tidak terlalu panjang
- Contoh variable yang benar : Nama, Alamat, Nilai_Ujian
- Contoh variable yang salah : 4XYZ, IP rata, Var:+xy,458;

2.3 Pemberian Nilai

- Ada dua cara yang dapat digunakan untuk memberi nilai pada suatu variable yaitu melalui proses : ***assignment*** dan ***pembacaan***.
- Pemberian nilai dengan cara assignment mempunyai bentuk umum sebagai berikut:
  - *Variable ← nilai;*
  - *Variable1 ← variable2;*
  - *Variable ← ekspressi;*
- Contoh assignment:
  - Nama ← "Ali bin Abu Thalib";
  - Jarak ← 100.56;
  - X ← Jarak;
  - Rentang ← X + 50 – 3*Y;
- Pemberian nilai dengan cara pembacaan dapat dilakukan melalui instruksi dengan bentuk umum sbb:
  - **read(**variable**);** atau
  - **read(** variable1, variable2, ... **);**
- Contoh pembacaan data:
  - **read(Nama);**
  - **read(Jarak, Rentang, X);**

2.4 Menampilkan Nilai

- Agar hasil pelaksanaan algoritma dapat dikomunikasikan maka nilai variable yang diproses dalam algoritma dapat ditampilkan.
- Instruksi untuk menampilkan nilai variable adalah: **write(**variable, ... **);**
- Contoh penampilan nilai adalah sbb:
  - **write("nama anda : ", Nama);**
  - **write("nilai ujian = ", nilai);**
  - **write("Jumlah variable = ", X + Y + Z);**

2.5 Ekspressi (Expression)

- Transformasi data dan peubah dalam bentuk persamaan yang direlasikan oleh *operator* dan *operand.*
- **Operand** adalah data, tetapan, peubah, atau hasil dari suatu fungsi.
- **Operator** adalah simbol-simbol yang memiliki fungsi untuk menghubungkan operand sehingga terjadi transformasi. Jenis-jenis operator adalah sbb:
  - **Operator aritmetika :** operator untuk melakukan fungsi aritmetika seperti: + (menjumlah), - (mengurangkan), ***** (mengalikan), / (membagi).
  - **Operator relational :** operator untuk menyatakan relasi atau perbandingan antara dua operand, seperti : > (lebih besar), < (lebih kecil),

>= (lebih besar atau sama), <= (lebih kecil atau sama), == (sama), **!=** (tidak sama) atau ><.

- **Operator logik :** operator untuk merelasikan operand secara logis, seperti **&&** (and), **||** (or), dan **!** (not).
- **Operator string :** operator untuk memanipulasi string, seperti : **+** (concatenation), dan **substr** (substring, mencuplik).

- Berdasarkan pada jenis operator yang digunakan maka ada empat macam ekspressi, yaitu: *ekspressi aritmetika, ekspressi relational, eskspressi logik, dan ekspressi string.*
- **Ekspressi Aritmetika :** ekspressi yang memuat operator aritmetika, contoh:
  - T ← 5 * (C + 32) / 9;
  - Y ← 5 * ((a + b) / (c + d) + m / (e * f));
  - Gaji ← GaPok * (1 + JumNak * 0.05 + Lembur * 1.25);
- **Ekspressi Relational :** ekspressi yang memuat operator relational, contoh:
  - Nilai_A > Nilai_B
  - (A + B) < (C + D)
  - (x + 57) != (y + 34)
- **Ekspressi Logik :** ekspressi yang memuat operator logik, contoh:
  - m ← (x > y) && (5 + z)
  - n ← (!A || !(B && C))
- **Ekspressi String :** eskpressi dengan operator string, contoh:
  - Alamat ← "Jl. P. Kemerdekaan " + "Km 9 Tamalanrea"
  - Hasil ← "Saudara : " + Nama + "adalah mahasiswa"
  - Tengah ← Substr(Kalimat, 5, 10);

Contoh Algoritma:

1. Susun algoritma yang menghitung pajak pertambahan nilai (ppn) 12.50% dengan meminta harga barang yang dibeli dari pengguna program.

Algoritma PPN
{ menghitung pajak pertambahan nilai 12.50% dari harga barang **}**

Deklarasi
real harga, pajak, total**;**

Deskripsi

write (“Masukkan harga barang : **“);**
read(harga**);**

pajak ← 0.125 * harga;
total = harga + pajak;

write(“Harga = “, harga, “ pajaknya = “, pajak**);**
write(“Total = “, total**);**

2. Susun algoritma yang meminta data dasar mahasiswa (mis: Nama, Alamat, e_mail, dan telepon) kemudian menampilkan-nya kembali tersusun.

Algoritma Data_dasar
{ membaca dan menampilkan data dasar mahasiswa**}**

Deklarasi
string nama, alamat, e_mail, telepon**;**

Deskripsi

write (“Masukkan nama anda : **“);**
read(nama);
write(“Dimana alamatnya : **“);**
read(alamat**);**
write(“No telepon : **“);**
read(telepon**);**
write(“Alamat e-mail : **“);**
read(e_mail**);**

write(nama**);**
write(alamat, telepon**);**
write(e_mail**);**

SOAL :

1. Definisikan sebuah record untuk data pegawai yang terdiri atas elemen Nomer_Pegawai (NIP), Nama, TgLahir, Tempat_Lahir, Jen_Kelamin, Agama, Status, Pangkat.

2. Tulis algortima sederhana untuk membaca dan menampilkan kembali data pegawai yang sesuai dengan struktur record pada soal 1.

3. Instruksi Utama

- Secara garis besar hanya ada tiga macam kategori instruksi utama, yaitu:

    1. **Instruksi Runtunan (Sequential)**
    2. **Instruksi Pemilihan (Selection)**
    3. **Instruksi Perulangan (Repetition)**

3.1 Instruksi Runtunan (Sequential)

- Instruksi runtunan adalah instruksi yang dikerjakan secara beruntun atau ber-urutan baris per-baris mulai dari baris pertama hingga baris terakhir, tanpa ada loncatan atau perulangan.

a. tiap instruksi dikerjakan sekali satu per-satu
b. urutan pelaksanaan instruksi sama dengan urutan penulisan algoritma
c. instruksi terakhir merupakan akhir dari algoritma
d. urutan penulisan instruksi bisa menjadi penting, bila diubah dapat menyebabkan hasil yang berbeda.

Contoh 1:

```
Algoritma Runtunan_1
{ menunjukkan urutan yang berbeda memberi hasil yang berbeda }

Deklarasi
        integer A, B;

Deskripsi
        A ← 10;
        A ← 2 * A;
        B ← A;
        write(B);
```

Algoritma diatas menampilkan hasil : 20

```
Algoritma Runtunan_2
{ menunjukkan urutan yang berbeda memberi hasil yang berbeda }

Deklarasi
        integer A, B;

Deskripsi
        A ← 10;
        B ← A;
        A ← 2 * A;
```

dengan urutan yang diubah maka algoritma ini memberi hasil : 10

Algoritma Runtunan_3;
{ algoritma untuk menghitung luas sebuah segitiga **}**

Deklarasi

real Alas, Tinggi;
real Luas;

Deskripsi

write (“Masukkan panjang alasnya : “);
read (Alas);
write (“Masukkan tingginya : “);
read (Tinggi);

Luas ← Alas * Tinggi / 2;
write (“Luas segitiga = “, Luas);

Algoritma Runtunan_4;
{ algoritma menampilkan gaji bersih pegawai, dengan memasukkan gaji pokok kemudian menghitung tunjangan sebesar 25%, dan pajak pph 15% **}**

Deklarasi

string nama;
real gajipokok, tunjangan, pajak;
real gajibersih;

Deskripsi

write (“Masukkan nama pegawai : “);
read (nama);
write (“Masukkan gaji pokoknya : “);
read (gajipokok);

tunj angan ← 0.25 * gajipokok;
pajak ← 0.15 * (gajipokok + tunjangan);
gajibersih ← gajipokok + tunjangan – pajak;

write (“Gaji saudara : “, nama);
write (“adalah = “, gajibersih);

3.2 Instruksi Pemilihan

Instruksi pemilihan adalah instruksi yang dipakai untuk memilih satu aksi dari beberapa kemungkinan aksi berdasarkan suatu persyaratan. Ada dua bentuk instruksi pemilihan yang sering digunakan yaitu:

a. Instruksi **if / then / else**
b. Instruksi **case**

Instruksi if / then / else

bentuk 1 kasus:

```
if (syarat)
   then aksi
endif
```

contoh : **if (** x > 100 **)**
then x ← x + 1
endif.

bentuk 2 kasus:

```
if ( syarat )
   then aksi-1
 else aksi-2
 endif.
```

contoh : **if (** a > 0 **)**
then write ("bilangan ini positif "**)**
else write ("bilangan ini negatif "**)**
endif.

bentuk bersusun (lebih dari 1 syarat) :

```
if ( syarat-1 )
   then aksi-1
   else if ( syarat-2 )
            then aksi-2
            else aksi-3
        endif
endif.
```

Contoh :

Algoritma Pemilihan_1
{ contoh algoritma untuk menunjukkan pemakaian instruksi pemilihan. algoritma ini menerima satu bilangan bulat kemudian memeriksanya apakah bilangan genap atau bilangan ganjil **}**

Deklarasi
integer bilangan;

Deskripsi
write (“masukkan satu bilangan bulat : “**);**
read (bilangan**);**

if (bilangan **%** 2 == 0 **)**
then write (“bilangan genap ! “**);**
else write (“bilangan ganjil ! “**);**
endif

Algoritma Pemilihan_2
{ contoh algoritma ini menerima 3 bilangan bulat kemudian menetapkan bilangan yang terbesar **}**

Deklarasi
integer A, B, C, maks;

Deskripsi
write (“masukkan bilangan 1 : “**);**
read (A **);**
write (“masukkan bilangan 2 : “**);**
read (B **);**
write (“masukkan bilangan 3 : “**);**
read (C **);**

if (A > B **)**
then maks ← A;
else maks ← B;
endif.

if (C > maks **)**
then maks ← C;
endif.

write (“maksimum = “, maks**);**

```
Algoritma Pemilihan_3
{ contoh algoritma ini menerima 3 bilangan bulat kemudian menetapkan
  bilangan yang terbesar, memanfaatkan bentuk bersusun }

Deklarasi
        integer A, B, C, maks;

Deskripsi
        write (“masukkan bilangan 1 : “);
        read ( A );
        write (“masukkan bilangan 2 : “);
        read ( B );
        write (“masukkan bilangan 3 : “);
        read ( C );

        if ( A > B )
           then if ( A > C )
                      then write (“ maksimum = “, A );
                      else write (“ maksimum = “, C );
                endif
            else if ( B > C )
                      then write (“ maksimum = “, B );
                      else write (“ maksimum = “, C );
                 endif
        endif
```

Instruksi case

Instruksi **case** digunakan apabila setiap *aksi dilaksanakan berdasarkan satu macam nilai dari suatu variable yang mungkin memiliki lebih dari dua macam nilai*

Bentuk instruksi case :

```
case ( variable )
                                  nilai-1 : aksi-1;
                                  nilai-2 : aksi-2;
                                  nilai-3 : aksi-3;
        .............
        default : aksi-n;
endcase.
```

Contoh : Gaji karyawan pada sebuah perusahaan di-dasarkan pada jam-kerja dalam satu bulan serta posisi atau golongannya dalam perusahaan itu. Upah perjam menurut golongan adalah sbb:

Golongan	Upah/jam (Rp)
A	**5000**
B	**6000**
C	**7500**
D	**9000**

Apabila karyawan bekerja lebih dari 150 jam perminggu, maka kelebihan jam kerja tersebut dihitung sebagai lembur dengan upah/jam 25% diatas upah reguler. Buat sebuah algoritma yang menerima nama, golongan, serta jam-kerja karyawan, kemudian menampilkan gaji total-nya dalam satu bulan.

```
Algoritma Gaji_Karyawan
{ algortima yang menerima nama, golongan serta jam-kerja kemudian menampilkan
  total gaji yang diterima karyawan }

Deklarasi
    real    gaji, total, jamkerja, lembur, upah;
    string  nama;
    char    golongan;

Deskripsi
    write (" masukkan nama karyawan : ");
    read ( nama );
    write (" masukkan golongan-nya : ");
    read ( golongan );
    write (" masukkan jam kerjanya : ");
    read ( jamkerja );

    case ( golongan )
        ' A ' : upah ← 5000;
        ' B ' : upah ← 6000;
        ' C ' : upah ← 7500;
        ' D ' : upah ← 9000;
        default : write (" golongannya salah ! ");
    endcase.

    if ( jamkerja > 150 )
      then lembur ← ( jamkerja – 150 ) * upah * 1.25;
           gaji ← 150 * upah;
      else lembur ← 0;
           gaji ← jamkerja * upah;
    endif

    total ← gaji + lembur;
    write (" Gaji yang diterima sdr : ", nama, " adalah = Rp. ", total);
```

3.3 Instruksi Pengulangan (Repetition)

Instruksi pengulangan adalah instruksi yang dapat mengulangi pelaksanaan sederetan instruksi-instruksi lainnya berulang-kali sesuai dengan persyaratan yang ditetapkan.

Struktur instruksi pengulangan pada dasarnya terdiri atas :

a. *Kondisi perulangan* : suatu kondisi yang harus dipenuhi agar perulangan dapat terjadi.
b. *Badan (body) perulangan* : deretan instruksi yang akan diulang-ulang pelaksanaan-nya.
c. *Pencacah (counter) perulangan* : suatu variable yang nilainya harus berubah agar perulangan dapat terjadi dan pada akhirnya membatasi jumlah perulangan yang dapat dilaksanakan.

Bentuk instruksi perulangan adalah :

1. Perulangan : **while – do**
2. Perulangan : **repeat – until**
3. Perulangan : **for**

Perulangan while – do :

Bentuk umum :

while (kondisi**) do**
.....
instruksi-instruksi

endwhile.

maknanya : ulangi .. instruksi-instruksi .. selama kondisi masih terpenuhi.
perlu perhatian :
- ada instruksi yang berkaitan dengan kondisi sebelum while/do
- ada satu instruksi diantara instruksi-instruksi yang diulang untuk membatasi jumlah perulangan.

Contoh: algoritma untuk menampilkan angka 1 hingga 100

{ mencetak angka 1 hingga 100 **}**

Deklarasi
    integer angka;

Deskripsi
    angka ← 1;
    while (angka < 101 **) do**
        write (angka **);**
        angka ← angka + 1;
    endwhile.

Contoh: mencetak syair "anak ayam yang mati" mulai dari 10 hingga habis

Algoritma Perulangan_2
{ mencetak syair anak ayam **}**

Deklarasi
    integer anak;

Deskripsi
    anak ← 10;
    while (anak > 0 **) do**
        write ("anak ayamku turun ", anak);
        anak ← anak – 1;
        if (anak > 0 **)**
            then write ("mati satu tinggal ", anak**);**
        else write ("mati satu tinggal saya "**);**
        endif.
    endwhile.

Perulangan Repeat – Until:

Bentuk Umum :

repeat

instruksi – instruksi

until (kondisi **).**

maknanya : ulangi pelaksanaan instruksi-instruksi hingga kondisi terpenuhi.
perhatian : - apabila kondisi tidak terpenuhi maka instruksi-instruksi akan diulang - instruksi-instruksi akan dikerjakan sebelum kondisi diperiksa

Contoh : menampilkan “Halo ... “ sebanyak 25 kali.

```
Algoritma Perulangan_3
{ memakai repeat-until untuk menampilkan Halo sebanyak 25 kali }
Deklarasi
        integer cacah;

Deskripsi
        cacah ← 1;
        repeat
                write ( “Halo ... “ );
            cacah ← cacah + 1;
            until ( cacah > 25 ).
```

Contoh : gunakan repeat-until untuk menghitung jumlah angka 1 + 2 + 3 + ... + N, dimana N adalah angka bulat yang dimasukkan lewat keyboard.

```
Algoritma Perulangan_4
{ menghitung jumlah 1 + 2 + 3 + ... + N, N dimasukkan lewat keyboard }

Deklarasi
        integer cacah, N, Jumlah;

Deskripsi
        write ( “Masukkan nilai N : “);
        read ( N );

        cacah ← 1;
        Jumlah ← 0;
        repeat
                Jumlah ← Jumlah + cacah;
                cacah ← cacah + 1;
        until ( cacah > N ).
        write ( “Jumlahnya = “, Jumlah);
```

Contoh : gunakan repeat-until untuk menghitung rata-rata dari N buah bilangan yang dimasukkan lewat keyboard.

Algoritma Perulangan_6
{ menghitung rata-rata dari N buah bilangan **}**

Deklarasi
integer bilangan, cacah, N, Jumlah;
real Rata;

Deskripsi
write ("Masukkan N : "**);**
read (N **);**

cacah ← 1;
Jumlah ← 0;
repeat
write ("masukkan bilangan ke-", cacah**);**
read (bilangan **);**
Jumlah ← Jumlah + bilangan;
cacah ← cacah + 1;
until (cacah > N **).**
Rata ← Jumlah / N;
write ("rata-rata = ", Rata **);**

Perulangan for:

Bentuk umum:

for (var = *awal* **to** *akhir* **step** n**)**

instruksi – instruksi

endfor.

maknanya : ulangi instruksi-instruksi tersebut berdasarkan variabel perulangan mulai dari nilai awal hingga nilai akhir dengan perubahan nilai sebesar n.

perhatian : - variabel perulangan (var) harus bertipe dasar (integer, real, atau char)
- nilai *awal* harus lebih kecil dari *akhir* bila n > 0 (positif)
- nilai *awal* harus lebih besar dari *akhir* bila n < 0 (negatif)
- mula-mula variabel var bernilai awal, kemudian setiap satu kali putaran maka nilai var bertambah sebesar n
- perulangan akan berhenti apabila nilai var sudah mencapai akhir

Contoh : menampilkan "Halo ... " sebanyak 10 kali

Algoritma Perulangan_6
{ menampilkan Halo ... memakai instruksi **for }**

Deklarasi
integer cacah;

Deskripsi
for (cacah = 1 **to** 10 **step** 1)
write ("Halo ... "**);**
endfor.

contoh : menghitung nilai rata dari N buah bilangan, menggunakan instruksi for.

Algoritma Perulangan_7
{ menghitung nilai Rata dari N buah bilangan **}**

Deklarasi
integer cacah, N, angka, Jumlah;
real Rata;

Deskripsi
write ("Masukkan berapa bilangan : "**);**
read (N **);**

Jumlah ← 0;
for (cacah = 1 **to** N **step** 1 **)**
write ("Masukkan bilangan ke – ", cacah**);**
read (angka **);**
Jumlah ← Jumlah + angka;
endfor.

Rata ← Jumlah / N;
write ("Rata-rata = ", Rata**);**

contoh : melakukan pencacahan mundur mulai dari 100, 99, 98, ... hingga 0

Algoritma Perulangan_8
{ mencacah terbalik atau count down **}**

Deklarasi
integer cacah;

Deskripsi
for (cacah = 100 **to** 0 **step** -1**)**
write (cacah **);**
endfor.
write ("Go !" **);**

4. STUDI KASUS

Setelah mempelajari beberapa instruksi utama maka pada dasarnya modal untuk merancang algoritma sederhana sudah memadai, oleh sebab itu pada bagian ini akan disajikan beberapa contoh soal yang dapat dijadikan sebagai studi kasus bagi mahasiswa yang mempelajari algoritma.

4.1 Akses data langsung.

Seorang sekretaris memerlukan satu program sederhana yang dapat membantu dia untuk mengetahui nomer telepon seseorang dengan cepat tanpa harus membuka-buka buku agendanya.

Andaikan nama-nama orang tersebut adalah sebagai berikut:

Anton	(0411) 324-678
Bahrul	(021) 434-6783
Charles	(022) 256-1234
Daud	(0411) 567-342
Endang	(0411) 344-235
Fahri	(021) 765-0856
Gunarsih	(0421) 123-876

Analisis:

1. Ketika program dijalankan maka muncul permintaan untuk memasukkan satu nama
2. Nama ini kemudian dicari misalnya dengan rentetan **if / then / else** atau dengan instruksi **case().**
3. Bila nama tersebut ketemu maka nomer-teleponnya ditampilkan.
4. Bila nama tersebut tidak ada maka tampilkan "nama tsb tidak ada!".

Algoritma Buku_telepon
{ mencari nomer telepon seseorang **}**

Deklarasi
string nama, notelp;

Deskripsi
write ("Ketik namanya : "**);**
read (nama **);**

case (nama**)**
'Anton' **:** notelp . '(0411) 324 – 678';
'Bahrul' **:** notelp . '(021) 434 – 6783';
'Charles' **:** notelp . '(022) 256-1234';

```
        'Daud' : notelp ← '(0411) 567-342';
        'Endang' : notelp ← '(0411) 344-235';
        'Fahri' : notelp ← '(021) 765-0856';
        'Gunarsih' : notelp ← '(0421) 123-876';
        default : notelp ← 'nama tsb tdk ada';
    endcase.

    write ( notelp );
```

4.2 Menjumlahkan Deret

Buatlah sebuah algoritma untuk menghitung jumlah deret dengan N buah suku sbb:
S = 1 – 1/2 + 1/4 - 1/6 + 1/8 - 1/10 + 1/12 - 1/14 + ...

Analisis:

1. Ketika dijalankan maka akan ada permintaan untuk memasukkan jumlah suku N
2. Bila diperhatikan maka tanda berselang seling positif dan negatif, pada posisi ganjil maka tandanya positif dan pada posisi genap tandanya negatif
3. Nilai yang dijumlahkan adalah kelipatan dari (1/2) yang dikalikan sesuai dengan posisi-nya, mula-mula 1/2 kemudian 1/(2*2), 1/(2*3), ...

```
Algoritma Jumlah_Deret
{ menjumlahkan deret bersuku N }

Deklarasi
    integer N, cacah, k;
    real S;

Deskripsi
    write ( "Berapa banyak suku ? ");
    read ( N );

    S ← 1;
    cacah ← 1;
    k ← 0;

    while ( cacah < = N ) do
        cacah ← cacah + 1;
        k ← k + 2;
        if ( cacah % 2 = 0 )
            then S ← S – 1/k;
            else S ← S + 1/k;
        endif.
    endwhile.
    write ( "Jumlah deret = ", S);
```

Algoritma Jumlah_Deret_V2
{ cara lain untuk menghitung jumlah deret **}**

Deklarasi
integer cacah, N, k, tanda;
real S;

Deskripsi
write ("Berapa banyak suku ? "**);**
read (N **)**;

S ← 1;
cacah ← 1;
k ← 0;
tanda ← +1;

while (cacah < = N **) do**
k ← k + 2;
cacah ← cacah + 1;
tanda ← (-1) * tanda;
S ← S + tanda * (1/k);
endwhile.

write ("Jumlah deret = ", S**);**

4.3 Mengelompokkan Data

Andaikan dari keyboard dimasukkan N buah data (bilangan bulat) kemudian akan dikelompokkan menjadi dua macam yaitu kelompok bilangan ganjil dan kelompok bilangan genap dalam bentuk jumlahan sehingga keluaran (output) berbentuk sebagai berikut:

Jumlah bilangan Ganjil = ...
Jumlah bilangan Genap = ...

Analisis:
1. Berapa banyak data harus diketahui terlebih dahulu N
2. Lakukan perulangan sebanyak N kali untuk:
   a. meminta data
   b. memeriksa data apakah masuk ganjil atau genap
   c. menjumlahkan data sesuai kelompoknya
3. Tampilkan hasil penjumlahan.

```
Algoritma GanjilGenap
{ mengelompokkan data dalam bentuk jumlahan bilangan ganjil dan genap }

Deklarasi
    integer cacah, N, angka, Genap, Ganjil;

Deskripsi
    write ( "Berapa banyak bilangan ? " );
    read ( N );
    cacah ← 1;
    Genap ← 0;
    Ganjil ← 0;

    repeat
        write ( "Masukkan bilangan ke-", cacah );
        read ( angka );
        if ( angka % 2 = 0 )
            then Genap ← Genap + angka;
            else Ganjil ← Ganjil + angka;
        endif.
        cacah ← cacah + 1;
    until ( cacah > N );

    write ( "Jumlah bilangan Ganjil = ", Ganjil);
    write ( "Jumlah bilangan Genap = ", Genap);
```

4.4 Memilih Operasi Berdasarkan Data Input

Andaikan operasi terhadap dua bilangan dapat dipilih melalui satu "menu" sebagai berikut :

```
Pilih Operasi yang di-inginkan :

+  Penjumlahan
-  Pengurangan
/  Pembagian
* Perkalian

Jenis operasi: _

Masukkan angka 1: _
Masukkan angka 2: _
Hasil = ...

Masih mau coba (Y/T) ?
```

Apabila jawaban untuk mencoba ulang adalah 'Y' maka menu operasi diatas dimunculkan kembali dan proses yang sama berulang kembali hingga jawaban pengguna program adalah 'T'.

Algoritma MenuProgram
{ memilih operasi berdasarkan pilihan pada Menu program **}**

Deklarasi
real angka1, angka2, hasil;
char pilihan, ulang;

Deskripsi
ulang ← 'Y';
while (ulang = 'Y' || ulang = 'y' **) do**
write ("Pilih Operasi yang di-inginkan :" **);**
write ("+ Penjumlahan " **);**
write (" - Pengurangan "**);**
write (" / Pembagian " **);**
write ("* Perkalian " **);**
write (" "**);**
write ("Jenis operasi: "**);**
read (pilihan **);**

write ("Masukkan angka 1: " **);**
read (angka1 **);**
write ("Masukkan angka 2: " **);**
read (angka2 **);**

case (pilihan **)**
' + ' : hasil ← angka1 + angka2;
' – ' : hasil ← angka1 – angka2;
' / ' : if (angka2 = 0 **)**
then write ("hasil tak berhingga " **);**
else hasil ← angka1 / angka2;
endif.
'*' : hasil ← angka1 * angka2;
default : write ("Pilihan operasi salah !"**);**
hasil ← 0;
endcase.
if (angka2 != 0 **)**
then write ("Hasil = ", hasil **);**
endif.

write ("Masih mau coba (Y/T) ? " **);**
read (ulang **);**
endwhile.

5. PROSEDUR DAN FUNGSI

Prosedur adalah bagian dari suatu program yang disusun secara terpisah untuk melakukan suatu tugas khusus / fungsi tertentu. Pada dasarnya ada dua macam prosedur yaitu: **Subrutin (Subprogram)** dan **Fungsi**.

Subrutin (Subprogram) adalah bagian dari program yang dibuat terpisah untuk melaksanakan sebagian dari tugas yang harus diselesaikan oleh suatu program.

Fungsi adalah bagian dari program yang dibuat terpisah untuk melaksanakan fungsi tertentu yang menghasilkan suatu nilai untuk dikembalikan ke program utama.

Manfaat dari pembuatan prosedur :

- *modularisasi* : suatu program yang besar dan kompleks dapat dibagi ke dalam beberapa prosedur sehingga setiap prosedur merupakan bagian yang mudah dikerjakan, dengan demikian maka program besar tersebut menjadi mungkin diselesaikan.
- *simplifikasi* : dalam suatu program sering diperlukan suatu tugas yang berulangkali harus dikerjakan dengan nilai-nilai variable yang berbeda, agar tidak merepotkan maka tugas ini cukup ditulis sekali saja dalam bentuk prosedur yang kemudian dipanggil berulang kali sesuai dengan kebutuhan.

5.1 Bentuk Umum Prosedur:

Prosedur nama_prosedur
{ spesifikasi dari prosedur, keadaan awal sebelum prosedur dilaksanakan dan juga keadaan akhir setelah prosedur dilaksanakan **}**

Deklarasi
{ deklarasi variabel-variabel prosedur **}**

Deskripsi
{ deskripsi dari tugas-tugas prosedur **}**

Contoh: Andaikan sebuah program menyediakan fasilitas untuk menghitung luas, keliling, dan diagonal dari sebuah persegi panjang dengan kemungkinan pemilihan melalui suatu menu.

Contoh soal diatas dapat dibagi ke dalam enam prosedur yaitu: prosedur menampilkan menu, prosedur membaca dimensi persegi panjang, menghitung luas, menghitung keliling, menghitung diagonal, dan menampilkan hasil.

```
Algoritma Empat_Persegi_Panjang
{ contoh pemakaian prosedur untuk menghitung luas, keliling, dan diagonal empat
  persegi panjang }

Deklarasi
        integer pilihan;
        real panjang, lebar, hasil;
        prosedur menu;
        prosedur baca_dimensi;
        prosedur hitung_luas;
        prosedur hitung_keliling;
        prosedur hitung_diagonal;
        prosedur tampil_hasil;

Deskripsi
        pilihan ← 0;
        repeat
                menu;
                write ( "Masukkan pilihan anda : " );
                read ( pilihan );

                if ( pilihan < 4 )
                    then baca_dimensi;
                endif.

                case ( pilihan )
                        1 : hitung_luas;
                        2 : hitung_keliling;
                        3 : hitung_diagonal;
                        4 : write ( "Selesai ... sampai jumpa " );
                    default : write ( "Pilihan salah, Ulangi ! " );
                    endcase.

                if ( pilihan < 4 )
                    then tampil_hasil;
                endif.
        until ( pilihan = 4 ).

prosedur menu
{ menampilkan menu program }

Deklarasi.

Deskripsi
        write ( "Menu Program Empat Persegi Panjang " );
        write ( " 1. Menghitung Luas " );
```

write (" 2. Menghitung Keliling " **);**
write (" 3. Menghitung Diagonal " **);**
write (" 4. Keluar dari Program" **);**

prosedur baca_dimensi
{ membaca dimensi persegi panjang **}**

Deklarasi.

Deskripsi
write ("Masukkan Panjang : "**);**
read (panjang **);**
write ("Masukkan Lebar : "**);**
read (lebar **);**

prosedur hitung_luas
{ menghitung luas empat persegi panjang **}**

Deklarasi
real luas;

Deskripsi
luas ← panjang * lebar;
hasil ← luas;

prosedur hitung_keliling
{ menghitung keliling empat persegi panjang **}**

Deklarasi
real keliling;

Deskripsi
keliling ← 2 * (panjang + lebar);
hasil ← keliling;

prosedur hitung_diagonal
{ menghitung diagonal empat persegi panjang **}**

Deklarasi
real diagonal;

Deskripsi
diagonal ← **sqrt (** panjang ^2 + lebar^2 **);**
hasil ← diagonal;

prosedur tampil_hasil
{ menampilkan hasil dari program ini **}**

Deklarasi.

Deskripsi
write ("hasil = ", hasil **);**

5.2 Variabel Lokal dan Variabel Global

Penggunaan Prosedur pada suatu program menyebabkan munculnya dua kategori variabel, yaitu variabel lokal dan variabel global. ***Variabel Lokal*** adalah variabel yang hanya dikenal dan berlaku dalam suatu prosedur saja. ***Variabel Global*** adalah variabel yang berlaku di semua bagian program dan di semua prosedur.

Semua variabel *yang didefinisikan pada deklarasi suatu prosedur* adalah *variabel lokal*, dan variabel-variabel *yang didefinisikan pada deklarasi algoritma utama* adalah *variabel global*.

Program yang menggunakan banyak variabel global terasa memudahkan karena variabel variabel tidak perlu didefinisikan lagi dalam prosedur namun terlalu banyak variabel global dapat menyebabkan program sulit di-debug (cari kesalahan) dan memerlukan memory yang lebih besar.

5.3 Parameter

Ketika suatu prosedur dipanggil maka pada hakekatnya bisa dilakukan pertukaran data antara program utama dan prosedur. Pertukaran ini dilakukan melalui parameter.

Parameter Aktual adalah parameter yang disertakan pada saat prosedur dipanggil untuk dilaksanakan, sering disebut sebagai *argumen.*

Parameter Formal adalah parameter yang dituliskan pada definisi suatu prosedur / fungsi. Ada tiga jenis parameter formal, yaitu:

1. ***parameter masukan (input)*** : parameter yang menerima nilai dari parameter aktual.
2. ***parameter keluaran (output)*** : parameter yang menyerahkan nilai ke parameter aktual.
3. ***parameter masukan dan keluaran (input-****output)*** : parameter yang menerima nilai dari parameter aktual untuk diproses dalam prosedur kemudian diserahkan kembali ke parameter aktual setelah selesai.

Contoh algoritma berikut ini menunjukkan pemakaian parameter masukan dan parameter keluaran untuk prosedur menghitung luas segitiga.

Algoritma Luas_segitiga
{ menghitung luas segitiga dengan menggunakan prosedur yang memanfaatkan parameter input dan parameter output **}**

Deklarasi
real alas, tinggi, luas;
prosedur Hit_Luas_segi_3 (input real a, t; **output real** ls; **);**

Deskripsi
write ("Masukkan alas segitiga : " **);**
read (alas **);**
write ("Masukkan tinggi-nya : " **);**
read (tinggi **);**

Hit_Luas_segi_3 (alas, tinggi, luas **);**
write ("Luas segitiga = ", luas **);**

prosedur Hit_Luas_segi_3 (input real a, t; **output real** ls; **)**
{ prosedur menghitung luas segi_3, menerima a (alas) dan t (tinggi), mengembalikan ls (luas) **}**

Deklarasi { }

Deskripsi
ls ← a * t / 2.0;

Contoh algoritma berikut ini menunjukkan pemakaian parameter input/output yang digunakan untuk prosedur yang melakukan pertukaran nilai variabel.

Algoritma Tukar_nilai
{ menukar nilai dua variabel yang dilakukan oleh suatu prosedur dengan parameter input/output **}**

Deklarasi
integer A, B;
prosedur Tukar (in-out integer a, b **);**

Deskripsi
write ("Masukkan nilai A : " **);**
read (A **);**
write ("Masukkan nilai B : " **);**

```
read ( B );
Tukar ( A, B );
write ( "Setelah ditukar : " );
write ( " A = ", A, " B = ", B );
```

prosedur Tukar (in-out integer a, b **)**
{ prosedur yang melaksanakan pertukaran nilai **}**

Deklarasi

```
integer temp;
```

Deskripsi

```
temp ← a;
a ← b;
b ← temp;
```

5.4 Fungsi

Fungsi pada hakekatnya serupa dengan prosedur tetapi harus mengembalikan nilai. Prosedur hanya bisa mengembalikan nilai melalui parameter input/output.

Bentuk Umum:

```
Fungsi nama_fungsi ( parameter formal ) → tipe_hasil
{ spesifikasi fungsi }

Deklarasi
    { variabel lokal }

Deskripsi
    { langkah / proses yang dilakukan oleh fungsi }
    ..............
    return hasil.
```

Contoh berikut ini adalah contoh yang melaksanakan fungsi matematis
$f(x) = x^2 + 8x + 10$.

```
Fungsi F ( input real x ) → real
{ menghitung nilai fungsi f(x) = x^2 + 8x + 10 }

Deklarasi
        real y;

Deskripsi
        y ← x * x + 8 * x + 10;
        return y;
```

Contoh berikut ini adalah pemakaian fungsi untuk mengganti bulan dalam angka (mis. 3) menjadi nama bulan (mis. Maret).

```
Algoritma Tanggal_Lahir
{ algoritma ini memanggil fungsi untuk menampilkan nama bulan }

Deklarasi
        integer tanggal, bulan, tahun;
        string nama_bulan;
        fungsi Nama_bulan ( input integer bln ) → string;

Deskripsi
        write ( "tanggal : " ); read ( tanggal );
        write ( "bulan : " );   read ( bulan );
        write ( "tahun : " );   read ( tahun );

        nama_bulan ← Nama_bulan ( bulan );
        write ( tanggal, ' – ', nama_bulan, ' – ', tahun );

fungsi Nama_bulan ( input integer bln ) → string
{ mengembalikan nama bulan berdasarkan angka bulan }

Deklarasi
        string nama_bln;

Deskripsi
        case ( bln )
                1 : nama_bln ← "Januari";
                2 : nama_bln ← "Februari";
                3 : nama_bln ← "Maret";

                ...
                11 : nama_bln ← "Nopember";
                12 : nama_bln ← "Desember";
        endcase.
        return nama_bln;
```

Contoh berikut ini menunjukkan pemakaian fungsi untuk menampilkan angka bulat (maksimum 4 digit) dalam bentuk kalimat, misal:

input : 2436
output : dua ribu empat ratus tiga puluh enam

Algoritma Angka_dalam_kalimat
{ menterjemahkan angka (max. 4 digit) kedalam kalimat **}**

Deklarasi

integer angka, sisa, d1, d2, d3, d4;
string angka1, angka2, angka3, angka4;
fungsi digit (input integer d **)** → **string;**

Deskripsi

write ("Masukkan sebuah angka (maks 4 digit) " **);**
read (angka **);**

{ memisahkan digit angka dalam urutan d4 d3 d3 d1 **}**
d4 ← angka \ 1000;
sisa ← angka % 1000;
d3 ← sisa \ 100;
sisa ← sisa % 100;
d2 ← sisa \ 10;
sisa ← sisa % 10;
d1 ← sisa;

{ menterjemahkan digit **}**
if (d4 > 1 **)**
then angka4 ← **digit (** d4 **) +** "ribu";
else if (d4 = 1 **)**
then angka4 ← "seribu";
else angka4 ← ' ';
endif.
endif.

if (d3 > 1 **)**
then angka3 ← **digit (** d3 **) +** "ratus";
else if (d3 = 1 **)**
then angka3 ← "seratus";
else angka3 ← ' ';
endif.
endif.

```
        if ( d2 > 1 )
            then angka2 ← digit ( d2 ) + "puluh";
                 if ( d1 = 0 )
                     then angka1 ← ' ';
                     else angka1 ← digit ( d1 ) ;
                 endif.
            else if ( d2 = 1 )
                     then if ( d1 = 0 )
                              then angka2 ← "sepuluh";
                                   angka1 ← ' ';
                              else if ( d1 = 1 )
                                       then angka2 ← "sebelas";
                                            angka1 ← ' ';
                                       else angka2 ← digit ( d1 ) + "belas";
                                            angka1 ← ' ';
                                   endif.
                          endif.
                     else angka2 ← ' ';
                          if ( d1 = 0 )
                              then angka1 ← ' ';
                              else angka1 ← digit ( d1 );
                          endif.
                 endif.
        endif.
        write ( angka, ' = ' , angka4 + angka3 + angka2 + angka1 );

fungsi digit ( input integer d ) → string
{ menterjemahkan digit ke dalam satu kata }

Deklarasi
        string kata;

Deskripsi
        case ( d )
             1 : kata ← "satu";
             2 : kata ← "dua";
             3 : kata ← "tiga";
             4 : kata ← "empat";
             5 : kata ← "lima";
             6 : kata ← "enam";
             7 : kata ← "tujuh";
             8 : kata ← "delapan";
             9 : kata ← "sembilan";
        endcase.
        return kata;
```

6. PEMROSESAN TEKS

Defenisi : ***Teks (text)*** adalah deretan karakter yang bisa direkam ke dalam suatu *file / berkas / arsip*.

- Suatu teks bisa terdiri atas beberapa *kata (words).*
- Setiap kata terpisah dari kata lainnya, dipisahkan oleh paling sedikit satu *spasi.*
- Suatu teks dapat terdiri atas beberapa *baris (lines)* yang dibentuk oleh beberapa kata.
- Setiap baris diakhiri oleh marka *end-of-line (EOL).*
- Suatu teks diawali oleh marka *begin-of-file (BOF)* dan diakhiri dengan marka *end-of-file (EOF).*

Deklarasi suatu teks didahului dengan tipe-data **text.**

Beberapa instruksi yang berkaitan dengan teks adalah sbb:

a. mengembalikan penunjuk teks ke posisi awal (BOF) : **reset (F)**
b. membuka suatu file teks yang ada di storage (disk) : **assign (F,** nama_file **)**
c. membaca satu karakter dari file teks : **read (F,** char **)**
d. merekam satu karakter ke file teks : **write (F,** char **)**
e. menutup file teks : **close (F)**

Contoh 1 : menghitung jumlah karakter (tidak termasuk marka EOL, BOF, dan EOF) yang ada dalam suatu file teks bernama : mytext.txt

Algoritma Hit_karakter
{ menghitung jumlah karakter yang ada dalam suatu file teks **}**

Deklarasi

```
text F;
char k;
integer jkar;
```

Deskripsi

```
assign ( F, "mytext.txt" );
{ kembalikan ke awal file }
read ( F, k );
if ( k != BOF )
    then reset ( F );
         read ( F, k );
endif.

{ baca hingga akhir file }
read ( F, k );
```

```
jkar ← 0;
while ( k != EOF ) do
        if ( k != EOL )
            then jkar ← jkar + 1;
        endif.
        read ( F, k );
endwhile.

write ( "jumlah karakter (termasuk spasi ) = ", jkar );
```

Contoh 2 : menghitung jumlah baris yang ada dalam suatu file teks.

```
Algoritma Hit_b aris
{ menghitung jumlah baris pada suatu file teks }

Deklarasi
        text F;
        char k;
        integer jbar;

Deskripsi
        assign ( F, "mytext.txt" );
        { kembalikan ke awal file }
        read ( F, k );
        if ( k != BOF )
            then reset ( F );
                 read ( F, k );
        endif.

        { baca hingga akhir file }
        read ( F, k );
        jbar ← 0;
        while ( k != EOF ) do
                if ( k = EOL )
                    then jbar ← jbar + 1;
                endif.
                read ( F, k );
        endwhile.

        write ( "jumlah baris = ", jbar );
```

Contoh 3 : menghitung jumlah kata yang ada dalam suatu file teks.

Algoritma Hit_kata
{ menghitung jumlah kata dalam suatu file teks **}**

Deklarasi

```
text F;
char k1, k2;
integer jkata;
```

Deskripsi

```
assign ( F, "mytext.txt" );
{ kembalikan ke awal file }
read ( F, k1 );
if ( k1 != BOF )
    then reset ( F );
         read ( F, k1 );
endif.

{ baca hingga akhir file }
read ( F, k1 );
read ( F, k2 );
jkata ← 0;
while ( k1 != EOF ) || ( k2 != EOF ) do
        if ( k1 != ' ' ) && ( k2 = ' ' )
            then jkata ← jkata + 1;
        endif.
        k1 ← k2;
        read ( F, k2 );
endwhile.
if ( k1 != ' ' ) && ( k2 = EOF )
    then jkata ← jkata + 1;
endif.
write ( "jumlah kata = ", jkata );
```

7. LARIK / ARRAY

. ***Larik (Array)*** adalah suatu bentuk struktur data yang menampung satu atau lebih dari satu data yang sejenis (bertipe data sama), yang diwakili oleh satu nama variabel.

- Setiap elemen atau anggota larik dapat dikenali atau diakses melalui suatu indeks.
- Larik berdimensi satu disebut ***vektor.***
- Larik berdimensi dua disebut ***matriks.***
- Larik berdimensi lebih dari dua disebut ***tensor.***

Mendefinisikan Larik :

a. nama_array : array [1..n] of tipe_data;
contoh : **A : array [1..10] of integer;**

b. tipe_data nama_array [n];
contoh : **integer A[10];**

c. type larik : array [1..n] of tipe_data;
nama_array : larik;
contoh : **type larik : array [1..10] of integer;**
A : larik;

Operasi Larik :

- membaca / mengisi larik
- mencetak / menampilkan larik
- menggeser isi larik
- menggabungkan beberapa larik
- menguraikan satu larik
- mengurutkan isi larik
- mencari elemen dalam larik

7.1 Membaca / mengisi Larik

Proses membaca atau mengisi suatu larik, dimulai dengan mendefinisikan array disertai dengan jumlah elemen yang akan disimpan, kemudian dengan memakai instruksi *perulangan* satu persatu elemen diisi dengan indeks yang berurutan mulai dari 1 hingga indeks maksimum.

Berikut ini disajikan dua algoritma untuk mengisi suatu larik. Algoritma yang pertama tidak menggunakan prosedur, algoritma yang kedua menggunakan pro sedur.

Algoritma IsiLarik_1
{ membaca atau mengisi larik tanpa menggunakan prosedur **}**

Deklarasi
const N = 10;
integer A[N];
integer indeks;

Deskripsi
for (indeks = 1 **to** N **step** 1**)**
write ("Masukkan elemen ke-", indeks**)**
read (A[indeks] **);**
endfor.

Algoritma IsiLarik_2
{ membaca atau mengisi larik dengan menggunakan prosedur **}**

Deklarasi
const N=100;
integer A[N];
integer K;
prosedur Baca_Larik (input integer M**, output integer** A[ ] **);**

Deskpripsi
write ("Masukkan Jumlah Elemen Larik (< 100) : "**);**
read (K **);**
Baca_Larik (K, A**);**

prosedur Baca_Larik (input integer M, **output integer** A[ ] **)**
{ prosedur membaca / mengisi larik **}**

Deklarasi
integer indeks;

Deskripsi
for (indeks = 1 **to** M **step** 1 **)**
write ("Masukkan elemen ke-", indeks **);**
read (A[indeks] **);**
endfor.

7.2 Menampilkan Isi Larik

Berikut ini disajikan sebuah prosedur untuk menampilkan isi suatu larik dengan M buah elemen. Prosedur ini dapat dipanggil oleh algoritma yang memerlukan prosedur untuk menampilkan sebuah larik.

prosedur Cetak_Larik (input integer M, **input integer** A[ ] **)**
{ prosedur untuk menampilkan isi suatu larik atau array **}**

Deklarasi
    integer indeks;

Deskripsi
    for (indeks = 1 **to** M **step** 1 **)**
        write (A[indeks] **);**
        endfor.

7.3 Menggeser Isi Larik

Beberapa aplikasi memerlukan pergeseran isi larik misalnya menggeser ke kiri atau menggeser ke kanan yang digambarkan sebagai berikut.

Larik Asli
5 4 6 8 3 2

Setelah Geser Kanan
2 5 4 6 8 3

4 6 8 3 2 5 Setelah Geser Kiri

Proses Geser Kanan berarti elemen ber-indeks i digeser ke posisi ber-indeks i+1, dengan catatan elemen terakhir akan dipindahkan ke posisi pertama. Sebaliknya proses Geser Kiri berarti elemen ber-indeks i digeser ke posisi ber-indeks i-1, dengan menggeser elemen pertama ke posisi terakhir.

prosedur Geser_Kanan (in-out integer A[ ], **input integer** M **)**
{ menggeser elemen suatu larik (vektor) ke kanan, A[i+1] ↤ A[i] **}**

Deklarasi
integer indeks, temp;

Deskripsi
temp ↤ A[M];
for (indeks = M- 1 **to** 1 **step** -1 **)**
A[indeks + 1] ↤ A[indeks];
endfor.
A[1] ↤ temp;

prosedur Geser_Kiri (in-out integer A[ ], **input integer** M **)**
{ menggeser elemen suatu larik (vektor) ke kiri, A[i-1] ↤ A[i] **}**

Deklarasi
integer indeks, temp;

Deskripsi
temp ↤ A[1];
for (indeks = 2 **to** M **step** 1 **)**
A[indeks-1] ↤ A[indeks];
endfor.
A[M] ↤ temp;

Suatu algoritma untuk menggeser elemen larik dengan memanfaatkan prosedur-prosedur yang telah digunakan diatas diberikan berikut ini.

Algoritma Geser_Larik
{ algoritma untuk menggeser isi larik dengan memanfaatkan prosedur prosedur larik **}**

Deklarasi
const Nmax = 100;
integer N, pilihan, A[Nmax];
prosedur Baca_Larik (input integer M, **input integer** A[ ] **);**
prosedur Cetak_Larik (input integer M, **input integer** A[ ] **);**
prosedur Geser_Kanan (input integer A[ ], **input integer** M **);**
prosedur Geser_Kiri (input integer A[ ], **input integer** M **);**

Deskripsi
write ("Masukkan jumlah elemen larik : " **);**
read (N **);**
Baca_Larik (N, A **);**

```
write ( "Pilih salah satu : " );
write ( " 1. Geser Kanan " );
write ( " 2. Geser Kiri " );
read ( pilihan );
if ( pilihan = 1 )
     then Geser_Kanan ( A, N );
   else Geser_Kiri ( A, N );
   endif.

Cetak_Larik ( N, A );
```

7.4 Menggabung (merge) Larik

Beberapa Larik dapat digabungkan menjadi satu Larik yang lebih besar. Misalkan Larik A dengan 10 elemen akan digabungkan dengan Larik B dengan 15 elemen, tentu saja gabungannya berupa larik C harus memiliki elemen yang sama atau lebih besar dari 25.

Berikut ini adalah sebuah prosedur yang menggabungkan Larik A, N buah elemen dengan Larik B, M buah elemen, menjadi Larik C dengan L elemen dimana L = N + M.

```
prosedur Gabung_Larik ( input integer A[ ], input integer N, input integer B[ ],
            input integer M, output integer C[ ], output integer L ) {
            menggabungkan dua larik menjadi larik yang lebih besar }

Deklarasi
     integer indeks;

Deskripsi
     L ← N + M;
     { salin isi A ke dalam C }
     for ( indeks = 1 to N step 1 )
          C[indeks] ← A[indeks];
     endfor.
     { salin isi B ke dalam C }
     for ( indeks = N+1 to L step 1 )
          C[indeks] ← B[indeks – N];
     endfor.
```

7.5 Memisah (split) Larik

Sebuah Larik dapat dipisahkan menjadi beberapa Larik yang lebih kecil. Misalkan satu Larik C dengan 25 elemen dapat dipisahkan menjadi Larik A dengan 10 elemen dan Larik B dengan 15 elemen.

Berikut ini adalah sebuah prosedur yang memisahkan larik C dengan L elemen, menjadi larik A dengan N elemen dan larik B dengan M elemen.

```
prosedur Pisah_Larik ( output integer A[ ], input integer N, output integer B[ ],
                       input integer M, input integer C[ ], input integer L )
{ memisahkan sebuhlarik menjadi dua larik yang lebih kecil }

Deklarasi
    integer indeks;

Deskripsi
    { salin isi C ke dalam A }
    for ( indeks = 1 to N step 1 )
        A[indeks] ← C[indeks];
    endfor.
    { salin sisanya ke B }
    for ( indeks = N+1 to L step 1 )
        B[indeks - N] ← C[indeks];
    endfor.
```

7.6 Mengurutkan (Sort) Isi Larik

Beberapa aplikasi memerlukan data yang ber-urut baik dari kecil ke besar (ascending) maupun dari besar ke kecil (descending). Pada bagian ini akan digunakan teknik sort yang sederhana yang disebut metoda gelembung (bubble sort), pada bagian lain nanti akan dibahas berbagai teknik sort yang lebih rumit.

Prinsip dari "bubble sort" adalah sebagai berikut:

- andaikan ada 5 elemen dalam larik [10, 8, 3, 5, 4]
- mula-mula ambil indeks 1 sebagai patokan A[1]=10
- bandingkan isi A[1] dengan A[2], A[3], A[4] dan A[5]
- bila A[1] > A[2] maka tukar tempat sehingga A[1] = 8 [8, 10, 3, 5, 4]
- bila A[1] > A[3] maka tukar tempat sehingga A[1] = 3 [3, 10, 8, 5, 4]
- bila A[1] > A[4] maka tukar tempat, tidak terjadi
- bila A[1] > A[5] maka tukar tempat, tidak terjadi
- Sekarang ambil indeks 2 sebagai patokan A[2] = 10
- bandingkan A[2] dengan A[3], A[4] dan A[5]
- hasilnya adalah [3, 4, 10, 8, 5]
- Sekarang ambil indeks 3 sebagai patokan A[3] = 10
- bandingkan A[3] dengan A[4] dan A[5]
- hasilnya adalah [3, 4, 5, 10, 8]
- Sekarang ambil indeks 4 sebagai patokan A[4] = 10

. bandingkan A[4] dengan A[5]
- hasilnya adalah [3, 4, 5, 8, 10]
- dengan demikian proses pengurutan selesai

Pada proses pengurutan yang dijelaskan diatas secara algoritma memerlukan dua buah indeks, indeks pertama sebagai patokan yang bergerak dari 1 hingga N-1 (1 sampai 4 pada contoh diatas), dan indeks kedua sebagai pembanding yang selalu bergerak dari posisi indeks patokan + 1 hingga posisi terakhir N (mulai dari 2 sampai 5 pada contoh diatas).

```
prosedur Sort_Larik ( in-out integer A[ ], input integer N )
{ mengurutkan isi larik secara ascending dengan metoda bubble sort }

Deklarasi
    integer idx1, idx2, temp;

Deskripsi
    for ( idx1 = 1 to N-1 step 1 )
        for ( idx2 = idx1 + 1 to N step 1 )
            { bila A[idx1] > A[idx2] tukar }
            if ( A[idx1] > A[idx2] )
                then temp ← A[idx1];
                     A[idx1] ← A[idx2];
                     A[idx2] ← temp;
            endif.
        endfor.
    endfor.
```

7.7 Mencari (Search) elemen Larik

Mencari suatu elemen dalan larik merupakan suatu proses dasar yang sangat penting, aplikasi lanjut dari proses ini banyak ditemukan pada sistem basis data untuk menemukan suatu rekaman data, atau pada pemroses teks (wordprocessing) dalam mencari kata (find) atau mengganti kata (replace).

Hasil pencarian yang diharapkan antara lain:
- indikator ‘Y’ bila ditemukan atau ‘N’ bila tidak ditemukan
- posisi dalam larik dimana elemen tersebut ditemukan

Berikut ini disajikan sebuah prosedur untuk mencari satu elemen didalam larik dengan jumlah elemen sebanyak N buah.

procedur Cari_elemen (input integer A[ ], **input integer** N, **input integer** x
output char indikator, **output integer** posisi **)**
{ suatu prosedur untuk mencari elemen x didalam larik A, dengan indikator dan posisi sebagai hasilnya **}**

Deklarasi
integer indeks;

Deskripsi
indikator ← 'N';
posisi ← 0;
indeks ← 1;
while (indeks < N+1 **&&** indikator = 'N' **) do**
if (A[indeks] = x **)**
then posisi ← indeks;
indikator ← 'Y';
endif.
indeks ← indeks + 1;
endwhile.

7.8 Matriks / Larik Dua Dimensi

Salah satu struktur data larik yang juga banyak digunakan dalam berbagai aplikasi adalah *matriks* atau larik 2D (dua dimensi), satu untuk menunjukkan *baris* dan yang lainnya menunjukkan *kolom*. Susunan angka berikut ini menunjukkan matriks 4 x 5 (4 baris dan 5 kolom).

10	**12**	**7**	**9**	**16**
8	**15**	**10**	**11**	**25**
13	**8**	**34**	**23**	**7**
45	**27**	**6**	**5**	**17**

Mengisi suatu matriks berdimensi 4 x 5 dilakukan baris demi baris, mulai dari baris 1 dengan mengisi kolom 1 sampai dengan kolom 5, kemudian pindah ke baris 2 dan mengisi kolom 1 sampai dengan kolom 5, dst.

Algoritma Isi_Matriks_4x5
{ algoritma mengisi suatu matriks 4 x 5 **}**

Deklarasi
const baris=4, kolom=5;
integer brs, kol;
integer A[baris] [kolom]**;**

Deskripsi

```
for ( brs=1 to baris step 1 )
    for ( kol=1 to kolom step 1 )
        write ( "elemen baris-", brs, "kolom-", kol );
        read ( A[brs][kol] );
    endfor.
endfor.
```

Menampilkan isi dari matriks 4x5 diatas adalah sebagai berikut.

Algoritma Tampilkan_Isi_Matriks
{ algoritma menampilkan isi matriks 4 x 5 **}**

Deklarasi

```
const baris=4, kolom=5;
integer brs, kol;
integer A[baris] [kolom];
```

Deskripsi

```
for ( brs=1 to baris step 1 )
    for ( kol=1 to kolom step 1 )
        write ( A[brs] [kol] );
        endfor.
    write ( );
endfor.
```

Prosedur untuk mengisi dan menampilkan elemen-elemen matriks berdimensi N x M disajikan berikut ini.

prosedur Isi_Matriks (input integer N, **input integer** M, **output integer** A[ ] **)**
{ prosedur untuk mengisi matriks berdimensi N x M **}**

Deklarasi

```
integer brs, kol;
```

Deskripsi

```
for ( brs=1 to N step 1 )
    for ( kol=1 to M step 1 )
        write ( "elemen baris-", brs, "kolom-", kol );
        read ( A[brs][kol] );
    endfor.
endfor.
```

```
prosedur Tampil_Matriks ( input integer N, input integer M, input integer A[ ] )
{ prosedur untuk menampilkan isi matriks berdimensi N x M }

Deklarasi
        integer brs, kol;

Deskripsi
        for ( brs=1 to N step 1 )
                for ( kol=1 to M step 1 )
                    write ( A[brs] [kol] );
                    endfor.
        endfor.
```

8. TEKNIK PENCARIAN (SEARCHING)

Pencarian elemen (searching) merupakan proses fundamental dalam pemrograman, berbagai proses dalam aplikasi memerlukan searching, antara lain:

- proses editing (perbaikan dan peremajaan data) selalu didahului dengan pencarian akan posisi data yang akan di-edit.
- proses inserting (penyisipan data) memerlukan searching posisi dimana suatu data akan di-sisipkan.
- program pengolah kata (word atau text processing) menyediakan fasilitas *Find, Replace,* dan *Goto* yang pada hakekatnya memerlukan teknik pencarian.

Definisi persoalan pencarian elemen dalam larik adalah sbb:

- Diberikan sebuah larik A yang elemen-elemennya sudah ditetapkan, kemudian ada sebuah elemen x yang tipe-nya sama dengan elemen larik A, maka akan dicari apakah ada elemen dalam A yang sama nilainya dengan x, bila ada maka tampilkan bahwa "x ditemukan", bila tidak mana tampilkan "x tidak ditemukan".
- Diberikan sebuah larik A yang sudah terisi penuh, kemudian ada sebuah elemen x yang tipe-nya sama dengan elemen larik A, maka akan dicari apakah ada elemen dalam larik A yang sama nilainyan dengan x, bila ada maka tentukan posisinya (indeks) dari elemen tersebut, bila tidak ada maka nyatakan indeks=0.
- Diberikan sebuah larik A yang sudah terisi penuh, kemudian ada sebuah elemen x yang tipe-nya sama dengan elemen larik A, maka akan dicari apakah ada elemen dalam larik A yang sama nilainyan dengan x, bila ada maka berikan nilai *true* kepada suatu variabel logis (boolean), bila tidak ada maka berikan nilai *false* kepada variabel tersebut.

Beberapa metoda pencarian yang akan dibicarakan pada bagian ini adalah:

- pencarian secara beruntun (sequential atau linier search)
- pencarian dengan teknik sentinel
- pencarian bagidua (binary search)

8.1 Pencarian Secara Beruntun

Pencarian secara beruntun dilakukan dengan cara memeriksa elemen larik satu persatu mulai dari indeks=1 hingga indeks dimana elemen tersebut ditemukan, bilamana indeks maksimum telah dilampaui maka berarti elemen tersebut tidak ditemukan.

Contoh: Andaikan larik A = [20 15 18 35 40 27], dan x = 18
bila A diperiksa mulai dari indeks=1 maka akan ditemukan pada indeks=3.
Tetapi bila x=45, maka pemeriksaan tidak berhasil menemukan elemen, karena hingga indeks=6 elemen ini tetap tidak ditemukan.

Algoritma Sequential_V.1
{ algoritma pencarian beruntun dimana elemen-elemen A tidak sorted dan
akan ditampilkan kalimat bila elemen ditemukan **}**

Deklarasi
integer m = 10;
integer A[m], x;
integer indeks;

Deskripsi
{ baca elemen matriks A **}**
for (indeks= 1 **to** m **step** 1**)**
write ("masukkan elemen ke-", indeks **);**
read (A[indeks] **);**
endfor.

{ baca elemen x **}**
write ("masukkan elemen x : " **);**
read (x **);**

{ mencari x dalam A **}**
indeks ← 1;
while (indeks < = m **&&** x != A[indeks] **) do**
indeks ← indeks + 1;
endwhile.

{ menetapkan hasilnya **}**
if (x = A[indeks] **)**
then write ("x ditemukan " **);**
else write ("x tidak ditemukan " **);**
endif.

Algoritma Sequential_V.2
{ algoritma pencarian beruntun dimana elemen-elemen A tidak sorted dan
akan ditampilkan posisi atau indeks bila elemen ditemukan **}**

Deklarasi
integer m = 10;
integer A[m], x;
integer indeks, posisi;

Deskripsi
{ baca elemen matriks A **}**
for (indeks= 1 **to** m **step** 1**)**
write ("masukkan elemen ke-", indeks **);**
read (A[indeks] **);**
endfor.

{ baca elemen x **}**
write ("masukkan elemen x : " **);**
read (x **);**

```
{ mencari x dalam A }
indeks ← 1;
while ( indeks < = m && x != A[indeks] ) do
        indeks ← indeks + 1;
endwhile.

{ menetapkan hasilnya }
posisi ← 0;
if ( x = A[indeks] )
    then posisi ← indeks;
endif.

write ( "posisi = ", posisi );
```

Algoritma Sequential_V.3

{ algoritma pencarian beruntun dimana elemen-elemen A tidak sorted dan akan ditetapkan variabel boolean apakah true atau false **}**

Deklarasi

```
integer  m  =  10;
integer  A[m],  x;
integer    indeks;
boolean ketemu;
```

Deskripsi

```
{ baca elemen matriks A }
for ( indeks= 1 to m step 1)
      write ( "masukkan elemen ke-", indeks );
      read ( A[indeks] );
endfor.

{ baca elemen x }
write ( "masukkan elemen x : " );
read ( x );

{ mencari x dalam A }
indeks ← 1;
while ( indeks < = m && x != A[indeks] ) do
        indeks ← indeks + 1;
endwhile.

{ menetapkan hasilnya }
ketemu ← false;
if ( x = A[indeks] )
    then ketemu ← true;
endif.

write ( ketemu );
```

Adakalanya elemen-elemen dalam larik A sudah di-urutkan (sorted), maka akan terdapat sedikit perbedaan dalam proses pencariannya.

Algoritma Sequential_V.4
{ algoritma pencarian beruntun dimana elemen-elemen A sorted dan
akan ditampilkan kalimat bila elemen ditemukan **}**

Deklarasi
integer m = 10;
integer A[m], x;
integer indeks;

Deskripsi
{ baca elemen matriks A **}**
for (indeks= 1 **to** m **step** 1**)**
write ("masukkan elemen ke-", indeks **);**
read (A[indeks] **);**
endfor.

{ baca elemen x **}**
write ("masukkan elemen x : " **);**
read (x **);**

{ mencari x dalam A **}**
indeks ← 1;
while (indeks < = m **&&** x < A[indeks] **) do**
indeks ← indeks + 1;
endwhile.

{ menetapkan hasilnya **}**
if (x = A[indeks] **)**
then write ("x ditemukan " **);**
else write ("x tidak ditemukan " **);**
endif.

8.2 Pencarian dengan Sentinel

Sentinel pada hakekatnya adalah elemen fiktif yang ditambahkan ke dalam suatu larik pada posisi terakhir yang nilainya sama dengan nilai dari elemen yang dicari. Pada proses pencarian ada dua kemungkinan yaitu:

- elemen ditemukan pada posisi indeks antara 1 sampai dengan m (indeks maksimum), berarti elemen yang dicari benar ada di dalam larik,
- atau elemen ditemukan pada posisi indeks (m+1) atau elemen sentinel, berarti sebenarnya elemen yang dicari tidak ada di dalam larik.

```
Algoritma Sentinel_Search
{ melakukan pencarian dengan menambahkan elemen sentinel }

Deklarasi
        integer m = 10;
        integer A[m+1], x;
        integer indeks;

Deskripsi
        { menambahkan sentinel }
        A[m+1] ← x;

        { mulai pencarian }
        indeks ← 1;
        while ( x != A[indeks] ) do
            indeks ← indeks + 1;
            endwhile.

        { menetapkan hasil }
        if ( indeks < m+1 )
            then write ( " x ditemukan pada posisi ", indeks );
          else write ( " x tidak ditemukan dalam larik " );
          endif.
```

8.3 Pencarian Bagi Dua (Binary Search)

Pencarian bagi-dua adalah teknik yang diterapkan hanya pada elemen larik yang telah terurut (sorted). Pencarian beruntun memiliki satu kekurangan yaitu dalam kasus terburuk (elemen yang dicari berada pada posisi terakhir) maka pencarian harus dilakukan sepanjang larik, semakin banyak elemen maka semakin lama pencarian harus dilakukan.

Proses pencarian bagi dua dilakukan sebagai berikut:

- andaikan jumlah elemen adalah m, maka tetapkan indeks= m/2, sehingga larik terbagi dua, yaitu bagian kiri dengan indeks dari 1 sampai m/2, dan bagian kanan dengan indeks m/2 hingga m.
- Periksa dulu apakah x = A[indeks], bila ya berarti elemen ditemukan, bila tidak teruskan ke langkah berikutnya.
- Periksa apakah x > A[indeks], bila ya maka cari disisi kanan, bila tidak maka cari disisi kiri.
- Teruskan pencarian pada sisi yang tepat dengan mengambil indeks tengah dari sisi tersebut, sampai elemen ditemukan atau tidak sama sekali.

Contoh: andaikan larik A = [7 10 13 16 18 21 76 81], dan x = 10.
pada contoh ini jumlah elemen m = 8, sehingga indeks = 8/2 = 4, sehingga A[4] = 16. Tetapi x tidak sama dengan A[4].
Periksa apakah x > A[4], atau x > 16?, jawabannya tidak, periksa sisi kiri.
Pada sisi kiri indeks = (1 + 4) / 2 = 2, sehingga A[2] = 10.
Ternyata x=A[2], sehingga elemen ditemukan dalam dua langkah.

Algoritma Binary_Search
{ pencarian elemen dengan metoda bagi dua **}**

Deklarasi
integer m=10;
integer A[m], x;
integer idx1, idx2, indeks;
boolean ketemu;

Deskripsi
idx1 ← 1;
idx2 ← m;
ketemu ← **false;**
while (!ketemu **&&** (idx1 < idx2) **) do**
{ menghitung titik tengah }
indeks ← (idx1 + idx2) \ 2;
if (x = A[indeks] **)**
then ketemu ← **true;**
else if (x > A[indeks] **)**
then idx1 ← indeks + 1; { sisi kanan }
else idx2 ← indeks – 1; { sisi kiri }
endif.
endif.
endwhile.
if (ketemu **)**
then write (" x ketemu di posisi : ", indeks **);**
else write (" x tidak ditemukan " **);**
endif.

9. TEKNIK PENGURUTAN (SORTING)

Pengurutan adalah proses yang mengatur sekumpulan objek sehingga nilainya tersusun apakah berurut menaik (ascending) dari kecil ke besar, atau berurut menurun (descending) dari besar ke kecil.

Contoh:

[23 30 45 52 67] data integer ascending

[67 48 50 25 17] data integer descending

[Amir Badu Charles Daud ...] data string ascending

<991023, Eko, A> <991055, David, B> <991075, Abdu, C> .... record mahasiswa ascending.

Ada dua kategori pengurutan yaitu:

- pengurutan internal – pengurutan yang dilaksanakan hanya dengan menggunakan memori komputer, pada umumnya bila jumlah elemen tidak terlalu banyak
- pengurutan eksternal – pengurutan yang dilaksanakan dengan bantuan virtual memori atau harddisk karena jumlah elemen yang akan diurutkan terlalu banyak.

Teknik pengurutan cukup banyak, namun yang akan dibahas hanya tiga macam yaitu:

- teknik Gravitasi
- teknik Minimum dan Maximum (MiniMax)
- teknik Penyisipan (Insertion)

9.1 Teknik Gravitasi

Teknik Gravitasi adalah suatu teknik yang merupakan variasi dari teknik gelembung (Bubble Sort) yang pernah dibahas pada bagian yang terdahulu. Teknik ini memanfaatkan sifat gravitasi dimana yang berat (nilainya besar) akan turun ke bawah.

- Mulai dari indeks=1 bandingkan A[1] dengan A[2], bila A[1] lebih berat maka adakan penukaran tempat.
- Selanjutnya indeks=2 dan bandingkan A[2] dengan A[3], bila A[2] lebih berat adakan penukaran tempat.
- Lakukan seterusnya, sehingga semua nilai yang berat turun ke bawah.

Sebagai contoh andaikan A = [25 27 10 8 76 21]

Putaran 1 : [25 27 10 8 76 21] menjadi [25 10 27 8 76 21]

[25 10 27 8 76 21] menjadi [25 10 8 27 76 21]

[25 10 8 27 76 21] menjadi [25 10 8 27 21 76]

Putaran 2 : [25 10 8 27 21 76] menjadi [10 25 8 27 21 76]
[10 25 8 27 21 76] menjadi [10 8 25 27 21 76]
[10 8 25 27 21 76] menjadi [10 8 25 21 27 76]

Putaran 3 : [10 8 25 21 27 76] menjadi [8 10 25 21 27 76]
[8 10 25 21 27 76] menjadi [8 10 21 25 27 76]

Putaran 4 : [8 10 21 25 27 76] larik A sudah berurut ascending.

Apabila larik A memiliki m buah elemen, dimana setiap kali ada dua elemen yang dibandingkan maka pada hakekatnya jumlah putaran maksimum adalah (m – 1) kali. Pada setiap putaran elemen terberat akan menempati posisi terbawah sehingga pada putaran berikutnya jumlah elemen yang harus dibandingkan selalu berkurang satu. Prosedurnya sebagai berikut:

```
Prosedur Teknik_Gravitasi ( in-out integer A[ ], input integer m )
{ mengurutkan data dengan teknik gravitasi }

Deklarasi
    integer putaran, indeks, jum_elemen, temp;

Deskripsi
    maks ← m;
    for ( putaran = 1 to m-1 step 1 )
        for ( indeks = 1 to maks – 1 step 1 )
            if ( A[indeks] > A[indeks + 1] )
                then temp ← A[indeks];
                     A[indeks] ← A[indeks + 1];
                     A[indeks + 1] ← temp;
            endif.
        endfor.
        maks ← maks – 1;
    endfor.
```

9.2 Teknik MiniMax

Teknik MiniMax merupakan suatu prosedur yang memilih elemen terkecil (minimum) atau yang terbesar (maximum) untuk ditempatkan pada posisi yang sesuai dengan tujuan pengurutan. Andaikan suatu larik akan diurutkan ascending maka prosedurnya sbb:

- Mula-mula ambillan elemen terakhir A[m], kemudian carilah elemen terbesar diantara elemen-elemen lainnya (A[1] sampai A[m-1]).
- Andaikan elemen pada posisi x adalah yang terbesar, bandingkan A[x] dengan A[m], bila A[x] > A[m] maka lakukan penukaran posisi, yang terbesar ada pada posisi m.

. Berikutnya ambillah elemen satu dari terakhir A[m-1], kemudian cari elemen terbesar diantara elemen lainnya (A[1] sampai A[m-2])

- Andaikan diperoleh A[y] terbesar, maka tukarkan dengan A[m-1]
- Demikian seterusnya hingga semua elemen telah diproses.

Sebagai contoh andaikan A = [25 27 10 8 76 21]

Putaran I : Ambil A[6] yaitu 21, kemudian cari yang terbesar diantara A[1] sampai A[5] diperoleh bahwa A[5] = 76 terbesar, karena 76 > 21 maka adakan penukaran tempat sehingga A = [25 27 10 8 21 76].

Putaran II : Ambil A[5] yaitu 21, kemudian cari yang terbesar diantara A[1] sampai A[4] diperoleh bahwa A[2] = 27 terbesar, karena 27 > 21 maka adakan penukaran tempat sehingga A = [25 21 10 8 27 76]

Putaran III : Ambil A[4] yaitu 8, kemudian cari yang terbesar diantara A[1] sampai A[3] diperoleh bahwa A[1] = 25 terbesar, karena 25 > 8 maka adakan penukaran tempat sehingga A = [8 21 10 25 27 76]

Putaran IV : Ambil A[3] yaitu 10, kemudian cari yang terbesar diantara A[1] sampai A[2] diperoleh bahwa A[2] = 21 terbesar, karena 21 > 10 maka adakan penukaran tempat sehingga A = [8 10 21 25 27 76]
Pada putaran ini ternyata larik A sudah terurut ascending.

prosedur Sort_MiniMax (in-out integer A[ ], **input integer** m **)**
{ mengurutkan larik dengan metoda maksimum **}**

Deklarasi
integer imax, temp;
integer putaran, idx;

Deskripsi
{ mula-mula maksimum ada pada A[m] }
imax ← m;
{ lakukan putaran sebanyak m-1 kali }
for (putaran = 1 **to** (m-1) **step** 1 **)**
{ cari yang terbesar diantara A[1] hingga A[m-putaran] }
for (idx = 1 **to** m-putaran **step** 1 **)**
if (A[idx] > A[imax] **)**
then {tukar tempat }
temp ← A[imax];
A[imax] ← A[idx];
A[idx] ← temp;
endif.
endfor.
{ ambil elemen berikutnya }
imax ← imax – 1;
endfor.

9.3 Teknik Penyisipan

Teknik penyisipan pada prinsipnya adalah dengan mengambil satu elemen berurut dari posisi awal satu persatu untuk disisipkan pada urutannya yang tepat. Prosedurnya adalah sebagai berikut.

- mula-mula anggaplah A[1] sudah tepat posisinya
- ambillah A[2], dan bandingkan dengan A[1], bila A[2] lebih kecil maka lakukan penggeseran elemen ke kanan, atau A[1] geser ke A[2], dan nilai A[2] yang tadi disisipkan pada A[1].
- ambil A[3], dan sisipkan pada posisinya antara A[1] dan A[2]
- lakukan hal yang sama untuk elemen-elemen selanjutnya.

Sebagai contoh andaikan A = [25 27 10 8 76 21]
Putaran I : A[1]=25, ambil A[2]=27, bandingan dengan A[1], posisi sudah tepat
Putaran II : ambil A[3]=10, bandingkan dengan A[1] dan A[2], geser A[1] ke A[2] geser A[2] ke A[3] dan sisipkan A[3] ke A[1] sehingga:
A = [10 25 27 8 76 21]
Putaran III : ambil A[4]=8, tempatkan pada posisinya diantara A[1] sampai A[3] geser A[1] hingga A[3] ke A[4], dan A[4] masuk ke A[1], sehingga
A = [8 10 25 27 76 21]
Putaran IV : ambil A[5]=76, bandingkan dengan yang lain, posisinya sudah tepat
Putaran V : ambil A[6]=21, bandingkan dengan yang lain, geser A[3] ke belakang sisipkan A[6] pada posisi A[3] sehingga: A = [8 10 21 25 27 76]
pada putaran ini larik A sudah terurut ascending.

prosedur Sort_Insertion (in-out integer A[ ], **integer** m **)**
{ mengurutkan larik dengan metoda penyisipan **}**

Deklarasi
integer x, posisi, idx;

Deskripsi

```
                                for ( posisi=2 to m step 1 )
                                    x ← A[posisi];
                idx ← 1;
                { cari posisi nya }
               while ( x > A[idx] && idx < posisi ) do
                        idx ← idx + 1;
                endwhile.
                { sisipkan pada posisi yang tepat }
               if ( x < = A[idx] )
                   then { geser ke kanan }
                        for ( k= posisi to idx+1 step -1 )
                            A[k] ← A[k-1];
                        endfor.
                        A[idx] ← x;
                endif.
        endfor.
```

10. OPERASI MATRIKS

Matriks adalah larik dua dimensi, memiliki dimensi baris dan dimensi kolom.

- Matriks *bujur sangkar* adalah matriks yang jumlah baris sama dengan jumlah kolom.
- Matriks *segi-empat* adalah matriks yang jumlah barisnya tidak sama dengan jumlah kolom.
- Matriks *diagonal* adalah matriks bujur-sangkar yang elemen-elemen-nya 0 kecuali elemen-elemen diagonal.
- Matriks *identitas* adalah matriks diagonal yang elemen-elemen diagonalnya semua bernilai 1.
- Matriks *segitiga atas* adalah matriks bujur-sangkar yang elemen-elemen dibawah diagonal semuanya 0.
- Matriks *segitiga bawah* adalah matriks bujur-sangkar yang elemen-elemen diatas diagonal semuanya 0.

Beberapa operasi matriks yang akan dibahas adalah:

- Menjumlahkan dua matriks
- Mengalikan dua matriks
- Mencari determinan matriks
- Melakukan transpose
- Mencari inversi matriks
- Menyelesaikan persamaan linier

10.1 Menjumlahkan Dua Matriks

Dua matriks hanya bisa dijumlahkan apabila dimensinya sama. Andaikan matriks A dan matriks B akan dijumlahkan menjadi matriks C, maka rumus umumnya adalah:

C[i, j] ← A[i, j] + B[i, j]

yang berarti hanya elemen pada posisi yang sama dapat dijumlahkan.

Contoh: andaikan A = [2 3 4] B = [6 3 7]
[5 8 6] [9 2 5]
[1 7 9] [4 1 8]

maka: C = [8 6 11]
[14 10 11]
[5 8 17]

Algortima-nya adalah sbb:

```
                    Algoritma Jumlah_Matriks
                    { menjumlahkan dua matrik }

Deklarasi
        const integer brs=3, klm=3;
        integer A[brs] [klm], B [brs] [klm], C [brs] [klm];
        integer ibx, ikx;

Deskripsi
        { baca matriks A }
        for ( ibx=1 to brs step 1 )
                for ( ikx=1 to klm step 1 )
                    read ( A[ibx][ikx] );
                    endfor.
        endfor.

        { baca matriks B }
        for ( ibx=1 to brs step 1 )
                for ( ikx=1 to klm step 1 )
                    read ( B[ibx][ikx] );
                    endfor.
        endfor.

        { menjumlahkan matriks A dan B }
        for ( ibx=1 to brs step 1 )
                for ( ikx=1 to klm step 1 )
                        C[ibx][ikx] ← A[ibx][ikx] + B[ibx][ikx];
                endfor.
        endfor.

        { menampilkan matriks C }
        for ( ibx=1 to brs step 1 )
                for ( ikx=1 to klm step 1 )
                    write ( C [ibx] [ikx] );
                    endfor.
                write ( );
        endfor.
```

10.2 Mengalikan Dua Matriks

Dua buah matriks dapat diperkalikan hanya apabila jumlah kolom dari matriks pertama sama banyaknya dengan jumlah baris dari matriks kedua.
Matriks A[3 x 4] bisa dikalikan dengan matriks B [4 x 5], menjadi C [3 x 5]
Matriks A[4 x 5] tidak bisa dikalikan dengan matriks B [3 x 4].

Rumus perkalian matriks C = A x B adalah: C[i, j] = Σ A[i, k] x B[k, j].

```
Contoh : A = [ 1 2 3 4 ]          B = [ 2 3 4 7 5 ]
             [ 5 8 9 7 ]              [ 6 8     1 9 3 ]
             [ 3 5 8 6 ]              [ 2 7 3 8 4 ]
                                      [ 3 5 2 9 6 ]
```

misalnya :

C[1][2] = A[1][1]*B[1][2] + A[1][2]*B[2][2] + A[1][3]*B[3][2] + A[1][4]*B[4][2]
= 1x3 + 2x8 + 3x7 + 4x5 = 60

C[2][5] = A[2][1]*B[1][5] + A[2][2]*B[2][5] + A[2][3]*B[3][5] + A[2][4]*B[4][5]
= 5x5 + 8x3 + 9x4 + 7x6 = 127

Algoritma Perkalian_Matriks
{ mengalikan dua buah matriks C = A x B **}**

Deklarasi

```
integer brsA=3, klmA=4, brsB=4, klmB=5;
integer A[brsA] [klmA], B [brsB] [klmB];
integer i, k, j;
```

Deskripsi

```
{ baca matriks A }
for ( i=1 to brsA step 1 )
    for ( k=1 to klmA step 1 )
        read ( A[i][k] );
    endfor.
endfor.

{ baca matriks B }
for ( i=1 to brsB step 1 )
    for ( j=1 to klmB step 1 )
        read ( B[i][j] );
    endfor.
endfor.

{ mengalikan matriks }
for ( i=1 to brsA step 1 )
    for ( j = 1 to klmB step 1 )
        C[i][j] ← 0;
        for ( k=1 to klmA step 1 )
            C[i][j] = C[i][j] + A[i][k] * B[k][j];
        endfor.
    endfor.
endfor.
```

```
{ menampilkan hasil perkalian }
for ( i=1 to brsA step 1 )
        for ( j=1 to klmB step 1 )
                write ( C[i][j] );
        endfor.
        write ( );
endfor.
```

10.3 Determinan Matriks

Determinan suatu matriks merupakan salah satu ukuran dari kekuatan elemen-elemen dari matriks bujur sangkar. Proses menghitung dapat dipahami melalui serangkaian contoh berikut ini:

Misalkan A = [a11 a12 a13]
[a21 a22 a23]
[a31 a32 a33]

maka Det(A) = a11.a22.a33 + a12.a23.a31 + a13.a21.a32
- a31.a22.a13 – a32.a23.a1 1 – a33.a21.a12
= a11.a22.a33 + a12.a23.a32 + a13.a21.a32
- a13.a22.a31 – a1 1.a23.a32 – a12.a21.a33

Misalkan B = [b11 b12 b13 b14]
[b21 b22 b23 b24]
[b31 b32 b33 b34]
[b41 b42 b43 b44]

maka Det(B) = b1 1.b22.b33.b44 + b12.b23.b34.b41 + b13.b24.b31.b42 +
b14.b21.b32.b43 - b14.b23.b32.b41 – b11.b24.b33.b42 –
b12.b21.b34.b43 - b13.b22.b31.b44

Proses menghitung determinan matriks dapat dibagi dua, yaitu:

1. Menghitung perkalian suku positif, yang dimulai dengan perkalian semua elemen diagonal, b1 1.b22.b33. dst, kemudian perkalian elemen-elemen dengan indeks baris mulai dari 1,2,3 dst yang diikuti indeks kolom mulai 2 dst, tetapi setiap kali indeks kolom telah mencapai maksimum (m) maka dikembalikan ke indeks 1. Algoritma bagian positif ini dapat ditulis sbb:

```
for (idx=1 to m step 1)
        pos[1] ← pos[1] * B[idx][idx];
endfor.
```

```
for (idx=2 to m step 1)
        klm ← idx;
        for (brs=1 to m step 1)
                pos[idx] ← pos[idx] * B[brs][klm];
                klm ← klm + 1;
                if (klm > m )
                   then klm ← 1;
                endif.
        endfor.
endfor.
```

2. Menghitung perkalian suku negatif, yang dimulai dengan perkalian elemen-elemen berindeks baris 1 s/d m, dan indeks kolom dimulai dari m dan menurun ke 1, misalnya: b1 1.b24.b33.b42, kemudian b12.b21.b34.b43 dst, algoritmanya sbb:

```
for (idx=1 to m step 1)
        klm ← idx;
        for (brs=1 to m step 1)
                neg[idx] ← neg[idx] * B[brs][klm];
                klm ← klm – 1;
                if ( klm < 1 )
                    then klm ← m;
                endif.
        endfor.
endfor.
```

Secara lengkap algoritma menghitung determinan adalah sbb:

Algoritma Determinan
{ menghitung nilai determinan sebuah matriks bujur sangkar **}**

Deklarasi
integer m = 4;
integer B [m] [m], pos [m], neg[m];
integer idx, brs, klm, D=0;

Deskripsi
{ baca elemen matriks **}**
for (brs=1 **to** m **step** 1 **)**
for (klm=1 **to** m **step** 1 **)**
read (B[brs][klm] **);**
endfor.
endfor.

{ inisialisasi nilai perkalian **}**

```
for ( idx=1 to m step 1 )
        pos[idx] ← 1;
        neg[idx] ← 1;
endfor.

{ perkalian positip }
for (idx=1 to m step 1)
        pos[1] ← pos[1] * B[idx][idx];
endfor.

for (idx=2 to m step 1)
        klm ← idx;
        for (brs=1 to m step 1)
                pos[idx] ← pos[idx] * B[brs][klm];
                klm ← klm + 1;
                if (klm > m )
                   then klm ← 1;
                endif.
        endfor.
endfor.

{ perkalian negatif }
for (idx=1 to m step 1)
        klm ← idx;
        for (brs=1 to m step 1)
                neg[idx] ← neg[idx] * B[brs][klm];
                klm ← klm – 1;
                if ( klm < 1 )
                    then klm ← m;
                endif.
        endfor.
endfor.

{ hitung hasil }
for ( idx=1 to m step 1 )
        D ← D + pos[idx] – neg[idx];
endfor.
write ( "Determinan = ", D );
```

10.4 Melakukan Transpose

Transpose adalah proses mengubah posisi elemen matriks sehingga elemen baris menjadi elemen kolom dan sebaliknya. Sebagai contoh:

```
A = [ 12 5 9   10 ]          Trans(A) = A^T = [ 12 7 4 ]
    [ 7 8 11    6 ]                           [ 5 8 10 ]
    [ 4 10 3    2 ]                           [ 9 11 3 ]
                                              [ 10 6 2 ]
```

Algoritma Transpose
{ melakukan transpose matriks **}**

Deklarasi

```
integer idx1=3, idx2=4;
integer A[idx1 ] [idx2], AT[idx2] [idx1 ];
integer ib, ik;
```

Deskripsi

```
{ membaca matriks }
for ( ib=1 to idx1 step 1 )
    for ( ik= 1 to idx2 step 1 )
        read ( A[ib][ik] );
    endfor.
endfor.

{ melakukan transpose }
for ( ib=1 to idx1 step 1 )
    for ( ik=1 to idx2 step 1 )
        AT[ik][ib] ← A[ib][ik];
        endfor.
endfor.

{ menampilkan hasil }
for ( ib=1 to idx2 step 1 )
    for ( ik= 1 to idx1 step 1 )
        write ( AT[ib][ik] );
    endfor.
endfor.
```

Pada hakekatnya sebuah matriks bujur-sangkar dapat ditranspose tanpa mendefinisikan suatu matriks baru (in-place transpose) tetapi ditranspose pada tempatnya sendiri.

```
A = [ 1 2 3 ]          Trans(A) = [ 1 4 7 ]
    [ 4 5 6 ]                     [ 2 5 8 ]
    [ 7 8 9 ]                     [ 3 6 9 ]
```

Perhatikan bahwa pada transpose elemen diagonal tidak berubah posisi, elemen A[1] [2] bertukar tempat dengan A[2][1], A[1][3] dengan A[3][1], dan A[2][3] dengan A[3][2]. Proses transpose-nya berlangsung sbb:

```
for ( ib=1 to idx1 step 1 )
    for ( ik=ib+1 to idx2 step 1 )
        if ( ib != ik )
            then temp ← A[ib][ik] );
                 A[ib][ik] ← A[ik][ib];
                 A[ik][ib] ← temp;
        endif.
    endfor.
endfor.
```

10.5 Menyelesaikan Persamaan Linier Simultan

- Persamaan Linier Simultan adalah suatu sistem persamaan yang terdiri atas lebih dari satu variabel dan melibatkan lebih dari satu persamaan dimana semua variabel dalam order-1 (linier).
- Tujuan dari persamaan linier simultan adalah untuk mencari nilai variabel yang terkait sedemikian rupa sehingga memenuhi semua persamaan yang ada.
- Secara teoritis agar dapat diselesaikan maka sebuah persamaan linier simultan harus memiliki persamaan yang minimal sama dengan banyaknya variabel yang tidak diketahui nilainya.

Bentuk umum suatu persamaan linier simultan dengan tiga variabel (x1, x2, dan x3) adalah sebagai berikut:

$$a11.x1 + a12.x2 + a13.x3 = b1$$
$$a21.x1 + a22.x2 + a23.x3 = b2$$
$$a31.x1 + a32.x2 + a33.x3 = b3$$

Bentuk umum diatas dapat diterjemahkan menjadi bentuk matriks sbb:

$$\begin{bmatrix} a11 & a12 & a13 \\ a21 & a22 & a23 \\ a31 & a32 & a33 \end{bmatrix} \begin{bmatrix} x1 \\ x2 \\ x3 \end{bmatrix} = \begin{bmatrix} b1 \\ b2 \\ b3 \end{bmatrix}$$

dalam simbol matriks ditulis sebagai : **A.x = b**, dimana **A** adalah matriks koefisien.

- Pada prinsipnya ada banyak cara untuk menyelesaikan persamaan linier simultan, namun pada kesempatan ini akan dibahas dua teknik saja, yaitu: metoda eliminasi Gauss, dan metoda eliminasi Gauss-Jordan.
- Metoda eliminasi Gauss, adalah metoda manipulasi yang meng-eliminir (me-nol-kan) elemen-elemen matriks yang berada dibawah diagonal sehingga matriks koefisien menjadi matriks segitiga atas, sbb:

```
[ a11 a12 a13 ] [ x1 ]       [ b1 ]
[ 0   a22 a23 ] [ x2 ]  =    [ b2 ]
[ 0    0   a33 ] [ x3 ]      [ b3 ]
```

Kemudian nilai variabel dapat dihitung dengan melakukan subsitusi balik:

```
x3 = b3 / a33;
x2 = (b2 – a23.x3) / a22;
x1 = (b1 – a12.x2 – a13.x3) / a11;
```

- Metoda Gauss-Jordan adalah metoda manipulasi yang meng-eliminir elemen-elemen matriks koefisien sehingga tercipta matriks diagonal, sbb:

```
[ a11   0   0   ] [ x1 ]    [ b1 ]
[   0  a22  0   ] [ x2 ]  = [ b2 ]
[   0   0  a33 ] [ x3 ]     [ b3 ]
```

dengan demikian maka nilai variabel dapat dihitung langsung, yaitu:

```
x1 = b1 / a11;
x2 = b2 / a22;
x3 = b3 / a33;
```

- Proses eliminasi dapat dilakukan dengan mengurangkan elemen dibawah diagonal dengan nilai pada diagonal, misalkan kita memiliki persamaan matriks sbb:

```
[  10   2   2 ] [ x1 ]    [ 20 ]
[   2  -5   4 ] [ x2 ]  = [  4 ]
[  -1   3   5 ] [ x3 ]    [ 20 ]
```

elemen A[2] [2] dapat dinolkan dengan cara mengurangkan baris dua dengan (2/10) kali baris pertama, dengan kata lain:

```
ambil m = A[2][1] / A[1][1] = 2 / 10
lalu A[2][1] = A[2][1] – m * A[1][1] = 0
     A[2][2] = A[2][2] – m * A[1][2] = -5 – (2/10)*2 = -5.4
     A[2][3] = A[2][3] – m * A[1][3] = 4 – (2/10)*2 = 3.6
     B[2] = B[2] – m*B[1] = 4 – (2/10)*20 = 0
```

```
untuk baris tiga, ambil m = A[3][1] / A[1][1] = -1 / 10
lalu A[3][1] = A[3][1] – m*A[1][1] = -1 + (1/10)*10 = 0
     A[3][2] = A[3][2] – m*A[1][2] = 3 + (-1/10)*2 = 2.8
     A[3][3] = A[3][3] – m*A[1][3] = 5 + (-1/10)*2 = 4.8
     B[3] = B[3] – m*B[1] = 20 + (-1/10)*20 = 18
```

dengan demikian matriks menjadi:

$$\begin{bmatrix} 10 & 2 & 2 \\ 0 & -5.4 & 3.6 \\ 0 & 2.8 & 4.8 \end{bmatrix} \begin{bmatrix} x1 \\ x2 \\ x3 \end{bmatrix} = \begin{bmatrix} 20 \\ 0 \\ 18 \end{bmatrix}$$

eliminasi elemen A[3][2] menjadi A[3][2] = 2.8 + (-2.8 / 5.4)*5.4 = 0

A[3][3] = 4.8 + (2.8 / 5.4) * 3.6 = 6

maka diperoleh matriks segitiga atas:

$$\begin{bmatrix} 10 & 2 & 2 \\ 0 & -5.4 & 3.6 \\ 0 & 0 & 6 \end{bmatrix} \begin{bmatrix} x1 \\ x2 \\ x3 \end{bmatrix} = \begin{bmatrix} 20 \\ 0 \\ 18 \end{bmatrix}$$

sehingga x3 = 18 / 6 = 3.0

x2 = (0 – 3.6*3) / 5.4 = 2.0

x1 = (20 – 2*2 – 2*3) / 10 = 1.0

Algoritma Gauss
{ menyelesaikan persamaan linier simultan dengan teknik eliminasi Gauss **}**

Deklarasi

```
real m;
integer brs=3, klm=3;
real A[brs][klm], B[brs], X[brs];
integer ib, ik, il;
```

Deskripsi

```
write ( "masukkan elemen matriks koefisien " );
for ( ib=1 to brs step 1 )
    for ( ik=1 to klm step 1 )
        read ( A[ib][ik] );
    endfor.
endfor.

write ( "masukkan nilai vektor ruas kanan " );
for ( ib=1 to brs step 1 )
    read ( B[ib] );
endfor.

{ proses eliminasi elemen matriks }
for ( ib=1 to brs-1 step 1 )
    for ( ik=ib+1 to brs step 1 )
        m ← - A[ik][ib] / A[ib][ib];
        for ( il=ib to brs step 1 )
            A[ik][il] ← A[ik][il] + m * A[ib][il];
        endfor.
```

```
            B[ik] ← B[ik] + m * B[ib];
        endfor.
    endfor.

    { proses subsitusi balik }
    X[brs] ← B[brs] / A[brs][brs];
    for ( ib=brs-1 to 1 step -1 )
        X[ib] ← B[ib];
        for ( ik=ib+1 to brs step 1 )
            X[ib] ← X[ib] – A[ib][ik] * X[ik];
        endfor.
        X[ib] ← X[ib] / A[ib][ib];
    endfor.

    { menampilkan hasil }
    for ( ib=1 to brs step 1 )
        write ( " x ", ib, " = ", X[ib] );
    endfor.
```

Algoritma Gauss_Jordan
{ menyelesaikan persamaan linier simultan dengan metode eliminasi Gauss Jordan yang melakukan eliminasi terhadap semua elemen kecuali elemen diagonal **}**

Deklarasi

```
    real m;
    integer brs=3, klm=3;
    real A[brs][klm], B[brs], X[brs];
    integer ib, ik, il;
```

Deskripsi

```
    write ( "masukkan elemen matriks koefisien " );
    for ( ib=1 to brs step 1 )
        for ( ik=1 to klm step 1 )
            read ( A[ib][ik] );
        endfor.
    endfor.

    write ( "masukkan nilai vektor ruas kanan " );
    for ( ib=1 to brs step 1 )
        read ( B[ib] );
    endfor.

    { proses eliminasi }
    for ( ik=1 to klm step 1 )
        for ( ib=1 to brs step 1 )
```

```
            { lakukan eliminasi kecuali diagonal }
              if ( ik != ib )
                  then m <- A[ib][ik] / A[ik][ik];
                        for ( il=ik to klm )
                              A[ib][il] <- A[ib][il] + m*A[ik][il];
                        endfor.
                       B[ib] <- B[ib] + m*B[ik];
                endif.
        endfor.
    endfor.

    { penyelesaian }
    for ( ib=1 to brs step 1 )
          X[ib] <- B[ib] / A[ib][ib];
        write ( "x", ib, " = ', X[ib] );
        endfor.
```

10.6 Mencari Inverse (Kebalikan) Matriks

- Matriks Inverse $\mathbf{A^{-1}}$ adalah suatu matriks yang apabila dikalikan dengan matriks aslinya **A** akan menghasilkan matriks identitas **I**, atau $\mathbf{A.A^{-1} = I}$.

- Apabila matriks invers ini diandaikan sebagai matriks **B** maka persamaan menjadi **A.B = I**, sehingga matriks **B** bisa diselesaikan dengan metoda eliminasi Gauss- Jordan.

```
[ a11  a12 a13 ] [ b11  b12 b13 ]     [ 1  0  0 ]
[ a21 a22 a23 ] [ b21 b22 b23 ] = [ 0  1  0 ]
[ a31 a32 a33 ] [ b31 b32 b33 ]     [ 0  0  1 ]
```

  pada hakekatnya apabila diperhatikan maka penyelesaian-nya bisa diperoleh dengan melakukan proses eliminasi Gauss-Jordan sebanyak tiga kali, atau sejumlah kolom pada matriks B.

- Suatu cara lain yang berasal dari proses eliminasi dan telah dibuktikan berhasil adalah dengan membuat matriks A dan matriks I berdampingan, kemudian eliminasi Gauss-Jordan diterapkan ke kedua matriks tersebut sehingga matriks A pada akhirnya menjadi matriks identitas I, dan pada saat itu matriks I menjadi inverse dari A.

```
[ a11  a12  a13  |  1  0  0 ]
[ a21  a22  a23  |  0  1  0 ]
[ a31  a32  a33  |  0  0  1 ]
```

  matriks ini dimanipulasi sehingga diperoleh:

```
[   1   0   0  |  b11   b12   b13 ]
[   0   1   0  |  b21   b22   b23 ]
[   0   0   1  |  b31   b32   b33 ]
```

maka matriks B adalah inverse dari matriks A.

Algoritma Balik_Matriks
{ menerapkan metoda Gauss-Jordan untuk mencari invers dari suatu matriks **}**

Deklarasi

```
real m;
integer brs=3, klm=3;
real A[brs] [klm], B [brs] [klm];
integer ib, ik, il, faktor;
```

Deskripsi

```
write ( "masukkan elemen matriks " );
for ( ib=1 to brs step 1 )
                for ( ik=1 to klm step 1 )
                    read ( A[ib][ik] );
    endfor.
endfor.

{ matriks identitas adalah B }
for ( ib=1 to brs step 1 )
                  for ( ik=1 to klm step 1 )
                        if ( ib = ik )
                then B[ib][ik] ← 1;
                else B[ib] ← 0;
             endif.
    endfor.
endfor.

{ proses eliminasi }
for ( ik=1 to klm step 1 )
    for ( ib=1 to brs step 1 )
          { lakukan eliminasi kecuali diagonal }
          if ( ik != ib )
              then m ← - A[ib][ik] / A[ik][ik];
                   for ( il=ik to klm )
                        A[ib][il] ← A[ib][il] + m*A[ik][il];
                     B[ib][il] ← B[ib][il] + m*B[ik][il];
                     endfor.
             endif.
    endfor.
endfor.
```

```
{ penyelesaian }
{ Jadikan matriks diagonal A menjadi matriks Identitas }
for ( ib=1 to brs step 1 )
      A[ib][ib] ← 1;
      B[ib][ib] ← B[ib][ib] / A[ib][ib];
endfor.

{ tampilkan hasil }
for ( ib= 1 to brs step 1 )
     for ( ik= 1 to klm step 1 )
               write ( B[ib][ik], “      “ );
     endfor;
     write ( );
endfor.
```

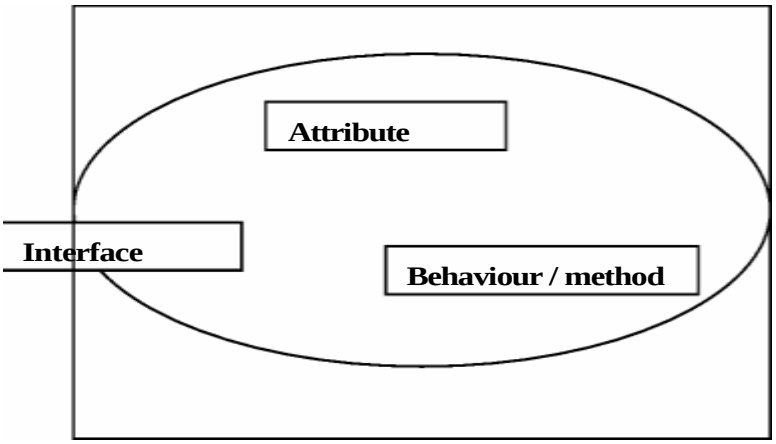
11. PEMROGRAMAN BER-ORIENTASI OBJEK

- Pemrograman ber-orientasi objek (Object Oriented Programming – OOP) merupakan satu ide luarbiasa dalam bidang pemrograman yang setelah diterapkan ternyata melahirkan bahasa yang lebih maju seperti: C++, Java, Python, dsb. Perkembangan dari OOP inipun melahirkan konsep pemrograman yang lain berbasis Windows atau GUI (Graphical User Interface) yang kemudian populer dengan bahasa visual, seperti Visual Basic, Delphi, Visual C, Visual Java, dsb.
- Para pemrograman yang terbiasa dengan konsep bahasa prosedural seperti BASIC, PASCAL, dan C, biasanya memerlukan suatu pengenalan ke dalam konsep objek agar kemudian dapat merancang program ber-orientasi objek.
- Bab ini akan membahas konsep objek dalam pemrograman serta prinsip perancangan program ber-orientasi objek.

11.1 Konsep Objek

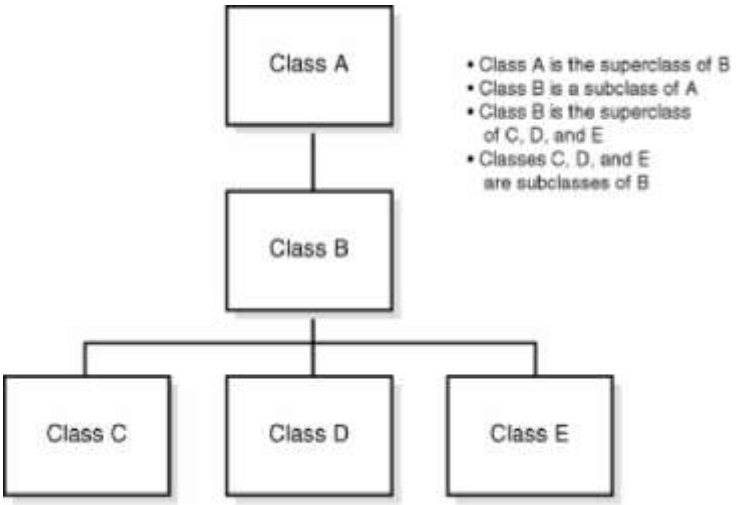
- Objek dalam pengertian sehari-hari adalah benda baik secara fisik dapat diketahui keberadaanya maupun yang bersifat khayal (virtual). Mobil Toyota, Komputer Mugen, Notebook Toshiba, Hotel, Restauran, atau suatu gagasan dan sebagainya adalah objek.
- Objek dalam pemrograman adalah suatu komponen dari suatu sistem yang dibangun, misalnya: Objek Matematis, Objek Jendela (window), Objek Tombol (button), Objek Kotak Gambar (picture box), Objek Kotak Teks (text box), dsb. Analogi Objek pada pemrograman prosedural adalah "fungsi" atau "prosedur" yang bekerja mandiri menghasilkan sesuatu dalam suatu proses pertukaran data dengan program utama.
- Setiap objek memiliki suatu kelompok, misalnya Toyota, Daihatsu, dan Honda adalah termasuk kelompok "mobil". Kelompok ini dalam konsep OOP disebut "kelas" atau "class".
- Kelas (class) pada hakekatnya sama dengan "cetak biru" atau "blue print" bagi suatu bangunan, kemudian rumah atau gedung yang dibangun berdasarkan cetak biru tersebut adalah Objek (Object).
- Setiap kelas merupakan suatu kesatuan yang terpisah (encapsulation) dari kelas lainnya. Kelas/objek harus memiliki "interface" agar bisa berkomunikasi dengan kelas/objek lain.
- Dalam suatu kelas terdapat dua hal utama, yaitu : sifat-sifat (attribute), dan fungsi (method) atau kelakuan (behavior). Misalnya kelas mobil memiliki atribut: warna body, jumlah tempat duduk, ukuran ban, dsb, kemudian terdapat fungsi seperti: pengapian, pengisian aki, percepatan (accelarator), rem, dsb.
- Sifat-sifat dari suatu kelas bisa diwariskan (inheritance) ke sub-kelas-nya, misal: sub-kelas mobil sedan, mewarisi attribute dari kelas mobil, demikian pula fungsi-fungsi kelas dapat diwariskan bila diperlukan.

. Kelas yang mewariskan sifat-sifat disebut “super-class”. Setiap sub-kelas dapat mengembangkan sendiri sifat-sifat serta kelakuannya sebagai tambahan dari sifat dan kelakuan utama yang diwarisi-nya dari super-class.



Objek Program

- Dalam pengertian fungsi dan kelakuan, dapat saja terjadi bahwa suatu fungsi yang nama-nya sama ternyata ber-kelakuan yang berbeda (polymorphism) yang dikenal dengan istilah method overriding.
- Suatu objek kelas dapat berhubungan dengan objek lain melalui suatu “interface” atau antar muka.



Contoh pembentukan objek dapat dilihat dari suatu bentuk laporan sebagai berikut:

Kode Pelanggan : A023-345 Nomor Pesanan : 12345
Nama Pelanggan : PT. Selalu Maju
Alamat : Jl. Pulau Kalimantan 345
Kota : Makassar
Telp : (411) 987-5556

Tgl Pemesanan : 25/02/2001 Tgl Kirim : 10/03/2001

NoUrut	KodeBarang	Nama-Barang	Jumlah	HargaSatuan
1	B123	Tinta Printer LX-80	5	150.000
2	C045	Diskette (box)	10	25.000
3	A43 1	Kertas HVS (rim)	6	3 0.000
4	B267	Tinta LaserJet HP	2	450.000

Paling tidak ada tiga (3) Kelas-Objek yang dapat di-ekstrak dari laporan diatas, yaitu:

Kelas Pelanggan :
Attribute : - Kode Pelanggan, Nama, Alamat, Kota, Telepon
Method : Input, Edit, Tampil
Interface : Kode Pelanggan

Kelas Pesanan :
Attribute : - Nomor Pesanan, Tgl Pemesanan, Tgl Kirim, Jumlah
Method : Hitung harga, Cetak Pesanan
Interface: - Kode Pelanggan, Kode Barang

Kelas Barang :
Attribut : Kode Barang, Nama Barang, Harga Satuan
Method : Input, Edit, Tampil
Interface : Kode Barang

11.2 Objek pada Disain Program

Pada rancangan program perlu ditetapkan beberapa tata-cara dalam mendefinisikan "kelas" atau "objek" beserta attribute dan method-nya.

Class *nama-kelas*
{
Attribute:
daftar atribut-atribut

Method:
definisi method / prosedur-prosedur

I

Contoh: berikut ini adalah defenisi kelas yang bernama Monster.

Class Monster
{
Attribute:
string warna;
string sex;
boolean lapar;

Method:
prosedur Beri_Makan( )
if (lapar=true **)**
then write("sedang melahap macan ..." **);**
lapar ← false;
else write("masih kenyang ... " **);**
endif.

prosedur Kenal( )
write ("Saya adalah Monster ", sex, "berwarna ", warna **);**
if (lapar=true **)**
then write("Sedang cari mangsa! awas!"**);**
else write ("Belum lapar, tidur dulu " **);**
endif.

I

Suatu Objek dapat didefinisikan sebagai salah satu implementasi dari suatu kelas, cara mendefinisikannya sebagai berikut:

Object *nama-kelas nama-objek;*

Contoh: **Object** Monster KingKong;
yang berarti KingKong adalah salah satu objek dari kelas Monster.

Object Monster Drakula;
Drakula adalah objek lain dari Monster.

Suatu algoritma yang memanfaatkan kelas Monster diatas dapat dibuat sebagai berikut:

Algoritma Monster
{ contoh algoritma yang memakai kelas Monster **}**

Deklarasi
Object Monster KingKong;

Deskripsi
KingKong ← **new** Monster;
KingKong.sex ← "jantan";
KingKong.warna ← "hitam";
KingKong.lapar ← true;

write ("Atribut dari Monster ini adalah sbb: "**);**
KingKong.**Kenal( );**
write ("Coba diberi makan : "**);**
KingKong.**Beri_Makan( )**;
write ("Beri makan lagi : "**);**
KingKong.**Beri_Makan( )**;

Ada beberapa hal yang perlu mendapat perhatian dalam contoh algoritma diatas:

1. Sebuah objek dapat di-inisialisasi melalui perintah **new,** misalnya:
   KingKong ← **new** Monster;

2. Atribut yang bersifat umum (public attribute) dapat diakses atau diberi nilai melalui program (algoritma) dengan menyebut nama objek diikuti nama atribut (perhatikan tanda titik diantara-nya), misalnya:
   KingKong.sex ← "jantan";
   KingKong.warna ← "hitam";
   KingKong.lapar ← true;

3. Kelakuan atau Method dapat dipanggil dengan cara menulis nama objek diikuti tanda titik dan nama method / fungsi, misalnya:
   KingKong.**Kenal( );**
   KingKong.**Beri_Makan( )**;

Bahasa yang ber-orientasi objek pada umumnya memiliki pustaka objek (*objects library*) dimana terdapat berbagai kelas yang dapat dimanfaatkan oleh programmer ketika membangun suatu sistem perangkat lunak.

Proses "inheritance" atau pewarisan atributdan metoda suatu kelas ke sub-kelasnya dapat dilakukan melalui perintah : ***extends nama_kelas***, misalnya suatu Kelas Tulisan mewariskan sifat-sifat dan fungsi-nya ke sub-kelas Makalah, maka dapat ditulis sbb:

Class Makalah **extends** Tulisan **{**

I

Apabila sub-kelas yang didefinisikan ini memiliki sifat-sifat tambahan maka sifat-sifat tambahan tersebut langsung ditambahkan saja dalam definisi atribut sub-kelas. Perhatikan contoh berikut ini;

```
Class Tulisan
{
        Attribute :
                int nomer;
                string judul;
                string penulis;
                string abstrak;

        Method :
                prosedur input_data( )
                        write ( "Nomor tulisan : "); read ( nomer );
                        write ( "Judul : "); read ( judul );
                        write ( "penulis : " ); read ( penulis );
                        write ( "abstrak : " ); read ( abstrak );

                prosedur tampil_data( )
                        write ( "Nomer : " , nomer );
                        write ( "Judul : " , judul );
                        write ( "Penulis : ", penulis );
                        write ( "Abstrak : ", abstrak );
I

Class Makalah extends Tulisan
{
        Attribute :
                string jurnal;
                string tahun;

        Method :
                prosedur input_data( )
```

```
                    Tulisan :: input_data( );
                      write ( "Nama Jurnal : " ); read ( jurnal );
                      write ( "Tahun terbit : " ); read ( tahun );

            prosedur tampil_data( )
                    Tulisan :: tampil_data( );
                    write ( "Nama Jurnal : ", jurnal );
                    write ( "Tahun terbit : ", tahun );
}
```

Pada contoh diatas terlihat bahwa Kelas Makalah adalah sub-kelas dari Tulisan, namun terdapat dua tambahan atribut sehingga method-nya juga harus disempurnakan. Perhatikan bagaimana fungsi / method dikembangkan dengan cara memanggil fungsi dari super-kelas menggunakan dua tanda titik-dua, Tulisan :: input_data( ), kemudian instruksi tambahannya mengikuti.

Algoritma yang menggunakan kedua kelas pada contoh diatas dapat ditulis sbb:

```
Algoritma Artikel
{ algoritma yang menggunakan inheritance pada definisi objek }

Deklarasi
    Makalah paper1, paper2;

Deskripsi
                                paper1 ← new Makalah;
                                paper2 ← new Makalah;

    write ( "Masukkan data makalah pertama " );
    paper 1. input_data( );
    write ( "Masukkan data makalah kedua " );
    paper2.input_data( );

    write ( "Ini data makalah pertama : ");
    paper1 .tampil_data( );
    write ( "Ini data makalah kedua : " );
    paper2.tampil_data( );
```

11.3 Graphical User Interface (GUI)

Dewasa ini bahasa program ber-orientasi visual tersedia, seperti Visual BASIC, Visual C, Visual JAVA, Delphi, Visual dBASE, Visual FoxPro, dsb. Pada hakekatnya bahasa-bahasa tersebut ber-orientasi objek, dimana tersedia objek-objek untuk visualisasi pada layar monitor yang populer dengan istilah ***Graphical User Interface (GUI)*** , seperti objek jendela (window, panel, atau form), objek teks (textbox, label), objek daftar pilihan

(list, combo-box, spin-box), objek tombol (button), dsb. Setiap objek tersebut memiliki attribute (properties) dan method (function). Objek-objek dapat berkomunikasi melalui *interface*, dan *events*. Events adalah kejadian/program yang diaktifkan melalui satu peristiwa, misalnya karena mouse di-tekan (*on_Click)*, atau isi suatu kotak data_entry diubah (*on_Change*), suatu jendela atau suatu file dibuka (*on_Open*), atau ditutup (*on_Close*), dsb.

Suatu pustaka objek bisa memuat berbagai macam objek, misalkan pustaka objek jendela (window tool kit) bisa memuat kelas-kelas untuk objek-objek yang disebutkan tersebut diatas. Untuk menggunakan objek-objek tersebut maka pustaka-nya harus dipanggil terlebih dahulu, misalnya dengan instruksi **import** seperti contoh berikut:

import window; memanggil semua objek dari pustaka window

import window.panel; memanggil hanya objek panel saja

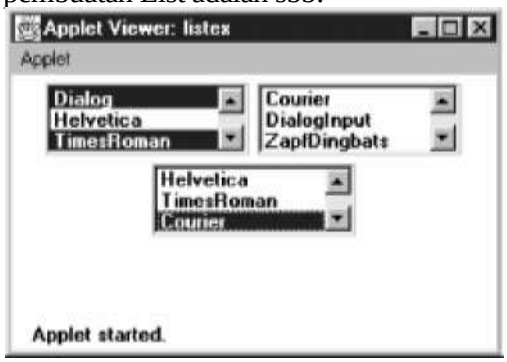
import window.list; memanggil hanya objek list saja.

Sebagai contoh kita tinjau objek list, andaikan memiliki method-method sebagai berikut:

Beberapa fungsi / method yang dapat diterapkan pada List adalah:

addItem(pilihan): fungsi untuk mengisi List dengan pilihan-pilihan
getItem(posisi) : fungsi untuk memperoleh teks pilihan pada posisi yang diberikan.
countItems( ) : fungsi untuk mengetahui berapa banyak pilihan dalam List
getSelectedIndexes( ) : fungsi untuk mengetahui indeks (nomor) dari pilihan tertentu
getSelectedItem( ) : fungsi untuk mengetahui pilihan yang saat ini terpilih
getSelectedItems( ) : fungsi untuk mengetahui semua pilihan yang terpilih
select(nomor) : pilih pilihan pada nomor yang diberikan
select(pilihan) : pilih pilihan yang sama dengan string pilihan yang diberikan.

Contoh pembuatan List adalah sbb:




```
List list1, list2, list3;
list1 = new List(3, true);
list1 .addItem("Dialog");
list1 .addItem("Helvetica");
list1 .addItem("TimesRoman");

 list2 = new List(3, false);
```

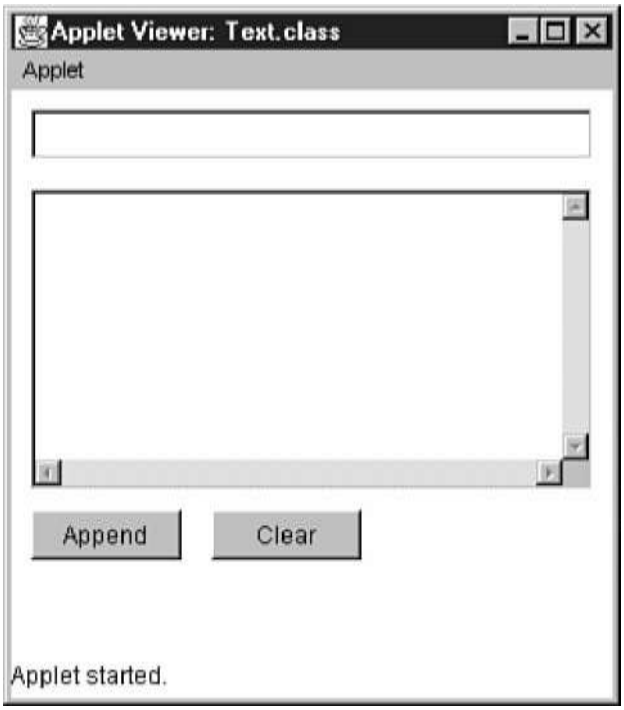
```
list2.addItem("Courier");
list2.addItem("DialogInput");
list2.addItem("ZaplDingbats");
list3 = new List(3, true);
list3.addItem("Helvetica");
list3.addItem("TimesRoman");
list3.addItem("Courier");
this.add(list1);
this. add(list2);
this. add(list3);
int  jum1  =  list1.countItems();
int  jum2  =  list2.countItems();
int  jum3  =  list3.countItems();
list1.select(1);
list3.select(3);
```

Perhatikan tampilan GUI berikut ini :

Tampilan ini terdiri atas beberapa objek antara lain:

- TextBox atau TextField
- TextArea
- Button

Algoritma untuk menampilkan GUI ini adalah sbb:



TextField

Text Area

Button

```
import applet.*;
import awt.*;
import awt . event. *;

public class Text extends Applet
{
    TextField textField = new TextField();
    TextArea textArea = new TextArea();
    Button append = new Button("Append");
    Button clear = new Button("Clear");

    public void init()
    {
       setLayout (null);
       textField.setBounds(10,10,280,25);
       textArea. setBounds (10 ,50 , 280, 150) ;
       append. setBounds (10 ,210 , 75, 25) ;
       clear . setBounds (100 ,210 , 75, 25) ;
       append. addActionListener (new ButtonHandler());
       clear. addActionListener (new ButtonHandler ());
       add(textField);
       add(textArea);
       add (append);
       add (clear);
    }

    class ButtonHandler implements ActionListener
    {
         public void actionPerformed(ActionEvent ev)
         {
             String s=ev.getActionCommand();
             if (s . equals ("Append"))
                then {
                       String text =
                           textArea . getText () +textField. getText ();
                       textArea. setText (text);
                     }
                else if(s.equals("Clear"))
                         then textArea.setText("");
         }
    }
```

Keterangan :

- Tiga baris pertama adalah meng-import pustaka objek yaitu :
    - applet – objek aplikasi pada halaman Web
    - awt – tool kit untuk window
    - awt.event – tool kit untuk mengantisipasi event

. Kemudian dibentuk suatu objek Text yang mewarisi objek Applet

- Menyusul definisi variabel/properties untuk TextField, TextArea, dan Button
- Kemudian definisi dari fungsi / method yaitu fungsi Init untuk inisialisasi dari objek-objek TextField, TextArea, dan Button
- Kemudian didefinisikan suatu kelas yang mengantisipasi event ketika Button Append dan Button Clear di-klik.
    - Setiap kali Append di-klik maka isi TextField ditambahkan ke dalam TextArea
    - Setiap kali Clear di-klik maka isi TextArea dibersihkan.