MODUL 1

Struktur Bahasa PASCAL secara umum

Pascal mempunyai struktur sebagai berikut:

1. Bagian Judul Program
2. Bagian Deklarasi e

a. Deklarasi tipe data (TYPE)

b. Deklarasi variabel (VAR)

c. Deklarasi konstanta (CONST)

d. Deklarasi label (LABEL)

e. Deklarasi sub-program (PROCEDURE dan FUNCTION)

3. Bagian Program Utama Perintah-perintah.

Teks Pascal setidaknya memiliki bagian Judul Program, bagian Deklarasi, dan Bagian Program Utama yang berupa perintah-perintah. Sedangkan untuk bagian deklarasi menyesuaikan dengan isi dari program itu sendiri. Contoh program PASCAL:

```
program TAMBAH_00; { Menjumlahkan dua bilangan yang nilainya diberikan dalam perintah}
var X, Y, Z: integer; { Deklarasi variabel X,Y dan Z sebagai bilangan bulat }
BEGIN { Program Utama Mulai }
X := 50; { Perintah memberikan nilai 50 pada var. X }
Y := 25; { Perintah memberikan nilai 25 pada var. Y }
Z := X + Y; { Perintah menjumlahkan X dan Y serta menyimpan hasilnya ke Z}
END. { Akhir Program Utama }
```

Pada contoh ini nilai X dan Y tidak bisa sembarang, karena didefiniskan tertentu. Agar nilai X dan Y bisa bebas ditentukan, nilai X dan Y dibaca dari default input.

```
program TAMBAH_01; { Menjumlahlan dua buah bilangan yang dibaca dari default input }
var X, Y, Z: integer; { Deklarasi variabel X,Y dan Z sebagai bilangan bulat }
BEGIN { Program Utama Mulai }
read(X); { Membaca nilai X lewat key-board }
read(Y); { Membaca nilai Y lewat key-board }
Z := X + Y; { Menjumlahkan X dan Y serta menyimpan hasilnya ke Z }
write(Z); { Menyajikan Z ke layar monitor }
END. { Akhir Program Utama }
```

Dasar Bahasa PASCAL

Unsur-unsur Pemrograman

a. Mendapatkan data dengan membaca data dari default input (key board, file atau sumber data lainnya).
b. Menyimpan data ke dalam memori dengan struktur data yang sesuai,
c. Memproses data dengan instruksi yang tepat.
d. Menyajikan atau mengirimkan hasil olahan data ke default output (monitor, file atau tujuan lainnya).

Jenis identifier

a. Identifier umum

Merupakan identifier yang didefinisikan sendiri oleh pemrogram. Pemrogram mempunyai kebebasan untuk menentukan nama identifiernya, dengan syarat nama tersebut tidak sama dengan identifier standar dan reserved word yang akan dibahas lebih lanjut. Hal ini untuk mencegah kesalahan yang bisa timbul akibat tumpang tindih identifier dalam program.

b. Identifier Standar (Baku)

Merupakan identifier yang didefinisikan oleh pembuat kompiler Pascal. Biasanya pembuat kompiler menyediakan suatu library yang sudah ada didalam kompiler. Library berisi berbagai procedure, fungsi atau unit yang sudah siap pakai. Misalnya Turbo Pascal Windows 1.5 memiliki suatu unit untuk memproses output yaitu wincrt, gotoxy, yang dengan mudah bisa dipakai oleh programmer di dalam menuliskan kode-kode programnya. Dinamai Identifier Standar karena suatu kompiler tidak harus memilikinya, masing-masing kompiler dimungkinkan mempunyai identifier yang berbeda untuk suatu tugas yang hampir sama. Misalnya Turbo Pascal versi DOS menggunakan crt untuk melakukan fungsi yang sama dengan wincrt (TPW 1.5). Beberapa Identifier Standar yang dimiliki oleh kompiler-kompiler Pascal antara lain:

abs arctan boolean char cos dispose eof eoln exp false input integer ln maxint new odd ord output pack page pred read readln real reset rewrite round sin sqr sqrt succ text true trunc write writeln

c. Identifier "reserved word", yaitu yang sudah didefinisikan dan digunakan oleh bahasa PASCAL sendiri (Kita tidak bisa menamai identifier kita dengan ini).

and array begin case const div do downto else end file for forward function goto if in label mod nil not of or packed procedure program record repeat set then to type until var while with

Deklarasi Variable:

Mendeklarasikan varibel adalah:

a. Memberikan nama variabel sebagai identitas pengenal

b. Menentukan tipe data variabel

Contoh deklarasi variabel:

```
var K : integer;
R : real;
C : char;
T : boolean;
```

Beberapa identifier yang sejenis bisa dideklarasikan bersamaan.

```
var i, j, k : integer;{Variabel i, j dan k sebagai integer}
namaMHS, alamatMHS : char; {Nama dan alamat mahasiswa }
```

Deklarasi Konstanta:

Mendeklarasikan konstanta adalah:

a. Memberikan nama konstanta sebagai identitas pengenal

b. Menentukan nilai konstanta

Contoh deklarasi konstanta:

```
const MaximumSize = 100; {integer }
ExitCommand = 'Q'; {char }
```

Tipe Data

Tipe data yang disediakan oleh PASCAL meliputi:

1. Tipe Data Sederhana

merupakan tipe data dasar yang sering dipakai oleh program, meliputi: integer (bilangan bulat), real (bilangan pecahan), char (alphanumerik dan tanda baca), dan boolean (logika). Untuk data integer dan real masing-masing terbagi menjadi beberapa kategori

a. Bilangan Integer

merupakan tipe data berupa bilangan bulat, terbagi atas beberapa kategori seperti

terlihat dalam tabel 1. tabel 1 menunjukkan jenis data, ukuran dalam memori dan rentang nilainya.

tabel 1. Tipe Data Bilangan Integer

Tipe Data	Ukuran Tempat	Rentang Nilai
Byte	1 byte	0 s/d +255
Shortint	1 byte	-28 s/d +127
integer	2 bytes	-32768 s/d 32767
Word	2 bytes	0 s/d 65535
Longint	4 bytes	2147483648 s/d 2147483647

Contoh bilangan integer adalah: 34 6458 -90 0 1112 Penggolongan tipe data integer tersebut dimaksudkan untuk membatasi alokasi memori yang dibutuhkan misalkan untuk suatu perhitungan dari suatu variabel bilangan diperkirakan nilai maksimumnya 32767 kita cukup mendeklarasikan variabel bilangan sebagai integer (2 byte), daripada sebagai longint(4 byte). Di dalam kompilernya, Pascal menyediakan konstanta untuk bilangan Integer yaitu: MaxInt and MaxLongInt, pemrogram bisa menggunakannya di dalam programnya tanpa harus terlebih dahulu mendefinisikannya.

-MaxInt bernilai 32.767
-MaxLongint bernilai 2.147.483.647.
contoh:

```
Program display_maxint;
uses wincrt;
begin writeln (maxint)
end.
```

Hasilnya: 32.767

b. Bilangan Real

Bilangan real atau nyata merupakan jenis bilangan pecahan, dapat dituliskan secara biasa atau model scientific . Contoh bilangan real: 34.265 -3.55 0.0 35.997E+11, dimana E merupakan simbol perpangkatan 10. Jadi 452.13 mempunyai nilai sama dengan 4.5213e2. Penggolongan tipe data bilangan real dapat dilihat pada tabel 2. Bilangan Real

Tabel 2. Tipe Data Bilangan Real

Tipe Data	Ukuran Tempat	Rentang Nilai
real	6 bytes	2.9 x 10-39 s/d 1.7 x1038
single	4 bytes	1.5 x 1045 s/d 3.4 x 1038
double	8 bytes	5.0 x 10-324 s/d 1.7 x 10308
extended	10 bytes	3.4 x 10-4932 s/d 1.1 x 104932
comp	8 bytes	-9.2x 1018 s/d 9.2x 1018

c. Char

Tipe data ini menyimpan karakter yang diketikkan dari keyboard, memiliki 266 macam yang terdapat dalam tabel ASCII (American Standard Code for Information Interchange). Contoh: 'a' 'B' '+', dsb. Yang perlu diingat bahwa dalam menuliskannya harus dengan memakai tanda kutip tunggal. Jenis data ini memerlukan alokasi memori sebesar 1(satu) byte untuk masing-masing data.

d. Tipe Data Boolean

merupakan tipe data logika, yang berisi dua kemungkinan nilai: TRUE (benar) atau FALSE (salah). Turbo Pascal for Windows memiliki tiga macam jenis ini yaitu: Boolean, WordBool, dan LongBool. Tipe boolean memakai memori paling kecil, sedangkan WordBool dan LongBool dipakai untuk menulis program yang sesuai dengan lingkungan Windows.

Tabel 3. Tipe Data Boolean

Tipe Data	Ukuran Tempat
Boolean	1 byte
WordBool	2 byte
Longbool	3 byte

Sebagai bilangan ordinal boolean true mempunyai nilai 1(satu), sedangkan false nilainya adalah 0(nol).

```
Contoh: Program display_bool;
uses wincrt;
begin writeln(ord(true));
writeln(ord(false));
end.
```

Hasilnya: 1 0 3.2.

Tipe Data Terstruktur

tipe ini terdiri atas : array, record, set, dan file. String adalah tipe data jenis array, tetapi karena string memiliki kekhasan tersendiri sebagai array dari karakter maka penulis perlu memberikan penjelasan tersendiri. Sedangkan untuk array, record, dan file perlu dijelaskan dalam bab yang lain karena agak banyak hal-hal yang perlu dibahas.

a. **Tipe Data String**

merupakan suatu data yang menyimpan array (larik), sebagai contoh 'ABCDEF' merupakan sebuah konstanta string yang berisikan 6 byte karakter. Ukuran Tempat untuk tipe data ini adalah 2 s/d 256 byte, dengan jumlah elemen 1 s/d 255. String dideklarasikan dengan string [konstanta] atau string. Bila ukuran string tidak didefinisikan maka akan banyak memakan ruang, karena ukuran string menyesuaikan dengan defaultnya. Misalkan

var kata: string [20]; atau var kata: string; karena string merupakan array dari karakter. Maka kata[1] merupakan karakter pertama dari string, kemudian kata[2], merupakan elemen kedua, dst.

Contoh:

```
Program hal_string;
Uses wincrt;
var s : string;
begin s:='Hello';
writeln(s);
writeln('panjang dari string adalah: ',ord(s[0]));
end.
```

Karakter nol merupakan karakter yang menyatakan panjang string. Sehingga ord(s[0]) menyatakan panjang dari string tersebut. Panjang string juga bisa dinyatakan sebagai length(s).

MODUL 2

ARRAY (LARIK)

1. Pendahuluan

Suatu array adalah sebuah struktur data yang terdiri atas banyak variabel dengan tipe data sama, dimana masing-masing elemen variabel mempunyai nilai indeks. Setiap elemen array mampu untuk menyimpan satu jenis data (yaitu: variabel). Suatu array dinyatakan dengan type, sehingga variabel yang bekerja akan dinyatakan dengan:

contoh type

A = array [1..10] of integer;

1	2	3	4	5	6	7	8	9	10

Secara logika pendefinisian array di atas merupakan sekumpulan kotak , dimana tiap kotak mempunyai nilai indeks integer 1, 2, 3, ...,9, 10 tiap elemen array ditandai dengan:

A[1], A[2], A[3], A[4], A[5], A[6], A[7], A[8], A[9], A[10]

2. Sifat Array

Array merupakan struktur data yang statis, yaitu jumlah elemen yang ada harus ditentukan terlebih dahulu dan tak bisa di ubah saat program berjalan. Untuk menyatakan array dalam PASCAL kita harus terlebih dahulu:

Mendefinisikan jumlah elemen array,–

Mendefinisikan– tipe data dari elemen array

Contoh. const N=10;

type

A= array [1..N] of integer;

3. Array Satu Dimensi

Pernyataan di atas merupakan penjelasan tentang array dengan satu dimensi. Pendefinisian array secara umum adalah sebagai berikut: jika kita ingin membuat beberapa array dengan tipe/jenis yang sama, kita lebih baik jika mendeklarasikan dengan type selanjutnya dengan deklarasi var.

SYNTAX

```
Type nama_array = ARRAY[bawah..atas] of tipe_data;
var variabel_array : nama_array;
```

atau dengan menggunakan statemen var :

```
var variabel_array : ARRAY[bawah..atas] of tipe_data;
```

Penjelasan: Bawah dan Atas menyatakan batas untuk array. tipe_data adalah merupakan tipe variabel yang dipunyai array (mis. Integer, char, real, dsb)

Contoh:

type intarray = ARRAY [1..20] of integer;

Pernyataan diatas adalah pernyataan untuk membentuk suatu array bernama intarray,yang berisi 20 tempat untuk bilangan integer. Setiap posisi disebut elemen, yang menyimpan suatu bilangan integer.langkah berikutnya adalah membuat suatu variabel kerja dengan tipe intarray yaitu,

var numbers : intarray;

kita bisa melakukan operasi pada setiap elemen dari numbers secara individual. Contoh kita bisa memberi nilai pada suatu elemen array seperti berikut:

numbers[2] := 10;

perintah ini memberikan suatu nilai integer 10 pada elemen ke-2 dari array numbers. Nomor dari elemen ditempatkan didalam kurung tegak. Contoh berikut adalah merupakan array yang menyimpan variabel-variabel integer. Data dengan

tipe integer hanya bisa dimasukkan satu persatu, kemudian baru bisa ditampilkan di monitor secara bersamaan

Contoh a.

```
program INT_ARRAY;
uses wincrt;
const N=10;
type int_array = ARRAY [1..N] of integer;
var bil : int_array;
indeks : integer;
BEGIN
writeln('masukkan sepuluh bilangan integer.');
for indeks := 1 to 10 do
begin
readln(bil[indeks]); { loop untuk memasukkan elemen array }
end;
writeln('Isi dari array ini adalah'); { tampilkan setiap elemen }
for indeks := 1 to 10 do
begin
writeln('bil[', indeks:2,'] adalah ',bil[indeks] );
end
END.
```

Contoh b.

```
program contoh_ARRAY;
uses wincrt;
var
a : array[1..10] of byte;{maksimum jumlah elemen=10}
begin
```

```
a[1]:=10;
a[2]:=15;
a[3]:=a[1]+a[2];
writeln(a[1]);
writeln(a[2]);
writeln(a[3]);
end.
```

Latihan

1. Menggunakan Array 1 dimensi buatlah program dengan ketentuan:

Input---→ Nilai PPN, Nilai PPA, Nilai Logika, Nilai Agama

Output-→ Total Nilai

Ket: Nama Array = nilai

Nama variabel = n

Jumlah Range = 5

2. Untuk soal no.1 tambahkan proses untuk mendapatkan kelulusan jika nilai logika > 7 dan proses untuk mendapatkan grade (A jika total >34, B jika total > 28, C jika total > 24 dan D jika total <=24).

MODUL 3

4. Array Multidimensi

Dalam array multidimensi terdiri atas baris (row) dan kolom (column). Index pertama adalah baris dan yang kedua adalah kolom .

SYNTAX

```
Type nama_array =ARRAY[bawah..atas, bawah..atas] of tipe_data;
var variabel_array : nama_array;
```

atau dengan menggunakan statemen var :

```
SYNTAX var variabel_array : ARRAY[bawah..atas, bawah..atas] of tipe_data;
```

Pernyataan berikut membentuk suatu array integer dengan nama bilangan , 10 x 10 elemen (100).

```
type matriks = ARRAY [1..10, 1..10] of integer;
var AKU: matriks;
```

untuk memasukkan tiap elemen maka, diperlukan suatu procedure dengan mempergunakan struktur pengulangan for ...do tersarangseperti berikut:

```
procedure ISI_MATRIK(AKU:matriks; m,n:integer);
var
i,j: integer; {faktor pengulang}
begin
for i:=1 to m do
begin
for j:=1 to n do
begin
```

```
read(A[i,j]);
end;
readln ;{ini memungkinkan kita menulis tiap baris elemen}
end;
```

untuk menampilkan tiap elemen maka, digunakan struktur pengulangan for ...do tersarang seperti berikut

```
procedure TULIS_MATRIK(AKU:matriks; m,n:integer);
var i,j: integer; {faktor pengulang}
begin
for i:=1 to m do
begin
for j:=1 to n do
begin
write(A[i,j]:6);
end;
writeln ; {ini memungkinkan kita menulis elemen dalam baris dan kolom }
end;
end;
```

5. Operasi pada Array

Sifat masing-masing elemen array mengikuti jenis data yang dimilikinya, untuk array dengan tipe bilangan integer atau real kita bisa melakukan berbagai standar operasi aritmatika seperti penjumlahan, perkalian, pengurangan, dsb. Yang perlu di garis bawahi, bahwa sifat dari array dimanfaatkan untuk operasi matrik.

a. Mencari Harga Tertentu pada Array

Mencari suatu elemen data di dalam suatu data merupakan suatu kejadian yang sering kita alami, contoh: mencari nama mahasiswa dari daftar presensi. Pencarian beruntun (sequence), merupakan suatu teknik untuk mencari suatu

elemen dalam suatu sistim yang lebih besar.

Contoh.

Misal array A[8], dengan elemen sbb:

A

60 12 76 23 11 42 18 42

Untuk mencari apakah bilangan x=11 ada didalam tabel maka dilakukan pemeriksaan terhadap :

60 12 76 23 11

Sehingga ditemukan x pada elemen ke-5, dalam bahasa PASCAL diterjemahkan seperti berikut:

```
type PITA = ARRAY [1..8] of integer;
var AKU: PITA;
procedure CARI_MATRIK(AKU: PITA);
var
i: integer; {faktor pengulang}
begin
for i:=1 to 8 do
begin
if AKU[i]:= 11 then
writeln(‘ terdapat bilangan 11 dalam pita ini ‘);
else
writeln(‘ tidak ada bilangan 11, pencarian berhenti ‘);
end;
end;
```

b. Mencari Harga Maksimum pada Array

Misal array di atas kita cari harga yang tertinggi, maka kita perlu menentukan nilai tertinggi dahulu sebelum melakukan pencarian ; diawali dengan nilai maksimum=0

```
procedure CARI_MAKSIMUM(AKU: PITA);
var
i: integer; {faktor pengulang}
MAKS : integer;
begin
MAKS := AKU[1];
for i:=1 to 8 do
begin
if AKU[i]> MAKS then
MAKS:= AKU[i];
End;
Writeln('NILAI MAKSIMUM = ',MAKS);
end;
```

b. Mencari Harga Minimum pada Array

Misal array di atas kita cari harga yang terendah, maka kita perlu menentukan nilai terendah dahulu sebelum melakukan pencarian ; diawali dengan nilai maksimum=3200

```
procedure CARI_MINIMUM(AKU: PITA);
var
i: integer; {faktor pengulang}
MIN : integer;
begin
MIN := 3200;
for i:=1 to 8 do
begin
if AKU[i]< MIN then
MIN:= AKU[i];
end;
```

```
writeln('NILAI MINIMUM = ',MIN);
end;
```

c. Matrik

Sebagai perwujudan dari array dua dimensi, operasi aritmatika seperti penjumlahan, perkalian, dan pengurangan bisa dilakukan.

Contoh.

- Mendefinisikan Elemen

```
Program OPERASI_MATRIK;
uses wincrt;
type
matrik=array[1..100,1..100] of real;
var
m,n, p, q: integer; {dimensi dari matrik}
A,B,C: matrik; {matrik A, B sebagai input, C sebagai hasil}
```

- Membaca Elemen Matrik

```
procedure bacamatrik(var A:matrik; m,n:integer);
var
i,j: integer; {faktor pengulang}
begin {read}
for i:=1 to m do
begin {do}
for j:=1 to n do
read(A[i,j]);
readln;
end; {do}
end; {read}
```

- Menampilkan Elemen Matrik

```
procedure tulismatrik(A:matrik; m,n:integer);
var
i,j: integer; {faktor pengulang}
begin {write}
for i:=1 to m do
begin {tiap baris}
writeln;
for j:=1 to n do
write(A[i,j]:6:2);
end; {tiap baris}
writeln;
end; {write}
```

- Penjumlahkan Matrik

```
procedure check_matrik(A,B,C:matrik; m,n,p,q:integer);
var i,j :integer;
begin
if (m=p) and (n=q) then
begin
for i:=1 to m do
begin
for j:=1 to n do
begin
C[m,n]=A[m,n]+B[m,n])
end;
end;
end
else
```

```
writeln('DIMENSI MATRIK TIDAK COCOK')
end;
```

- Pengurangan Matrik

```
procedure check_matrik(A,B,C:matrik; m,n,p,q:integer);
var i,j :integer;
begin
if (m=p) and (n=q) then
begin
for i:=1 to m do
begin
for j:=1 to n do
begin
C[m,n]=A[m,n]- C[m,n])
end;
end;
end
else
writeln('DIMENSI MATRIK TIDAK COCOK')
end;
```

-. Perkalian Matrik

```
procedure perkalian_matrik(A,B,C:matrik; m,n,p,q:integer);
var i,j, k :integer;
C1: matrik;
begin
if (n=p) then
begin
```

```
for i:=1 to m do
begin
for j:=1 to p do
begin {inner product}
C1[i,j]:=0;
for k:=1 to n do
C1[i,j]:=C1[i,j]+A[i,k]*B[k,j];
end; {inner product}
end;
n:=q;
for i:=1 to m do
for j:=1 to n do
C[i,j]:=C1[i,j];
end
else
writeln('DIMENSI MATRIK TIDAK COCOK')
end;
```

- Transpose Matrik

```
procedure Transpose(A,B:matrik; m,n,p,q:integer);
var i,j:integer;
begin
for i:=1 to n do
begin
for j:=1 to m do
begin
B[m,n]=A[n,m]
end;
end;
end;
```

-. Mencari Elemen yang Kosong pada Matrik

```
procedure CHECK_ZERO_ELEMEN(A,matrik; m,n:integer);
var i,j:integer;
begin
for i:=1 to m do
begin
for j:=1 to n do
begin
if B[m,n]= 0 then
writeln (‘terdapat elemen yang kosong’)
else
writeln (‘tidak terdapat elemen yang kosong’)
end;
end;
end;
```

Latihan.

1.

N1	N2	N3	Total

Buat program Array 2 dimensi untuk memasukkan data nilai matakuliah tiap mahasiswa.

Kolom-→ N1, N2, N3, Total

Baris---→ Yani, Riski, Eko

2.

Belajar	Renang	Sepak bola	Musik

Baris--→senin, selasa, rabu, kamis, jum'at, sabtu

- Buat program Array 2 dimensi untuk memasukkan jumlah anak yang ikut kegiatan tertentu.

- Outputkan dalam bentuk matrik 2 dimensi.

MODUL 4

RECORD (REKAMAN)

Sebuah record rekaman disusun oleh beberapa field. Tiap field berisi data dari tipe dasar / bentukan tertentu. Record mempunyai kelebihan untuk menyimpan suatu sekumpulan elemen data yang berbeda-beda tipenya (di banding array). Contoh , sebuah record dengan empat buah field.

Field 1 | Field 2 | Field 3 | Field 4

Cara pendeklarasian dari record adalah sbb:

• Mendefinisikan tipe dari record (jumlah field, jenis tipe data yang dipakai),

• Mendefinisikan variabel untuk dilakukan operasi.

SYNTAX

```
type

nama_record = record
identifier_1 : tipe_data_1;
:
:
identifier_n : tipe_data_n;
end;
var variabel : nama_record;
```

Contoh.

```
type
```

```
Data_mahasiswa = record
Nama : string;
Usia : integer;
Kota : String;
Kodepos : integer;
end;
Var
x: Data_mahasiswa;
```

1. Pengaksesan Elemen Record

Nama variable disertai nama field.

```
x.Nama
x.Usia
x.Kota
x.Kodepos
```

Contoh.

```
program RECORD_INTRO;
type tanggal = record
bulan, hari, tahun : integer;
end;
var waktu : tanggal;
begin
waktu.hari :=25;
waktu.bulan:=09;
waktu.tahun:= 1983;
writeln('hari ini adalah ',waktu.hari,':',waktu.bulan,':', waktu.tahun)
end.
```

2. Pengunaan With … do

Pernyataan with untuk lebih menyederhanakan pengaksesan field-field pada record. Pemrograman dapat mengakses field cukup dengan menyebutkan nama field-nya saja. Misalkan pernyataan :

```
x.Nama
x.Usia
x.Kota
x.Kodepos
```

menjadi

```
with x do
Begin
Nama
Usia
Kota
Kodepos
end
```

Contoh.

```
program RECORD_INTRO;
type tanggal = record
bulan, hari, tahun : integer;
end;
var waktu : tanggal;
begin {program utama}
with waktu do {mulai with}
begin
hari :=25;
bulan:=09;
```

```
tahun:=1983;
writeln('hari ini adalah ',hari,':',bulan,':', tahun)
end {akhir with}
end.
```

3. Array dari Record

Suatu array dapat juga berisi record contoh suatu deklarasi record tanggal.

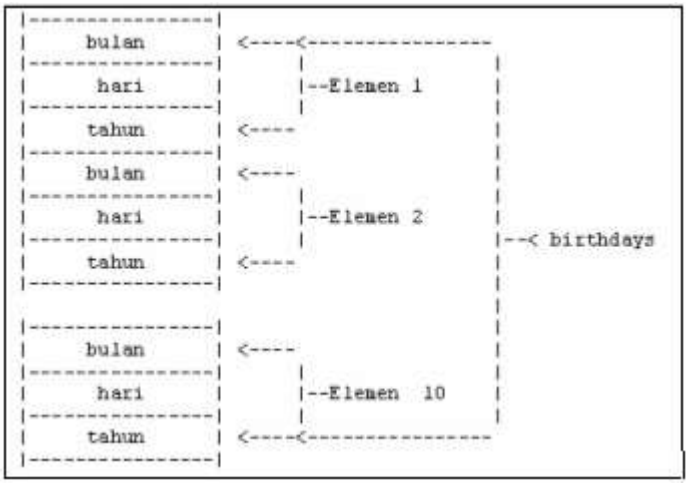
```
type tanggal = record
bulan, hari, tahun : integer;
end;
var waktu : tanggal;
```

kemudian kita membentuk suatu array dari record ini, namakan birthdays.

```
var birthdays : array[1..10] of tanggal;
```

pernyataan ini akan membentuk suatu array dengan 10 elemen. Dimana tiap elemen adalah sebuah record tanggal, yaitu, terdiri atas bulan, hari, tahun dengan tipe data Integer.

Digambarkan seperti berikut:



Contoh Pemberian nilai awal dari masing-masing elemen birthdays:

```
Birthdays[1].hari :=25;
Birthdays[1].bulan:=09;
Birthdays[1].tahun:=1983;
```

4. Record di dalam Record

Record bisa berisi record lain sebagai field. Seperti contoh record tanggal dan jam dikombinasikan menjadi sebuah record saat ini,

```
type tanggal = record
bulan, hari, tahun : integer;
end;
type waktu =record
jam, menit, detik : integer;
end;
type waktu_ini =record
tanggal_ini : tanggal;
waktu_ini : waktu
end;
```

Kemudian kita perlu membuat variabel kerja

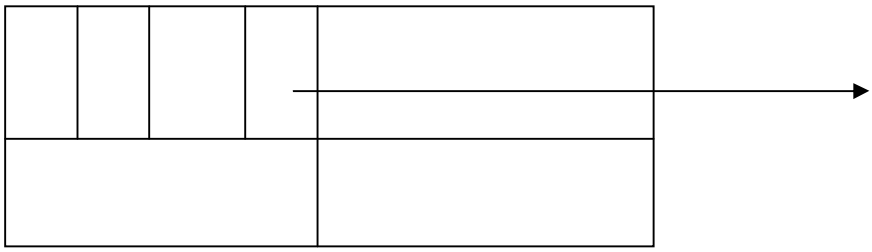
```
var saat_ini : waktu_ini;
```

pemberian nilai akan terjadi seperti di bawah ini:

```
saat_ini.tanggal.bulan:= 11;
saat_ini.tanggal.hari:= 2;
saat_ini.tanggal.tahun:= 1985;
saat_ini.waktu.jam:= 3;
saat_ini.waktu.menit:= 3;
saat_ini.waktu.detik:= 33;
```

Latihan.

1.



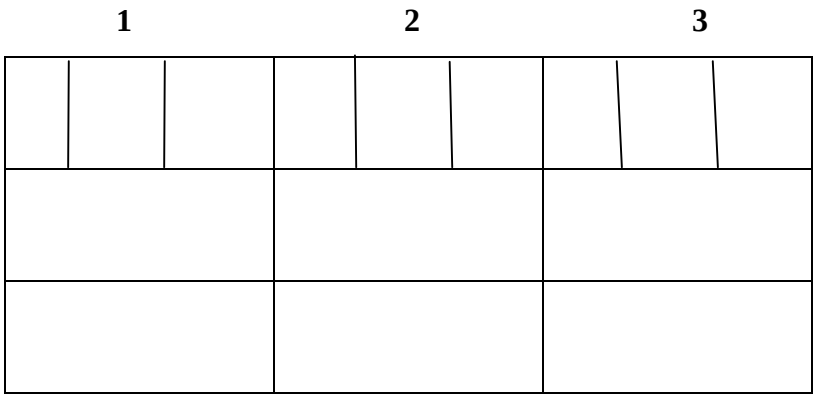
Nama Record---→ Barang

Nama Field-----→ Kd_barang, Nama, Harga, Jumlah

Buat program untuk input dan output dengan menggunakan metode mengakses record no.1 (nama variabel disertai nama field).

2. Dengan gambar yang sama, buat program untuk input dan output dengan menggunakan metode mengakses record no.2 (menggunakan with .. do).

3.



Nama array---→ Mahasiswa

Nama Record-→ Mhs

Nama variable--→ M

Nama Field--→ NIM, Nama, Alamat

Buat program untuk input dan output dengan menggunakan metode mengakses record dalam Array 1 Dimensi. (Gunakan perulangan dan with..do).

4.

1	**2**	**3**

Nama array---→ Nilai

Nama Record-→Data_nilai

Nama variable--→ N

Nama Field--→ N1, N2, N3

Ordo----------→ 3x3

Buat program untuk input dan output dengan menggunakan metode mengakses record dalam Array 2 Dimensi. (Gunakan perulangan dan with..do).

MODUL 5

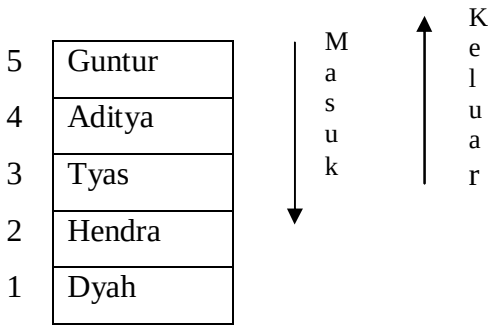
STACK (Tumpukan)

-→ Adalah tumpulan data yang seolah-olah ada data di atas data lain.

-→ Suatu metode untuk Input dan hapus di dalam memori komputer.

Konsep utama dalam STACK adalah LIFO (Last In First Out).

Contoh:



Data nomor 1 datang/masuk duluan, data nomor 5 yang paling atas yang keluar terlebih dahulu.

Algoritma:

1. Input/tambah data
    - Jika ada input maka no stack/no tumpukan yang semula 0 akan tambah 1 demi 1 sampai maksimal tumpukan.

2. Pengambilan data
    - Jika ada pengambilan data maka data dipindahkan di variabel lain contohnya temp. Dan posisi tumpukannya yang semula maksimal akan berkurang 1 demi 1 sampai posisi 0 kembali.

1. Deklarasi STACK

```
Type
        Const
        Max = 5;
        Nama record = Record
                Data    : type data;
                Top     : byte;
        End;
        Nama_array = ARRAY [1..max] of Nama record;
Var
        STACK : nama Array;
```

1	2	3	4
Data Top Dyah 1			

Nama Array-----→ Barang

Nama Record---→ Coba

Nama Variabel--→ Stack

Contoh Deklarasi dari gambar diatas:

```
Type
        Coba = record
        Data    :string;
        Top     : byte;
        End;
```

```
Barang = ARRAY [1..4] of coba;
Var
        Stack:barang;
```

2. Operasi pada STACK
   - CREATE

     Membuat stack baru yang masih kosong.

```
Procedure create;
Begin
    Stack.top:=0;
End;
```

   - FULL

     Untuk memeriksa apakah stack sudah penuh atau belum.

```
Fuction full:bolean;
Begin
    Stack.top:=max;
End;
```

   - PUSH

     Menambah sebuah elemen (data) kedalam stack

     Syarat: tidak bisa dilakukan jika stack sudah penuh.

```
Procedure push ( input:string );
Begin
    If not full then
    Begin
            Stack.top:=stack.top;
```

```
            Stack.data:=input;
        End;
    End;
```

- EMPTY

```
Fuction empty: bolean;
Begin
    Empty:=false;
    If top:=0 then empty:=true;
End;
```

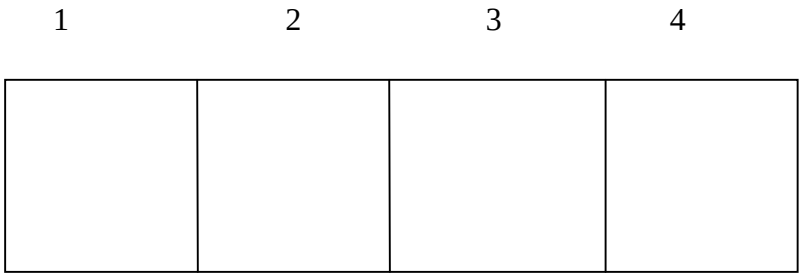
- POP

Mengambil elemen teratas dari stack.

Syarat: Stack tidak boleh kosong.

```
Procedure Pop ( elemen:string );
Begin
    If not empty then
    Begin
    Elemen:=stack.data;
    Stack.top:=top – 1;
    End;
End;
```

Contoh:

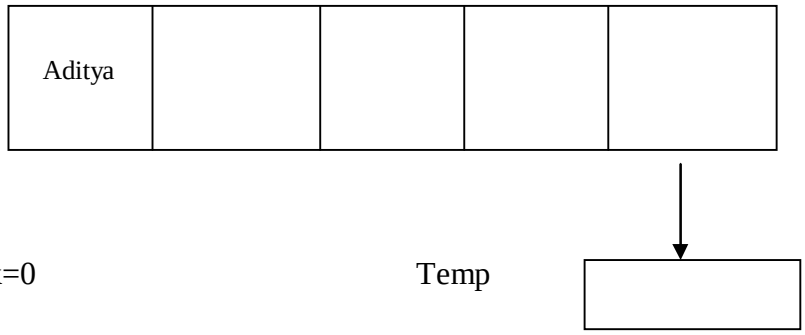



```
Uses wincrt;
Type
kelas = ARRAY[1..4] of string;
Var
        Stack: kelas;
        top:byte;
        Elemen: string;
        I : integer;
Begin
top:=0;
For i:=1 to 4 do
Begin
        Writeln('masukkan nama ke', '  ',i,'  ','='); readln(stack[i]);
      top:=top+1;
End;
writeln('posisi tumpukan=',top);
Writeln('pengambilan data');
For i:=1 to 4 do
Begin
        Elemen:=stack[i];
        top:=top - 1;
End;
writeln;
```

```
Writeln('data elemen sekarang=',elemen);
writeln('posisi tumpukan=',top);
Readln;
End.
```

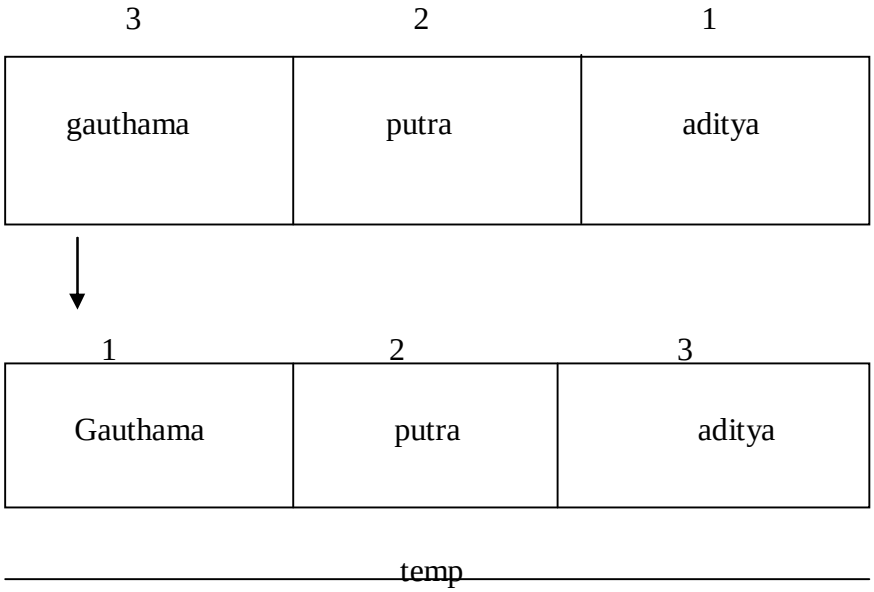
Latihan.

1.



Buat program untuk menambah dan mengambil data dari stack.

2.



Buat program untuk menambah dan mengambil data kemudian diletakkan pada array yang bernama temp.

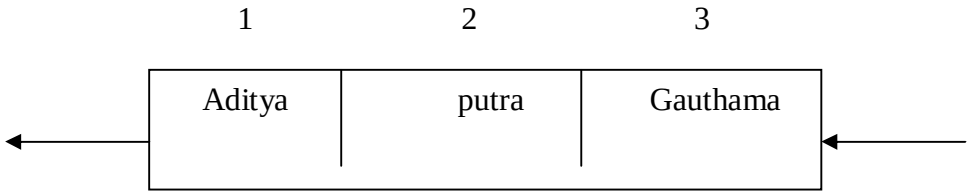
MODUL 6

QUEUE (ANTRIAN)

-→ Kumpulan data dimana data masuk dan keluar pada ujung yang berbeda.

-→ Konsep utama FIFO (Fisrt In First Out).

Contoh:



Data nomor 1 datang/masuk dan keluar duluan.

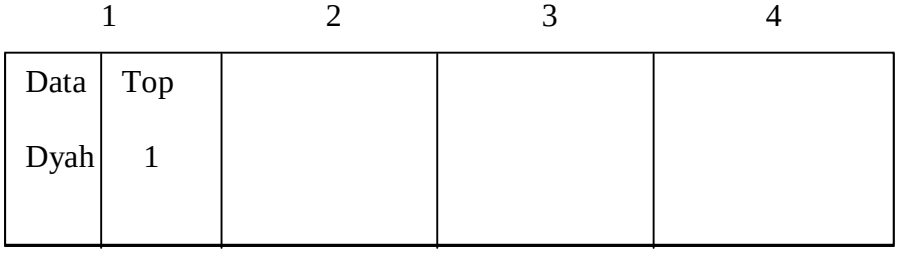
Algoritma:

1. Input/tambah data
    - Jika ada input maka no antrian yang semula 0 akan tambah 1 demi 1 sampai maksimal antrian.

2. Hapus/Pengambilan data
    - Jika ada pengambilan data maka data dipindahkan di variabel lain contohnya temp, antrian ke-dua akan maju ke antrian pertama dan seterusnya. Dan jumlah antrian yang semula maksimal akan berkurang 1 demi 1 sampai antrian 0 kembali.

3. Deklarasi Queue

```
Type
        Const
        Max = 5;
        Nama record = Record
                Data    : type data;
                Top     : byte;
        End;
        Nama_array = ARRAY [1..max] of Nama record;
Var
        Antri : nama Array;
```



Nama Array-----→ Barang

Nama Record---→ Coba

Nama Variabel--→ Antri

Contoh Deklarasi dari gambar diatas:

```
Type
        Coba = record
        Data    :string;
        Top     : byte;
        End;
Barang = ARRAY [1..4] of coba;
Var     Antri:barang;
```

4. Operasi pada queue

- CREATE

  Membuat antrian baru yang masih kosong.

```
Procedure create;
Begin
    antri.top:=0;
End;
```

- FULL

  Untuk memeriksa apakah antrian sudah penih..

```
Fuction full:bolean;
Begin
    antri.top:=max;
End;
```

- PUSH

  Menambah sebuah elemen (data) kedalam antrian.

  Syarat: tidak bisa dilakukan jika antrian sudah penuh.

```
Procedure push ( input:string );
Begin
    If not full then
    Begin
            antri.top:=antri.top;
            antri.data:=input;
    End;
End;
```

- EMPTY

  Fuction empty: bolean;

  Begin

  Empty:=false;

  If top:=0 then empty:=true;

  End;

- POP

  Mengambil 1 elemen dari sebuah antrian.

  Syarat: antrian tidak boleh kosong.

  Procedure Pop (elemen:string);

  Begin

  If not empty then

  Begin

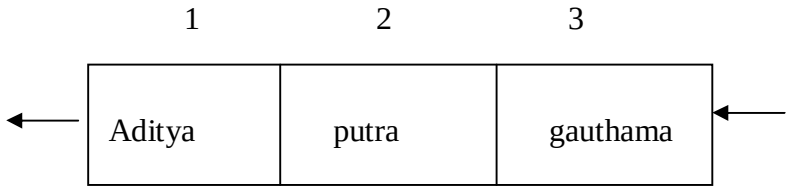
  Elemen:=antri.data;

  antri.top:=top – 1;

  End;

  End;

Contoh:



```
Program antrian_1;
Uses wincrt;
Type
        Data= array [1..3] of string;
Var
        D: data;
        I, antri: integer;
        Temp: string;
Begin
Antri:=0;
{untuk input}
For I:=1 to 3 do
Begin
Writeln('masukkan nama ke',' ',i);
Readln(d[i]);
Antri:=antri+1;
End;
{untuk Output}
For I:=1 to 3 do
Begin
Temp:=d[i];
Antri:=antri-1;
End;
{lihat output di var temp setelah pengambilan }
Writeln('hasil var temp=',temp);
Readln;
End.
```

MODUL 7
POINTER

Variabel Pointer

Pada materi sebelumnya telah dijelaskan mengenai variabel bertipe array, suatu tipe data yang bersifat statis (ukuran dan urutannya sudah pasti). Selain itu ruang memori yang dipakai olehnya tidak dapat dihapus bila variabel bertipe array tersebut sudah tidak digunakan lagi pada saat program dijalankan. Untuk memecahkan masalah diatas, kita dapat menggunakan variabel pointer. Tipe data pointer bersifat dinamis, variabel akan dialokasikan hanya pada saat dibutuhkan dan sesudah tidak dibutuhkan dapat dialokasikan kembali.

Array vs Pointer

Berikut tabel di bawah ini diberikan perbedaan antara variabel bertipe array dengan varibel bertipe pointer.

Kriteria	Array	Pointer
Sifat	Statis	Dinamis
Ukuran	Pasti	Sesuai kebutuhan
Alokasi variabel	Saat program dijalankan sampai selesai	Dapat diatur sesuai kebutuhuan

Deklarasi Variabel Pointer

Bentuk umum :

Var <namavar> : <^tipedata>

Contoh :

```
Var
   Jumlahdata  : ^integer;
   Namasiswa  :^string[25];
  Nilaisiswa     :^real;
```

Pendeklarasian variabel pointer tidak jauh berbeda dengan pendeklarasian variabel biasa, hanya perlu ditambahan simbol topi (^) sebelum tipe datanya. Simbol topi tersebut menandahkan bahwa variabel tersebut menunjuk ke lokasi tertentu pada memori.

Anda juga dapat membuat variabel pointer bertipe record yang anda definisikan sendiri. Pendeklarasiannya adalah seperti berikut ini.

Bentuk umum :

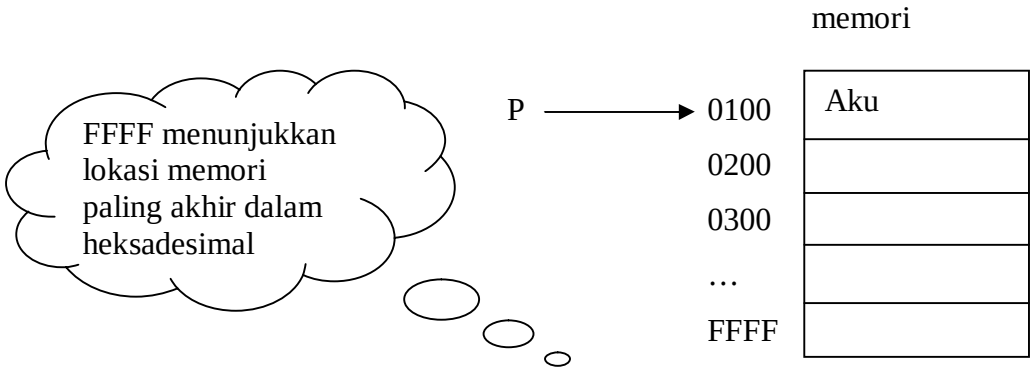
```
Tipe
    <namapointer> = <^namarecord>;
    <namarecord> =  record
                        <item1>:<tipedata1>;
                        <item2>:<tipedata2>;
                            …
                            <itemN>:<tipedataN>;
                        end;
Var
    <namavar>:<namapointer>;
```

contoh :

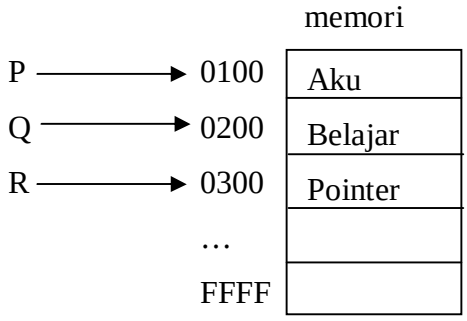
```
Type
    PointMhs = ^RecMhs;
    RecMhs   = record
                    Nama : string[25];
                    NIM  : string[10];
                    Alm  : string[30];
                    IPK  : real;
                   End;
var
    datamahasiswa : pointMhs;
```

Varibel Biasa vs Variabel Pointer

Variabel Pointer adalah suatu variabel yang menunjuk ke alamat memori yang digunakan untuk menampung data yang akan diproses, seperti digambarkan dibawah ini:



P adalah variabel pointer yang menunjuk ke alamat memori 100 yang berisi data bertipe string “Aku”. Apabila anda ingin menambah data dengan menggunakan variabel yang berbeda, maka anda dapat mendeklarasikan variabel pointer baru misalnya Q dan R dst sehingga tampak sbb :

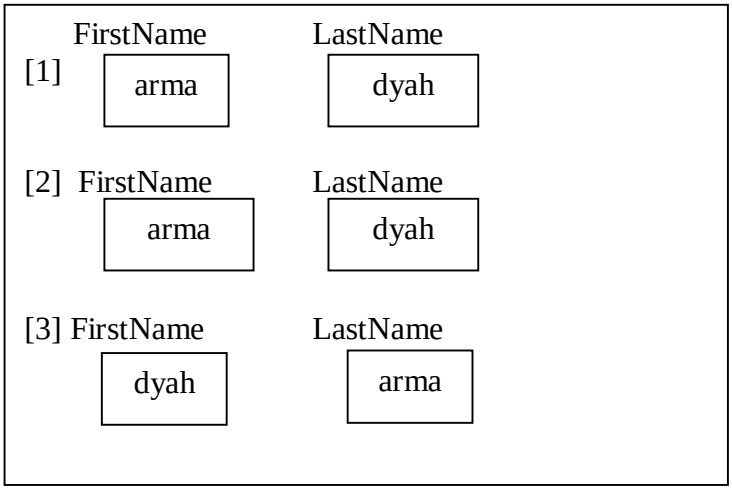


Untuk membedakan antara variabel pointer dengan variabel biasa, perhatikan contoh berikut :

Variabel Biasa

Var

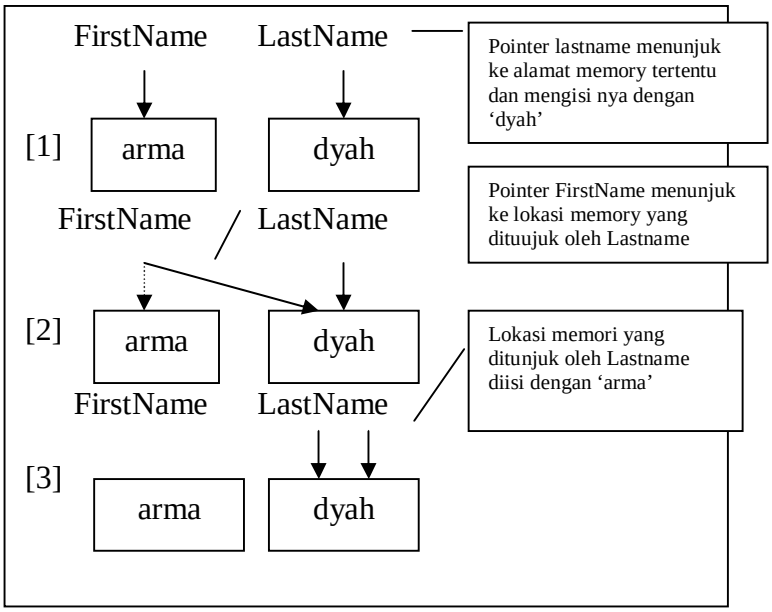
```
FirstName, LastName :
String;
begin
[1] FirstName:=’arma’;
    LastName:=’dyah’;
[2] FirstName:=LastName;
[3] LastName:=’dyah’;
    Writeln(FirstName);
```



Variabel Pointer

Var

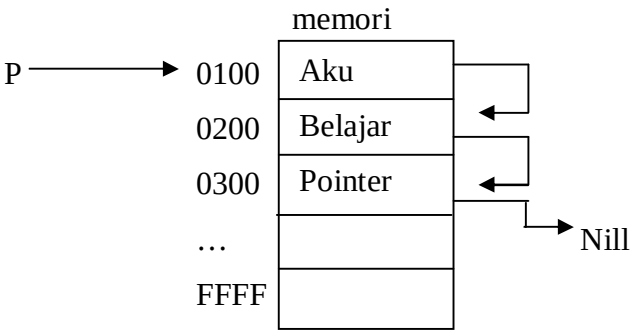
```
FirstName, LastName :
^String;
begin
[1] FirstName^:=’arma’;
    LastName^:=’dyah’;
[2] FirstName^:=LastName;
[3] LastName^:=’dyah’;
    Writeln(FirstName^);
    Writeln(LastName^);
End.
```



Single Linked List

Apabila setiap kali anda ingin menambahkan data selalu dengan menggunakan variabel pointer yang baru, anda akan membutuhkan banyak sekali variabel pointer(penunjuk).

Oleh karena itu ada baiknya jika anda hanya menggunakan satu variabel pointer saja untuk menyimpan banyak data dengan metode yang kita sebut ***Linked List***. Jika diterjemahkan, maka berarti suatu daftar isi yang saling berhubungan. Untuk lebih jelasnya perhatikan gambar di bawah ini :



Pada gambar diatas tampak bahwa sebuah data terletak pada sebuah lokasi memory area. Tempat yang disediakan pada suatu area memory tertentu untuk menyimpan data dikenal dengan sebutan **node/simpul**. Pada setiap node memiliki pointer(penunjuk) yang menunjuk ke simpul berikutnya sehingga terbentuk suatu untaian dan dengan demikian hanya diperlukan sebuah variabel pointer. Susunan berupa untaian semacam ini disebut **Single Linked List**. (ket: Nill tak memiliki nilai apapun. Biasanya linked list pada titik akhirnya akan menunjuk ke Nill).

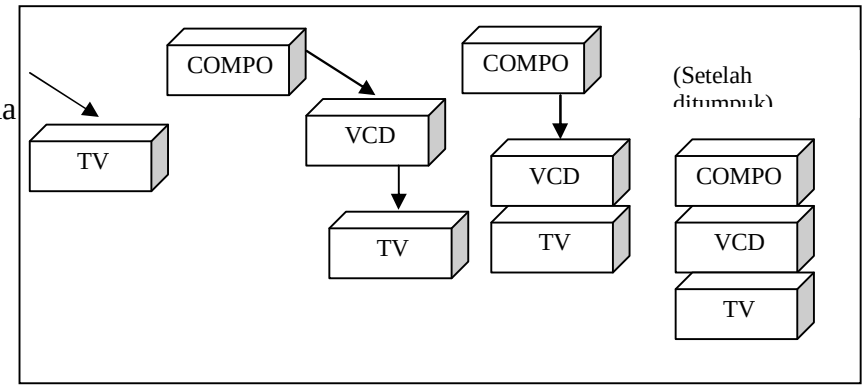
Dalam pembuatan single linked list dapat menggunakan 2 metode :

- LIFO (Last In First Out), aplikasi : Stack(Tumpukan).
- FIFO (First In First Out), aplikasi : Queue(Antrian).

LIFO (Last In First Out)

LIFO adalah suatu metode pembuatan linked list, dimana data yang masuk paling akhir adalah data yang keluar paling awal. Hal ini dapat dianalogikan (dalam kehidupan sehari-hari) pada saat anda menumpuk barang, seperti digambarkan di bawah ini :

(keadaan mula-mula adalah kosong)



Pembuatan sebuah simpul dalam suatu linked list seperti digambarkan diatas disebut dengan istilah INSERT. Jika linked list dibuat dengan metode LIFO, maka terjadi penambahan/insert simpul di **belakang**.

PROSEDURE INSERT

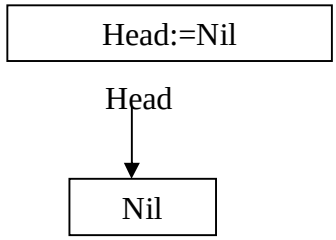
Istilah insert berarti menambahkan sebuah simpul baru ke dalam suatu linked list. Berikut di bawah ini adalah penggalan listing prosecure insert untuk LIFO berserta contoh dan penjelasan cara kerjanya.

```
Type
 Point     = ^RecPoint;
 Recpoint = Record
                  Isi       : TipeData;
                  Next     : Point;
               End;
Var
 Head, Tail, Now : Point;
```

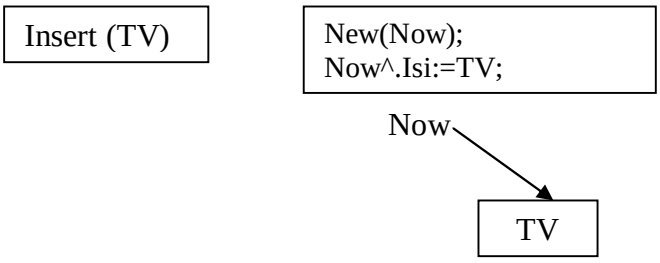
Penggalan deklarasi tipe data dan variabel di atas ini akan dipakai pada penjelasan procedure-procedure selanjutnya.

```
Procedure INSERT(elemen:TipeData);
Var Now : Point;
 Begin
  New(Now);
  Now^.Isi:=elemen;
  If Head=Nil then
        Now^.Next:=Nil;
  Else
        Now^.Next:=Head;
  Head:=Now;
End;
```

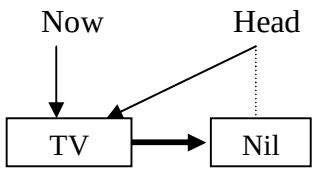
Penggalan procedure INSERT untuk LIFO

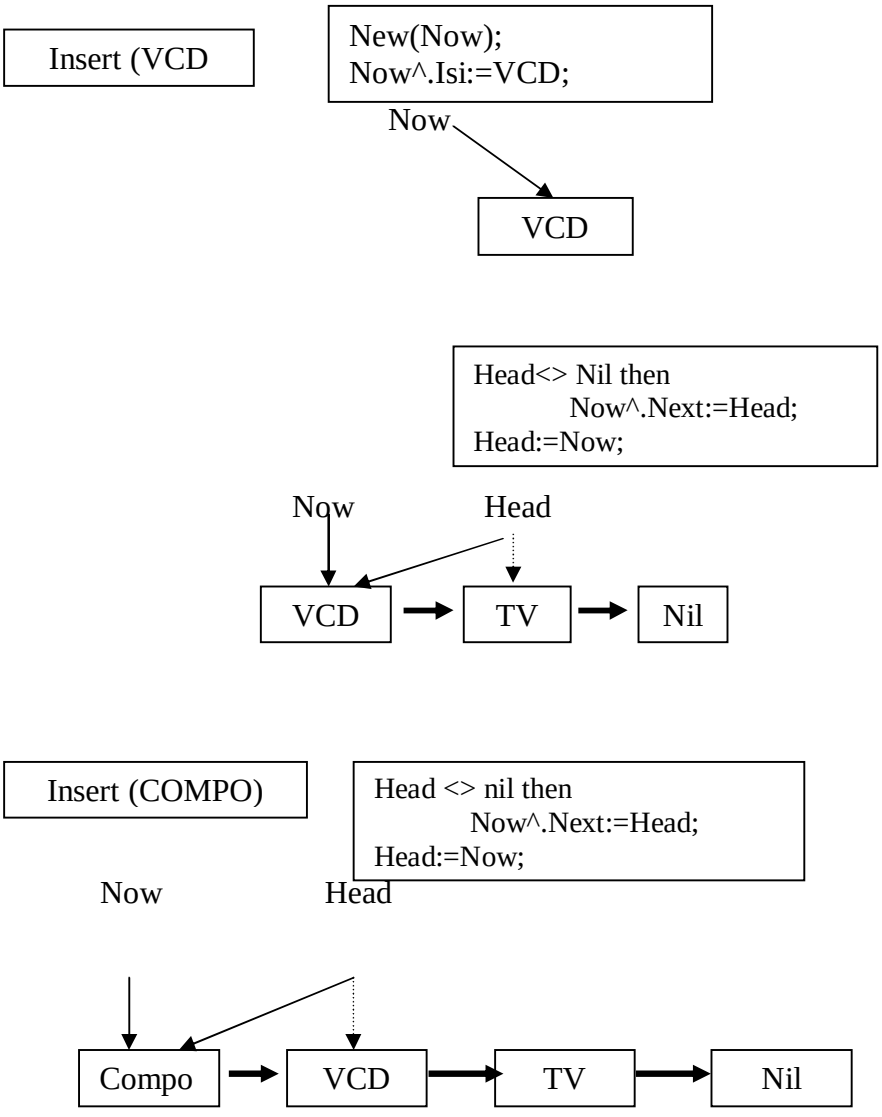


(ket : head mula-mula selalu dideklarasikan sebagai nil)




```
Head=Nil then
        Now^.Next:=Nil;
Head:=Now;
```




Insert (VCD
New(Now);
Now^.Isi:=VCD;
Now
VCD
Head<> Nil then
Now^.Next:=Head;
Head:=Now;
Now
Head
VCD
TV
Nil
Insert (COMPO)
Head <> nil then
Now^.Next:=Head;
Head:=Now;
Now
Head
Compo
VCD
TV
Nil

MODUL 8

FIFO (First In First Out)

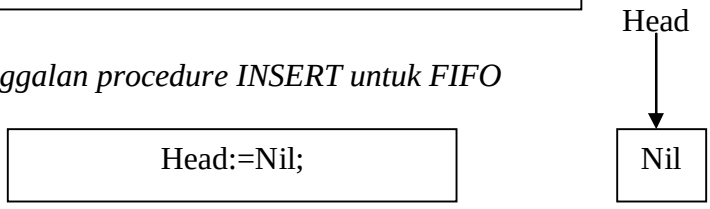
FIFO adalah suatu metoda pembuatan Linked List dimana data yang masuk paling awal adalah data yang keluar paling awal juga. Hal ini dapat dianalogikan (dalam kehidupan sehari-hari) misalkan saat sekelompok orang yang datang (ENQUEUE) mengantri hendak membeli tiket di loket.

Jika Linked List dibuat dengan metode FIFO, maka terjadi penambahan/Insert simpul di depan.

PROCEDURE INSERT

```
Procedure INSERT(elemen:TipeData);
Var Now:Point;
        Begin
           New(Now);
           If head = nil then
                 Head:=now
           else
                 Tail^.next:=now;
           Tail:=Now;
            Tail^.next:=nil;
             Now^.isi:=elemen;
End;
```

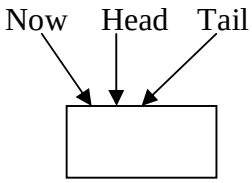
Penggalan procedure INSERT untuk FIFO



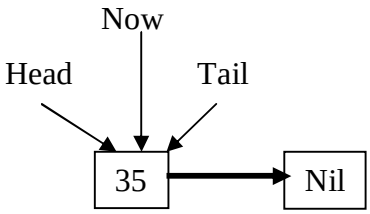
{*head mula-mula selalu diidentifikasikan sebagai nil*}

Insert (35)

```
New(Now);
Head=Nil then
        Head:=Now;
Tail:=Now;
```

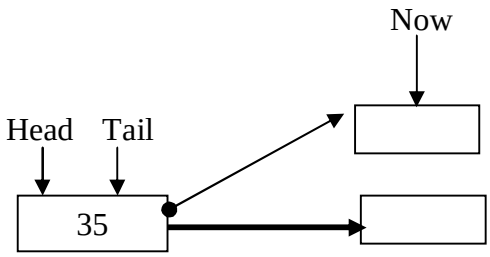



```
Tail^.next:=nil;
Now^.isi:=35;
```

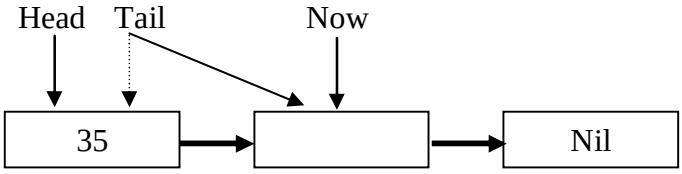


Insert (5)

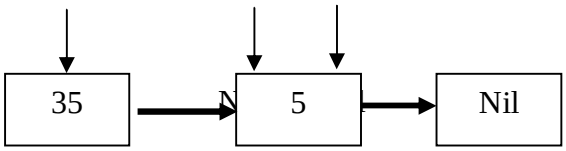
```
New(Now);
Head<>nil then
        Tail^.next:=now;
```



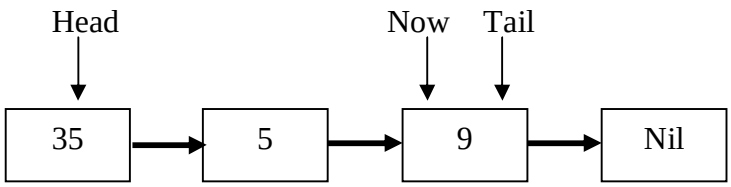

```
Tail:=Now;
Tail^.next:=nil;
```



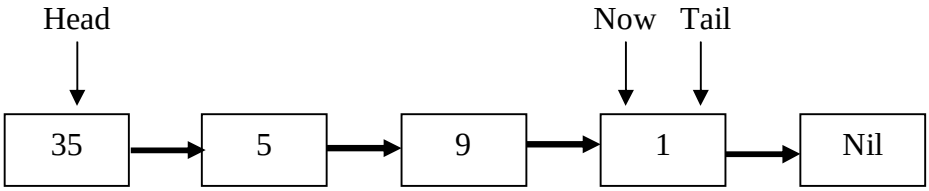

```
Now^.isi=5;
```



Insert (9)



Insert (1)



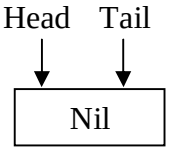
Procedure dan Function Linked List Lainnya

Selain procedure insert di atas, pada linked list juhga terdapat procedure serta function lainnya.

Di bawah ini diberikan procedure-procedure serta function umum dalam aplikasi Linked List.

❖ Create : Membuat sebuah linked list yang baru dan masih kososng. (ket: procedure ini wajib dilakukan sebelum menggunakan linked list)

```
Procedure Create;
        Begin
                Head:=nil;
                Tail:=nil;
        End;
```

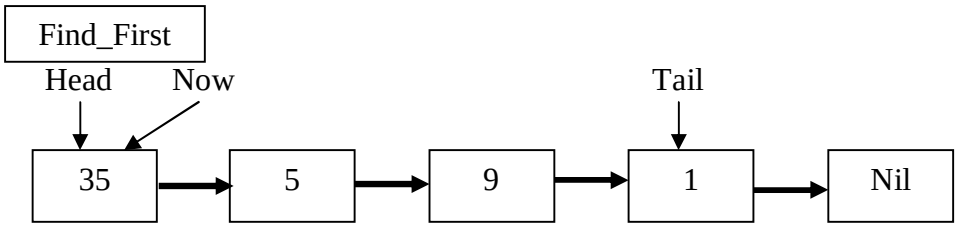


❖ Empty : Function untuk menentukan apakah linked list kosong atau tidak.

```
Function Empty : Boolean;
 Begin
        If head = nil then
                Empty:= true
        else
                empty:= false;
 end;
```

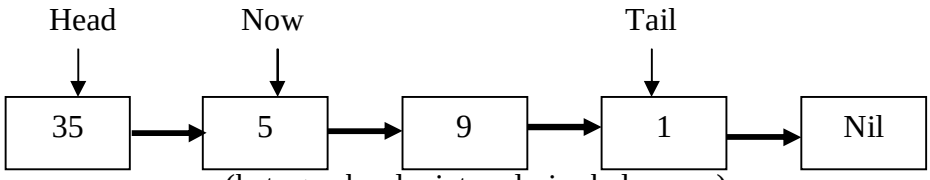
❖ Find First : Mencari elemen pertama dari linked list

```
Procedure Find_First;
        Begin
                Now:= head;
        End;
```



❖ Find Next : Mencari elemen sesudah elemen yang ditunjuk now.

```
Procedure Find_Next;
        Begin
                If Now^.next <> nil then
                   Now:= Now^.next;
        End;
```



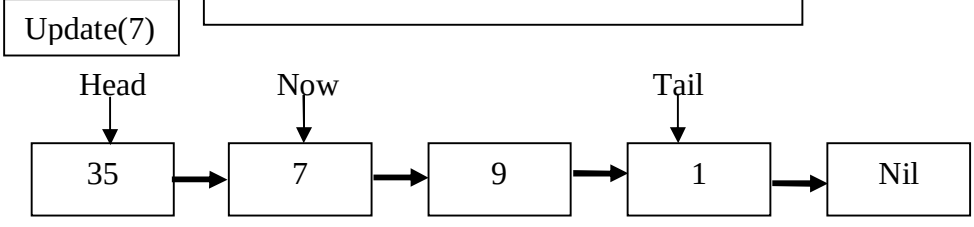
(ket: gambar lanjutan dari sebelumnya)

❖ Retrieve : Mengambil elemen yang ditunjuk oleh now. Elemen tersebut lalu ditampung pada suatu variabel (di bawah dicontohkan variabel r).

```
Procedure Retrieve(var r: TipeData );
 Begin
        R:= Now^.isi;
 End;
```

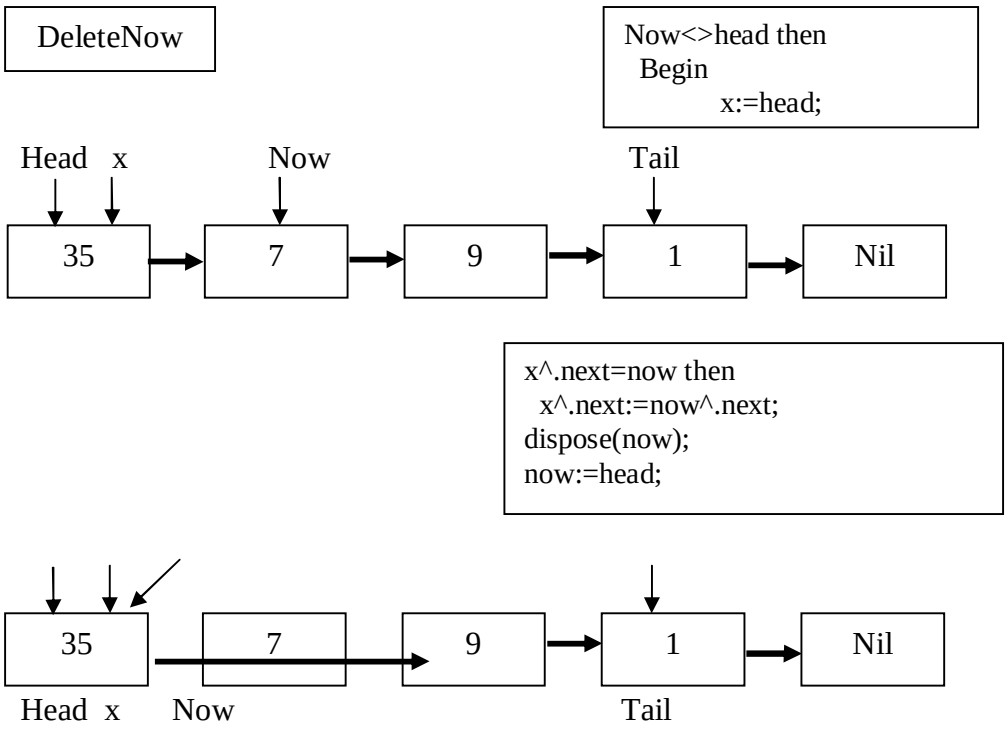
❖ Update : Mengubah elemen yang ditunjuk oleh now dengan isi dari suatu variabel (di bawah dicontohkan variabel u).

```
Procedure Update(u: TipeData );
 Begin
         Now^.isi:=u;
 End;
```



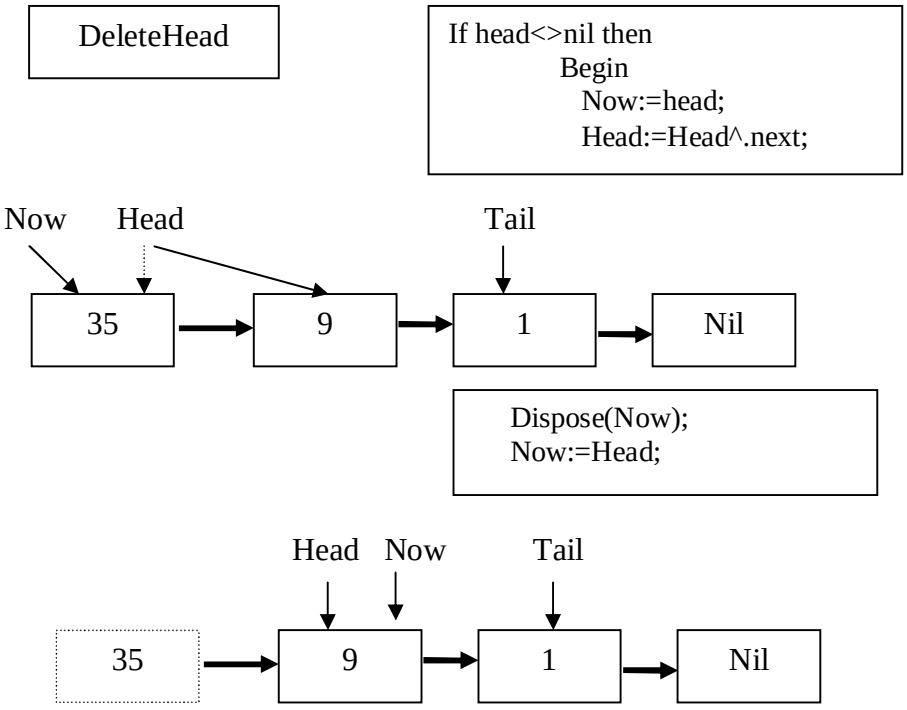
❖ Delete Now : Menghapus elemen yang ditunjuk oleh now. Jika yang dihapus adalah elemen pertama dari linked list(head), maka head akan berpindah ke elemen berikut.

```
Procedure DeleteNow;
Var x : point;
        Begin
                If now<>head then
                  Begin
                    x:=head;
                    while x^.next<>now do
                          x:=x^.next;
                    x^.next:=now^.next;
                   end
                else head:= head^.next;
                          dispose(Now);
                Now:= head;
        End;
```



❖ Delete Head : Menghapus elemen yang ditunjuj head. Head berpindah ke elemen sesudahnya.

```
Procedure DeleteHead;
  Begin
        If head<>nil then
          Begin
            Now:=head;
            Head:=Head^.next;
            Dispose(Now);
            Now:=Head;
        End;
  End;
```

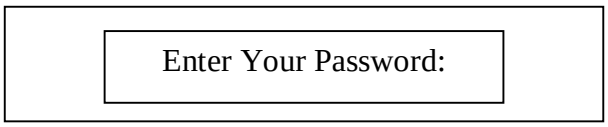


❖ Clear : Untuk menghapus linked list yang sudah ada.wajib dilakukan bila ingin mengakhiri program yang menggunakan linked list. Jika tidak data-data yang dialokasikan ke memori pada program sebelumnya akan tetap tertinggal di dalam memori.

```
Procedure Clear;
 Begin
  While head <> nil do
    Begin
      Now:=head;
      Head:=head^.next;
      Dispose(Now);
    End;
 End;
```

Latihan Soal beserta jawaban (Listing Program) dan penjelasan

Buatlah sebuah program untuk mendeteksi password/ kata sandi. Gunakan metode single linked list. Jika passwordnya benar, program akan selesai, jika salah maka user akan diminta memasukkan password kembali. (passw.pas)



Jawaban :

```
Uses crt;
Type
  Point  = ^Rec;
  Rec    = record
             Isi : char;
             Next : point;
           End;
Const
  Password = 'pascal'; {password yang harus dimasukkan}
Var
  i                    : byte;
  Tekan                : char;
  Passwd               : Boolean;
  Head, Tail, Now      : Point;


Procedure Create;
 Begin
        Head:=nil;
        Tail:=nil;
 End;
```

```
Procedure Push(isi:char);
Var Now:point;
 Begin
        New(now);              {membuat simpul baru}
        If Head=Nil then       {mendeteksi simpul awal}
        Begin
          Head:=Now;
          Tail:=Head;
        end else
        begin   {menyambung simpul yang baru pada simpul yang sudah ada}
          Tail^.next:=Now;
          Tail:=Tail^.Next;
          end;
        Now^.isi:=isi; {mengisi simpul yang baru}
        Now^.next:=Nil;
 end;
Function Check:Boolean; {function untuk mencheck input}
Var Temp : string[15];
 Begin
 Temp:=' '; Now:=Head;
 While Now <> nil do
        Begin
          Temp:=temp + Now^.isi;
          Now:=Now^.next;
        End;
 If temp <> Password then check:=False;
 End;
Procedure BuatBingkai(x1,y1,x2,y2:byte); {tampilan aplikasi}
Var i : byte;
 Begin
  GotoXY (x1,y1); write('┌'); GotoXY (x2,y1); write ('┐');
  GotoXY (x1,y2); write('└'); GotoXY (x2,y2); write ('┘');
  For i := x1+1 to x2-1 do
    Begin
        GotoXY (i,y1); write ('-');
        GotoXY (i,y2); write ('-');
    End;
  For i := y1+1 to y2-1 do
    Begin
        GotoXY(x1, i); write('│');
        GotoXY(x2, i); write ('│');
    End;
 End;

Procedure Pop; {procedure penghapus simpul}
```

```
 Begin
  Now:=head;
  head:=head^.next;
  While Now <> nil do
     Begin
          Dispose(Now);
          Now:=head;
          Head:=head^.next;
     End;
 End;
Begin {program utama}
 Repeat
   Create; I:=0;
   Repeat
          Tekan:=ReadKey;
          Write(Tekan);
          If Tekan <>#13 then Push(Tekan); inc(i);
    Until (Tekan= #13) or (I=10); {enter ditekan atau panjang = 10}
  Passwd:=check; Pop;
  Until Passwd;
End.
```

MODUL 9
TREE

Tree

Merupakan salah satu bentuk struktur data tidak linear yang menggambarkan hubungan yang bersifat hirarkis (hubungan one to many) antara elemen-elemen. Tree bisa didefinisikan sebagai kumpulan simpul/node dengan satu elemen khusus yang disebut Root dan node lainnya terbagi menjadi himpunan-himpunan yang saling tak berhubungan satu sama lainnya (disebut subtree). Untuk jelasnya, di bawah akan diuraikan istilah-istilah umum dalam tree :

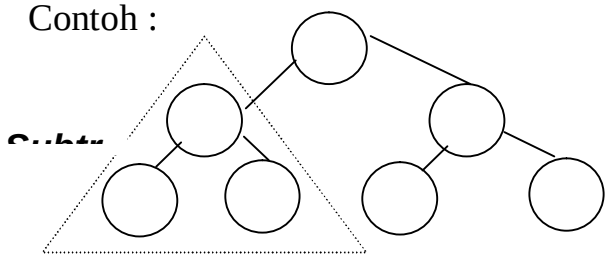
a) Prodecessor : node yang berada diatas node tertentu.
b) Successor : node yang berada di bawah node tertentu.
c) Ancestor : seluruh node yang terletak sebelum node tertentu dan terletak pada jalur yang sama.
d) Descendant : seluruh node yang terletak sesudah node tertentu dan terletak pada jalur yang sama.
e) Parent : predecssor satu level di atas suatu node.
f) Child : successor satu level di bawah suatu node.
g) Sibling : node-node yang memiliki parent yang sama dengan suatu node.
h) Subtree : bagian dari tree yang berupa suatu node beserta descendantnya dan memiliki semua karakteristik dari tree tersebut.
i) Size : banyaknya node dalam suatu tree.
j) Height : banyaknya tingkatan/level dalam suatu tree.

k) Root : satu-satunya node khusus dalam tree yang tak punya predecssor.

l) Leaf : node-node dalam tree yang tak memiliki seccessor.

m) Degree : banyaknya child yang dimiliki suatu node.

Contoh :

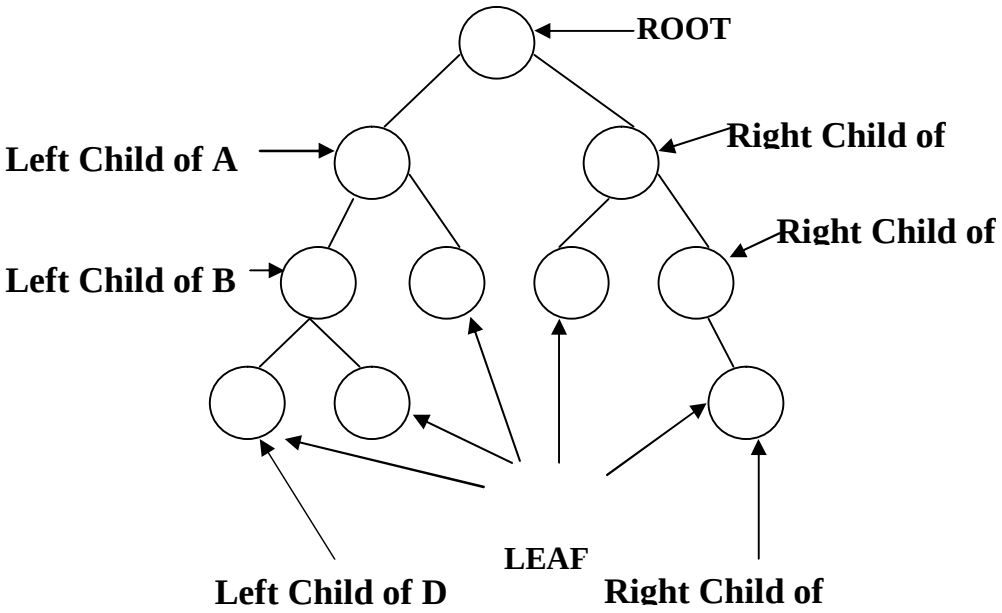


Ascestor (F) = C,A
Descendant (C) = F,G
Parent (D) = B
Child (A) = B,C
Sibling (F) = G
Size = 7
Height = 3
Root = A
Leaf = D,E,F,G
Degree (C) = 2

Beberapa jenis Tree yang memiliki sifat khusus :

1) Binary Tree

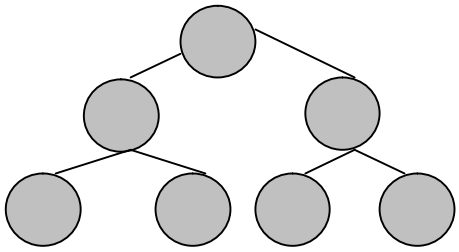
Binary Tree adalah tree dengan syarat bahwa tiap node hanya boleh memiliki maksimal dua subtree dan kedua subtree tersebut harus terpisah. Sesuai dengan definisi tersebut, maka tiap node dalam binary tree hanya boleh memiliki paling banyak dua child.



Jenis-jenis Binary Tree :

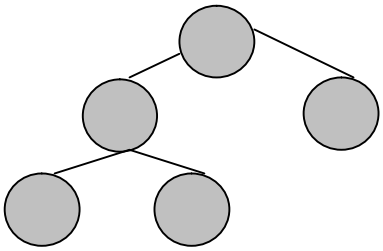
a) Full Binary Tree

Binary Tree yang tiap nodenya (kecuali leaf) memiliki dua child dan tiap subtree harus mempunyai panjang path yang sama.



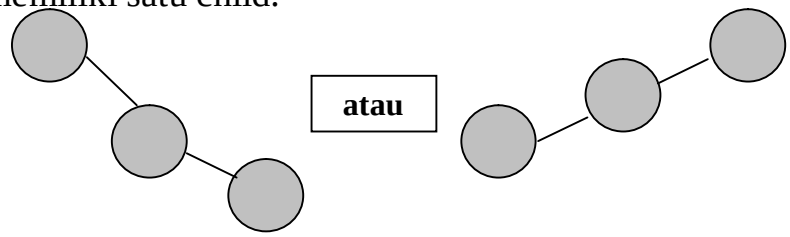
b) Complete Binary Tree

Mirip dengan Full Binary Tree, namun tiap subtree boleh memiliki panjang path yang berbeda. Node kecuali leaf memiliki 0 atau 2 child.



c) Skewed Binary Tree

akni Binary Tree yang semua nodenya (kecuali leaf) hanya memiliki satu child.



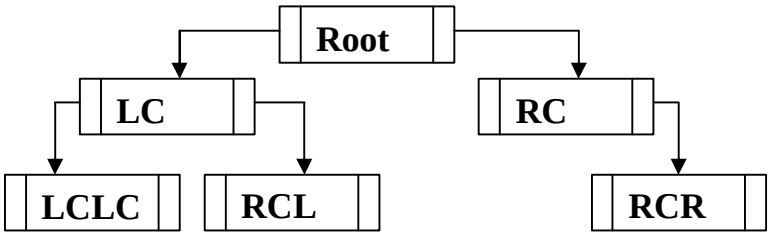
Implementasi Binary Tree

Binary Tree dapat diimplemntasikan dalam Pascal dengan menggunakan double Linked List. Untuk nodenya, bisa dideklarasikan sbb :

```
Type Tree    = ^node;
     Node  = record
                Isi         : TipeData;
                Left,Right : Tree;
```

end;

Contoh ilustrasi Tree yang disusun dengan double linked list :



(Ket: LC=Left Child; RC=Right Child)

Operasi-operasi pada Binary Tree :

- Create : Membentuk binary tree baru yang masih kosong.
- Clear : Mengosongkan binary tree yang sudah ada.
- Empty : Function untuk memeriksa apakah binary tree masih kosong.
- Insert : Memasukkan sebuah node ke dalam tree. Ada tiga pilihan insert: sebagai root, left child, atau right child. Khusus insert sebagai root, tree harus dalam keadaan kosong.
- Find : Mencari root, parent, left child, atau right child dari suatu node. (Tree tak boleh kosong)
- Update : Mengubah isi dari node yang ditunjuk oleh pointer current. (Tree tidak boleh kosong)
- Retrieve : Mengetahui isi dari node yang ditunjuk pointer current. (Tree tidak boleh kosong)
- DeleteSub : Menghapus sebuah subtree (node beserta seluruh descendantnya) yang ditunjuk current. Tree tak boleh

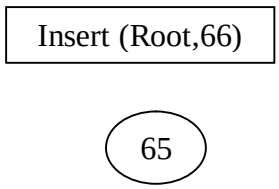
kosong. Setelah itu pointer current akan berpindah ke parent dari node yang dihapus.

- ❖ Characteristic : Mengetahui karakteristik dari suatu tree, yakni : size, height, serta average lengthnya. Tree tidak boleh kosong. (Average Length = [jumlahNodeLvl1*1+jmlNodeLvl2*2+…+jmlNodeLvln *n]/Size)
- ❖ Traverse : Mengunjungi seluruh node-node pada tree, masing-masing sekali. Hasilnya adalah urutan informasi secara linier yang tersimpan dalam tree. Ada tiga cara traverse : Pre Order, In Order, dan Post Order.

Langkah-Langkahnya Traverse :

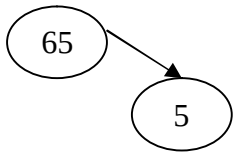
- ➢ PreOrder : Cetak isi node yang dikunjungi, kunjungi Left Child, kunjungi Right Child.
- ➢ InOrder : Kunjungi Left Child, Cetak isi node yang dikunjungi, kunjungi Right Child.
- ➢ PostOrder : Kunjungi Left Child, Kunjungi Right Child, cetak isi node yang dikunjungi.

Untuk lebih jelasnya perhatikan contoh operasi-operasi pada Binary Tree berikut ini :



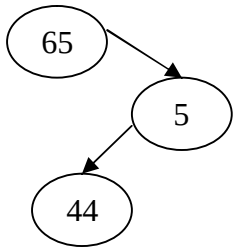
Memasukkan sebuah node ke dalam Tree yang masih kosong (Sebagai Tree)

Insert (RightChild,5)



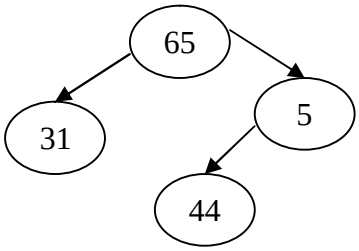
Menambahkan sebuah node sebagai right child dari Root.

Insert (LeftChild,44)



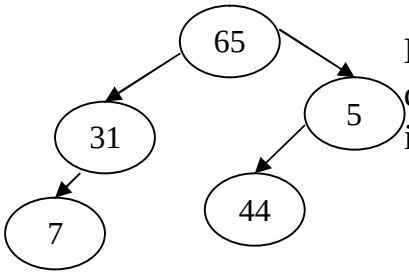
Menambahkan sebuah node sebagai left child dari node yang sebelunya di-insert.

Find (Root)
Insert (LeftChild,31)



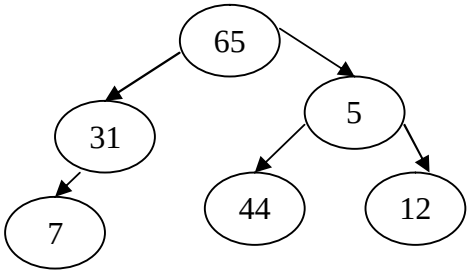
Memindahkan pointer ke Root kemudian menambhakan sebuah node sebagai left child dari root.

Insert (LeftChild,7)



Menambhakan sebuah node sebagai left child dari node yang sebelumnya di-insert.

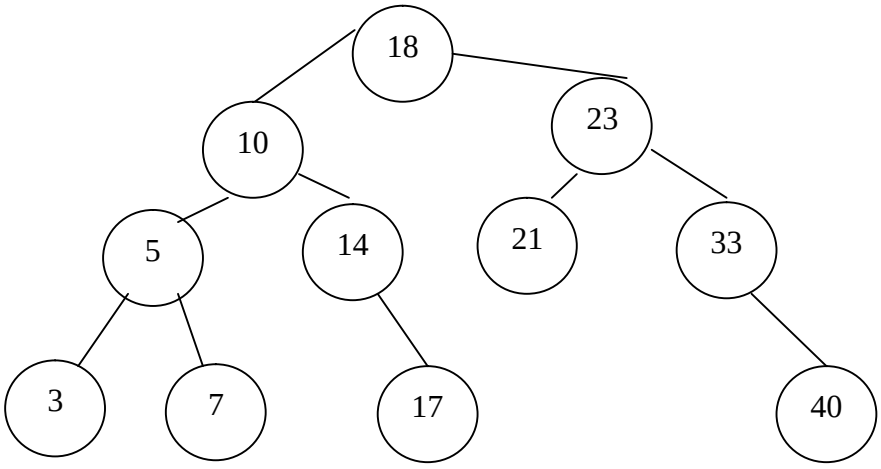
```
Find (Root)
Find (RightChild)
Insert (RightChild,12)
```



Memindahkan pointer ke Root, kemudian pindahkan lagi pointer ke right child dari Root, kemudian masukkan sebuah node sebagai right child dari node yang sedang ditunjuk oleh pointer.

2) Binary search Tree

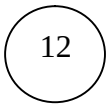
Adalah Binary Tree dengan sifat bahwa semua left child harus lebih kecil daripada right child dan parentnya. Juga semua right child harus lebih besar dari left child serta parentnya. Binary seach tree dibuat untuk mengatasi kelemahan pada binary tree biasa, yaitu kesulitan dalam searching / pencarian node tertentu dalam binary tree. Contoh binary search tree umum :



Pada dasarnya operasi dalam binary search tree sama dengan Binary tree biasa, kecuali pada operasi insert, update, dan delete.

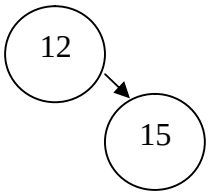
1. Insert : Pada Binary Search Tree, insert dilakukan setelah ditemukan lokasi yang tepat. (Lokasi tidak ditentukan oleh user sendiri).

Insert (12)



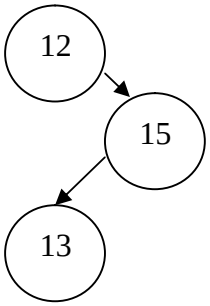
Memasukkan sebuah node yang berisi angka 12. Karena tree masih kosong, maka secara otomatis node tersebut menjadi root.

Insert (15)



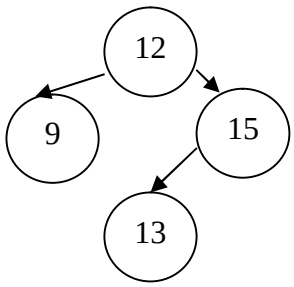
Karena 15 lebih besar dari 12, maka sesuai dengan peraturan harus berada disebelah kanan parent (right child)

Insert (13)



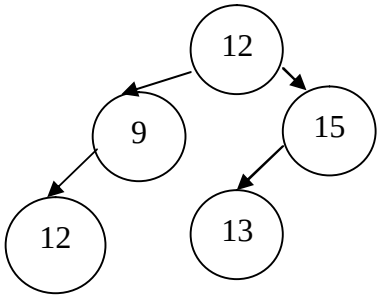
Karena 13 lebih besar dari 12, maka sesuai dengan peraturan harus berada di sebelah kanan parent (12), lalu bandingkan lagi dengan 15. karena lebih kecil maka 13 menempati left child dari 15.

Insert (9)

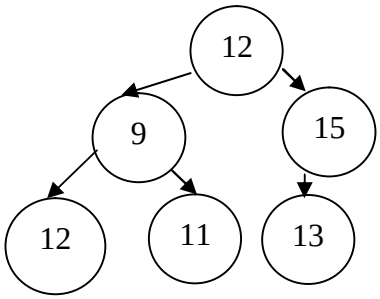


Proses yang sama seperti sebelumnya berlaku untuk selanjutnya.

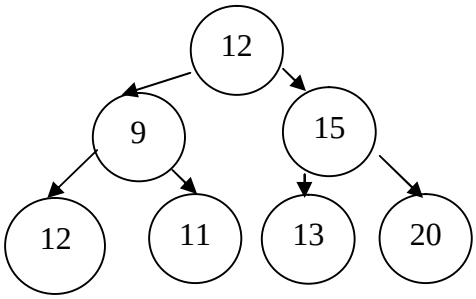
Insert (5)



Insert (11)



Insert (20)

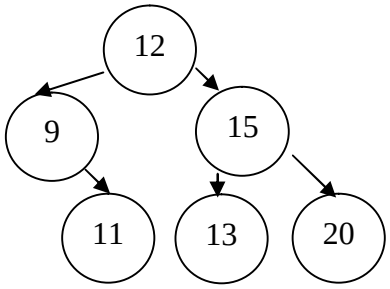


2. Update : Seperti pada Binary Tree biasa, namun disini uapte akan berpengaruh pada posisi node tersebut selanjutnya. Bila setelah diupdate mengakibatkan tree tersebut bukan Binary Search Tree lagi, maka harus dilakukan perubahan

pada tree dengan melakukan perubahan pada tree dengan melakukan rotasi supaya tetap menjadi Binary Search Tree.

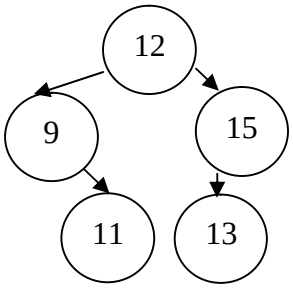
3. Delete: Seperti halnya update, delete dalam Binary Search Tree juga turut mempengaruhi struktur dari tree tersebut.

Delete (5) (*Keadaan awal merupakan lanjutan gambar sebelumnya*)



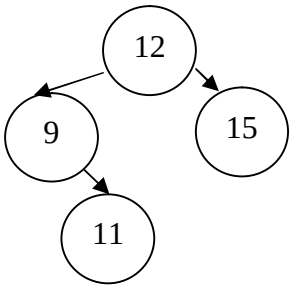
Karena node yang berisi 5 adalah leaf, maka ia dapat langsung dihapus.

Delete (20)



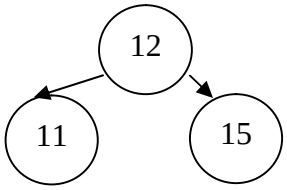
Karena node yang berisi 20 adalah leaf, maka ia dapat langsung dihapus.

Delete (13)



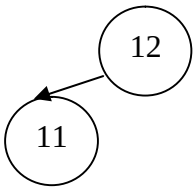
Pada operasi di samping, delete dilakukan terhadap Node dengan 2 child. Maka untuk menggantikannya, diambil node paling kiri dari Right SubTree yaitu 13.

Delete (13)



Pada operasi di samping, delete dilakukan terhadap node dengan 1 child. Maka child tersebut (11) akan menggantikan posisi dari node yang didelete (9).

Delete (15)



MODUL 10

SORT

Definisi Sort

Sort adalah proses pengurutan data yang sebelumnya disusun secara acak sehingga menjadi tersusun secara teratur menurut suatu aturan tertentu.

Pada umumnya terdapat 2 jenis pengurutan :

- ❖ Ascending (Naik)
- ❖ Descending (Turun)

Contoh :

Data Acak : 5 6 8 1 3 25 10

Terurut Ascending : 1 3 5 6 8 10 25

Terurut Descending : 25 10 8 6 5 3 1

Untuk melakukan proses pengurutan tersebut dapat digunakan berbagai macam cara / metoda. Beberapa metoda diantaranya :

a) Buble / Exchange Sort

b) Selection Sort

c) Insertion Sort

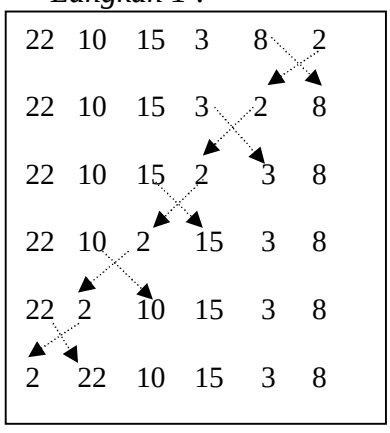
d) Quick Sort

Bubble / Exchange Sort

Memindahkan elemen yang sekanag dengan elemen yang berikutnya, jika elemen sekarang > elemen berikutnya, maka tukar

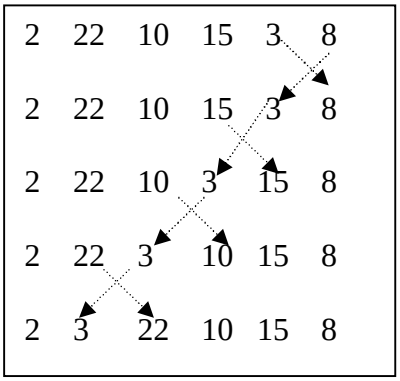
Proses :

Langkah 1 :



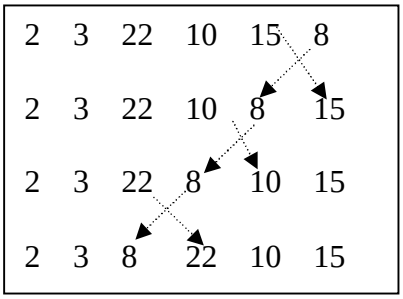
Pengecekan dapat dimulai dari data paling awal atau paling akhir. Pada contoh di samping ini pengecekan di mulai dari data yang paling akhir. Data paling akhir dibandingkan dengan data di depannya, jika ternyata lebih kecil maka tukar. Dan pengecekan yang sama dilakukan terhadap data yang selanjutnya sampai dengan data yang paling awal.

Langkah 2 :

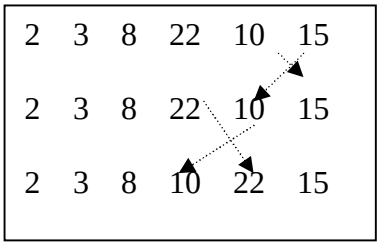


Kembalinya data paling akhir dibandingkan dengan data didepannya jika ternyata lebih kecil maka tukar, tetapi kali ini pengecekan tidak dilakukan sampai dengan data paling awal yaitu 2 karena data tersebut pasti merupakan data terkecil (didapatkan dari hasil pengurutan pada langkah 1).

Langkah 3 :

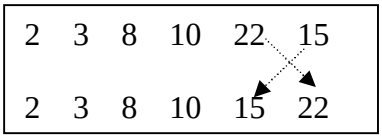


Langkah 4 :

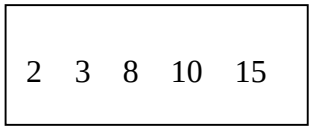


Proses pengecekan pada langkaj 3 dst. Sama dengan langkah sebelumnya.

Langkah 5 :



Terurut :



Proses di atas adalah pengurutan data dengan metoda bubble ascending.

Untuk yang descending adalah kebalikan dari proses diatas.

Berikut penggalan listing program Procedure TukarData dan Procedure Bubble Sort.

Procedure TukarData

```
Procedure TukarData(var a,b : word);
Var c : word;
    Begin
      c:=a;
      a:=b;
      b:=c;
    end;
```

Procedure Bubble Sort Ascending

```
Procedure Asc_Bubble(var data:array; jmldata:integer);
Var i,j : integer;
    Begin
     For i:= 2 to jmldata do
      For j:= jmldata downto I do
            If data[j] < data[j-1] then
              Tukardata (data[j], data[j-1]);
    end;
```

Untuk pengurutan secara descending anda hanya perlu menggantikan baris ke-6 dengan berikut ini :

```
If data[j] > data[j-1] then
```

Selection Sort

Membandingkan elemen yang sekarang dengan elemen yang berikutnya sampai dengan elemen yang terakhir. Jika ditemukan elemen lain yang lebih kecil dari elemen sekarang maka dicatat posisinya dan kemudian ditukar. Dan begitu seterusnya.

Proses :

Langkah 1:

```
i = 1   2   3   4   5   6
    22  10  15  3   8   2

pembanding      Posisi
22 > 10         2
10 < 15         2
10 > 3          4
3  < 8          4
3  > 2          6
Posisi data ke-1(22) = 6
Tukar data ke-1 dengan data ke-6

        2   10   15   3   8   22
```

Langkah 2 :

```
i = 1   2   3   4   5   6
    2   10  15  3   8   22

pembanding      Posisi
10 < 15         2
10 > 3          4
3  < 8          4
3  < 22         4

Posisi data ke-2(10) = 4
Tukar data ke-2 dengan data ke-4

        2   3   15   10   8   22
```

Langkah 3 :

```
i = 1   2   3   4   5   6
    2   3   15  10  8   22

pembanding      Posisi
15 > 10         4
10 < 8          4
8  > 22         5

Posisi data ke-3(15) = 5
Tukar data ke-2 dengan data ke-5

        2   3   8   10   15   22
```

Langkah 4 :

```
i = 1   2   3   4    5    6
    2   3   8   10   15   22
pembanding      Posisi
10 < 15         4
10 < 22         4

Posisi data ke-4 tetap
Pada posisinya = 4 (tidak berubah)

        2   3   8   10   15   22
```

Langkah 5 :

```
i = 1   2   3   4    5    6
    2   3   8   10   15   22

pembanding      Posisi
15 < 20         5
posisi data ke-5 tetap
pada posisinya = 5 (tidak berubah)
```

Terurut :

```
2   3   8   10   15   22
```

Proses pengurutan di atas adalah dengan metoda selection Ascending. Untuk descending hanyalah kebalikan dari proses di atas. Berikut penggalan listing program Procedure Selection Sort secara ascending

Procedure Selection Sort Ascending

```
Procedure Asc_Selection;
Var min, pos : byte;
 Begin
    For i:= 1 to max-1 do
 Begin
       Pos:=i;
            For j:= i+1 to max do
              If data[j] < data[pos] then pos:=j;
            If  i <> pos then tukardata(data[i],data[pos]);
         end;
end;
```

untuk pngurutan secara desending, anda hanya perlu mengganti baris ke-8 sbb :

if *data[pos] < data[j]* ***then*** *pos:=j;*

Insertion Sort

Pengurutan dilakukan dengan cara membandingkan data ke-I (dimana I dimulai dari data ke-2 sampai dengan data terakhir) dengan data berikutnya. Jika ditemukan data yang lebih kecil maka data tersebut disisipkan ke depan sesuai posisi yang seharusnya.

Proses :

Langkah 1:

```
i =  1   2   3   4   5   6
     22  10  15  3   8   2

temp      cek            geser
10        temp<22        data ke-1→ posisi 2

temp menempati posisi ke-1.

10   22   15   3   8   2
```

Langkah 2 :

```
i =  1   2   3   4   5   6
     10  22  15  3   8   2

temp      cek            geser
15        temp<22        data ke-2→ posisi 3
          temp>10

temp menempati posisi ke-2.

10   15   22   3   8   2
```

Langkah 3 :

```
i =  1   2   3   4   5   6
     22  10  15  3   8   2

temp      cek            geser
3         temp < 22      data ke-3→ posisi 4
          temp < 15      data ke-2→ posisi 3
          temp < 10      data ke-1→posisi 2

temp menempati posisi ke-1.

3   10   15   22   8   2
```

Langkah 4:

```
i =  1   2   3   4   5   6
     22  10  15  3   8   2

temp      cek            geser
 8        temp < 22      data ke-4→posisi 5
          temp < 15      data ke-3→posisi 4
          temp < 10      data ke-2→posisi 3
          temp > 3

temp menempati posisi ke-2.

3   8   10   15   22   2
```

Langkah 5 :

```
i =  1   2   3   4   5   6
     22  10  15  3   8   2

temp      cek            geser
 8        temp < 22      data ke-5→posisi 6
          temp < 15      data ke-4→posisi 5
          temp < 10      data ke-3→posisi 4
          temp > 8       data ke-2→posisi 3
          temp > 3       data ke-1→posisi 2

temp menempati posisi ke-1.
```

Langkah 6 :

```
2   3   8   10   15   22
```

Procedure Insertion Sort Ascending

```
Procedure Asc_Insert;
Var i , j , temp : byte;
 Begin
  For i := 2 to max do
    Begin
      Temp :=data[i];
       j := i-1;
       while (data[j] > temp) and (j>0) do
          begin
               data[j+1] := data[j];
               dec(j);
          end;
       data[j+1]:=temp;
    end;
 end;
```

Untuk pengurutan secara descending anda tinggal mengganti baris ke 8 dengan baris berikut ini :

While**(data[j]<temp)and(j>0)**do

QUICK SORT

Membandingkan suatu elemen (disebut pivot) dengan elemen yang lain dan menyusunnya sedemikian rupa sehingga elemen- elemen lain yang lebih kecil daripada pivot tersebut terletak di sebelah kirinya dan elemen-elemen lain yang lebih besar daripada pivot tersebut terletak di sebelah kanannya. Sehingga dengan demikian telah terbntuk dua sublist, yang terletak di sebelah kiri dan kanan dari pivot. Lalu pada sublist kiri dan sublist kanan kita anggap sebuah list baru dan kita kerjakan proses yang sama seperti sebelumnya. Demikian seterusnya sampai tidak terdapat sublist lagi. Sehingga didalamnya telah terjadi proses Rekursif.

Proses :

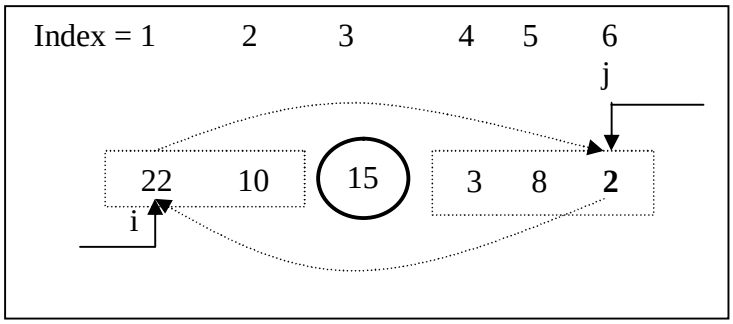
Bilangan yang di dalam kurung merupakan pivot

Persegi panjang yang digambarkan dengan garis terputus-putus menunjukkan sublist.

i bergerak dari sudut kiri ke kanan sampai mendapatkan nilai yang >= pivot.

j bergerak dari sudut kanan ke kiri sampai menemukan nilai yang < pivot.

Langkah 1 :



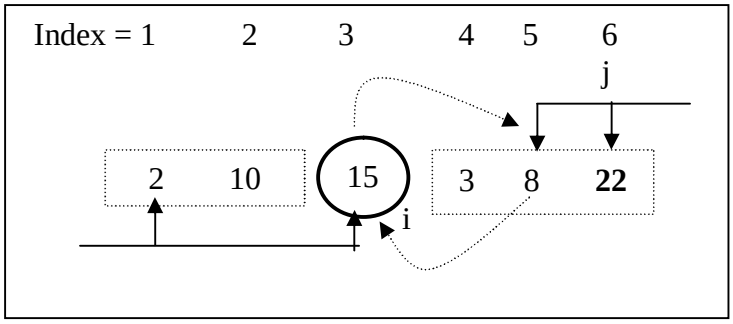
i berhenti pada index ke-1 karena langsung mendapatkan nilai yang > dari pivot (15).

j Berhenti pada index ke-6 karena juga langsung mendapatkan nilai yang < dari pivot.

Karena i < j maka data yang ditunjuk olh I ditukar dengan data yang ditunjuk oleh j sehingga menjadi :

2 10 15 3 8 22

Langkah 2 :



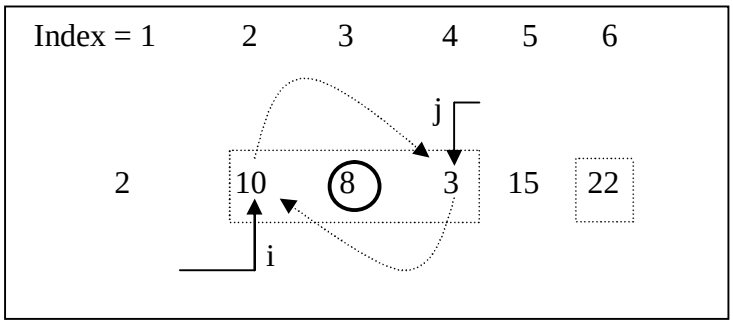
i berhenti pada index ke-3 (pivot) karena tidak menemukan bilangan yang > dari pivot.

j berhenti pada index k-5 menunjuk pada nilai yang < dari pivot.

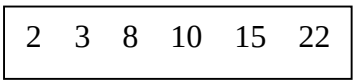
Karena i < j maka data yang ditunjuk oleh i (pivot) ditukar dengan data yang ditunjuk oleh j sehingga menjadi :

2 10 8 3 15 22

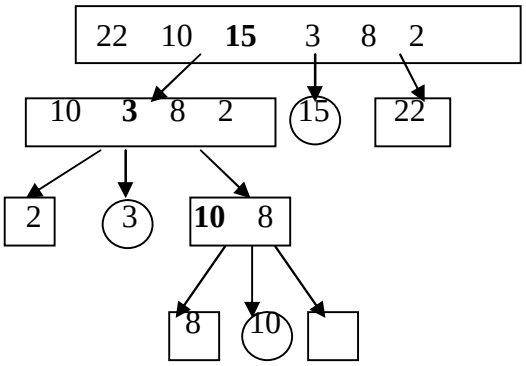
Langkah 3 :



Proses yang sama seperti sebelumnya dilakukan terhadap 2 buah sublist yang baru (ditandai dengan persegi panjang dengan garis terputus-putus).



Atau dapat juga digambarkan dalam bentuk tree seperti di bawah ini dengan pivot yang ditandai dengan huruf tebal. Kemudian setelah terurut dibaca inorder.



Procedure Quisort dengan nilai paling kiri sebagai pembanding (pivot).

```
Procedure Asc_Quick(L,R : Integer);
Var i, j:integer;
 Begin
   If L<R then
     Begin
    i := L;  j := R+1;
     repeat
       repeat inc(i) until data[i] >= data[1];
```

```
        repeat dec(j) until data[j] <= data[1];
            if  i <  j then tukardata (data[i], data[j]);
       until  i > j;
     tukardata (data[1], data[j]);
     Asc_Quick(L,j-1);
     Asc_Quick(j+1,R);
      End;
End;
```

Untuk pengurutan secara descending anda tinggal mengganti tanda aritmatik pada baris k 8 dan 9 sehingga menjadi seperti baris berikut :

```
        repeat inc(i) until data[i] >= data[l];
        repeat dec(j) until data[j] <= data[l];
```

Procedure Quick Sort dengan nilai tengah sebagai pembanding (pivot).

```
 Procedure Asc_Quick(L,R : Integer);
Var
  mid, i, j : integer;
  begin
    i:= L;     j:=R     mid := data[(L+R) div 2];
    repeat
      while data[i] < mid do inc(i);
      while data[j] > mid do dec(j);
      if  i < j then
         begin
              change(data[i],data[j]);
              inc(i); dec(j);
         end;
     until  i > j;
     if L < j then Asc_Quick(L , j);
     if  i > R then Asc_Quick(i , R);
end;
```

Untuk pengurutan secara descending, anda hanya perlu mengganti baris ke-6 & 7 sbb :

while *data[j] < mid* ***do*** *inc(j);*

while *data[k] > mid* ***do*** *dec(k);*

Latihan Soal beserta jawaban (Listing program) dan penjelasan.

Anda diminta membuat sbuah program sorting dengan metode bubl sort. Mintalah user untuk memasukkan 10 angka. Lalu tampilkan angka-angka trsebut setelah disort baik secara ascending maupun descendeing

Layar 1 :

```
Masukkan 10 data
= = = = = = = = = =
Data ke-1 = 5          Data ke-6 = 45
Data ke-2 = 2          Data ke-7 = 8
Data ke-3 = 67         Data ke-8 = 23
Data ke-4 = 43         Data ke-9 = 39
Data ke-5 = 90         Data ke-10 = 7
```

{ket : tampilan ketika mengiput 10 angka}

Layar 2 :

```
5  2  67  43  90  45  8  23  39  7
Data yang telah diurutkan :
* * * * * * * * * * * * * *
Ascending  : 2   5   7   8   23  39  43  45  67  90
Descending : 90  67  45  43  39  23  8   7   5   2
```

{ket : tampilan setelah dilakukan bubble sort}

jawaban :

```
uses crt;
const        max = 10;
Type         arr = array[1..max] of byte;
Var  i       : byte;
       Data : arr;

Procedure Input;
 begin
     Clrscr;
     Writeln (‘Masukkan 10 data’);
     Writeln (‘= = = = = = = = = =’);
     For i := 1 to max do    {input 10 data}
          begin
        write(‘Data ke-‘, i ,’=’); readln(data[i]);
     end;
     Clrscr;
     For i := 1 to max do
               Write(data[i],’ ‘);
     Writeln;
     Writeln (‘ * * * * * * * * * * * * * * *);
     Writeln (‘Data yang telah diurutkan :’);
 end;

Procedure Change (var a,b :byte); {procedure untuk menukar data}
Var c : byte;
 Begin
     c := a;  a := b;  b := c;
 end;

Procedure Asc_Buble; {pengurutan secara ascending}
Var p,q  : byte;
```

```
    flaq : boolean;
  begin
    flaq:=false;
    p:=2;
    while (p<max) and (not flaq) do
      begin
        flaq:=true;
    for  q := max downto p do
    if data[q] < data[q-1] then
      begin
            change (data[q], data[q-1]);
            flaq:=false;
              end;
           inc(i);
      end;
    write('Ascending :');
  end;

Procedure Desc_Buble; {pengurutan secara descending}
Var p, q  : byte;
    Flaq : boolean;
  Begin
       flaq:=false;
       p:=2;
       while (p<max) and (not flaq) do
            begin
         flaq:=true;
    for  q := max downto p do
    if data[q] < data[q-1] then
      begin
            change (data[q], data[q-1]);
            flaq:=false;
```

```
                end;
            inc(i);
        end;
    write('Descending :');
  end;

Procedure Output;
Begin
    For i := 1 to max do
            Write(data[i],'');
    Writeln;
end;
Begin {program utama}
    Input;
    Asc_buble; output;
    Desc_buble; output;
    Readkey;
end.
```