Algoritma dan Struktur Data

Algoritma Pemrograman

Bekti Wulandari, M.Pd
Kelas B TE
2014

Program

- Program: sederetan perintah-perintah yang harus dikerjakan oleh komputer untuk menyelesaikan masalah.
- 3 level bahasa pemrograman:
  1. Bahasa tingkat rendah
  2. Bahasa tingkat menengah
  3. Bahasa tingkat tinggi

Bahasa Tingkat Rendah

- Bahasa mesin
- Berisi: kode-kode mesin yg hanya dapat diinterpretasikan langsung oleh mesin komputer.
- Berupa kode numerik 0 dan 1
- Microcode: sekumpulan instruksi dalam bahasa mesin
- (+) : Eksekusi cepat
- (-) : Sulit dipelajari manusia

Bahasa Tingkat Menengah

- Bahasa Assembly
- Bahasa simbol dari bahasa mesin
- Contoh: ADD, SUB, dll
- Macro instruksi: sekumpulan kode dalam bahasa assembly
- (+) : Eksekusi cepat, masih dapat dipelajari daripada bahasa mesin, file kecil
- (-) : Tetap sulit dipelajari, program sangat panjang

Bahasa Tingkat Tinggi

- The 3rd Generation Programming Language
- Lebih dekat dengan bahasa manusia
- Memberi banyak fasilitas kemudahan dalam pembuatan program, mis.: variabel, tipe data, konstanta, struktur kontrol, loop, fungsi, prosedur, dll
- Contoh: Pascal, Basic, C++, Java
- (+) : Mudah dipelajari, mendekati permasalahan yang akan dipecahkan, kode program pendek
  (-) : Eksekusi lambat, tidak dapat dilakukan langsung oleh komputer (membutuhkan translator)

Translator

- Translator: penerjemah dari bahasa tingkat tinggi ke bahasa tingkat rendah.
- Assembler merupakan penerjemah bahasa Assembly ke bahasa mesin.

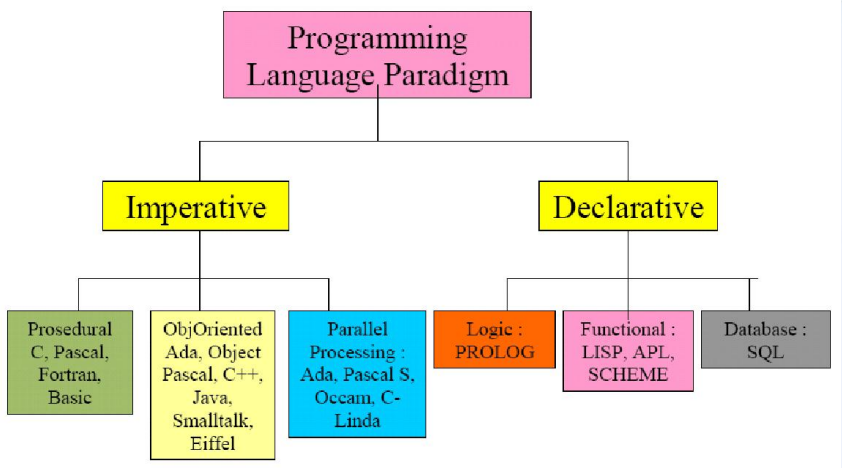
Interprenter

- Perintah diterjemahkan baris demi baris jadi program tidak dianalisis seluruhnya dulu tapi bersamaan dengan jalannya program.
- Interpreter adalah suatu program komputer yang mampu menerjemahkan program dari bahasa tingkat tinggi yang dimengerti oleh manusia dan langsung menjalankan program tersebut.
- (+) : mudah bagi user, debugging cepat
- (-) : eksekusi program lambat,
  tidak langsung menjadi program executable
- Contoh: Basic, List

Compiler

- Seluruh program diterjemahkan.
- Semua perintah (pascal, C++) dirubah dalam bentuk exe atau bahasa asembly.

Paradigma Bahasa Pemrograman



Algoritma

- Algoritma: sederetan langkah-langkah logis yang disusun secara sistematis untuk memecahkan suatu masalah.
- Disebut Logis karena setiap langkah bisa diketahui dengan pasti.
- Algoritma lebih merupakan alur pemikiran untuk menyelesaikan suatu pekerjaan atau suatu masalah.

Syarat Algoritma

- Algoritma harus tidak ambigu
- Algoritma harus tepat
- Algoritma harus pasti
- Algoritma harus berhenti setelah mengerjakan sejumlah langkah terbatas.
- Algoritma memiliki nol atau lebih masukkan,
- Algoritma memiliki satu atau lebih keluaran.
- Algoritma harus efektif

Belajar Memprogram dan Belajar Bahasa Pemrograman

- Belajar memprogram adalah belajar tentang metodologi pemecahan masalah, kemudian menuangkannya dalam suatu notasi tertentu yang mudah dibaca dan dipahami.
- Belajar bahasa pemrograman adalah belajar memakai suatu bahasa, aturan tata bahasanya, instruksi-instruksinya, tata cara pengoperasian compiler-nya untuk membuat program yang ditulis dalam bahasa itu saja.

Notasi Algoritma

- Penulisan algoritma tidak tergantung dari spesifikasi bahasa pemrograman dan komputer yang mengeksekusinya.
- Notasi algoritma bukan notasi bahasa pemrograman tetapi dapat diterjemahkan ke dalam berbagai bahasa pemrograman.

Notasi Algoritma

1. Uraian kalimat deskriptif (narasi)

Contoh:

Algoritma Kelulusan_mhs

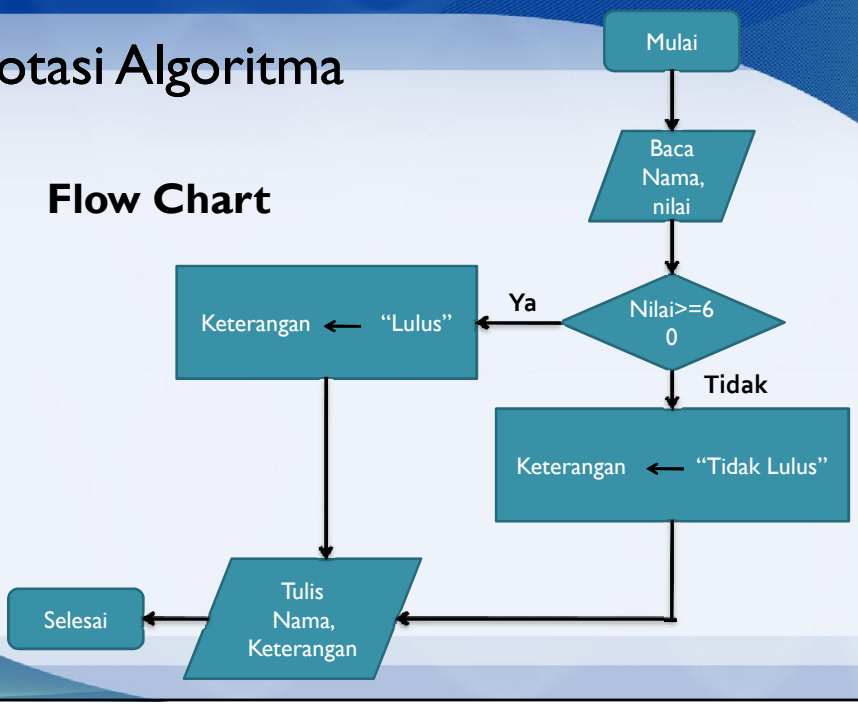
Diberikan nama dan nilai mahasiswa, jika nilai tersebut lebih besar atau sama dengan 60 maka mahasiswa tersebut dinyatakan lulus jika nilai lebih kecil dari 60 maka dinyatakan tidak lulus.

DESKRIPSI :

1. baca nama dan nilai mahasiswa.
2. jika nilai >= 60 maka
3. keterangan ← lulus
4. tetapi jika
5. keterangan ← tidak lulus.
6. tulis nama dan keterangan

Notasi Algoritma

2. **Flow Chart**



Notasi Algoritma

3. **Pseudo Code**
   Ada 3 bagian: Judul, Deklarasi, Deskripsi.

```
Algoritma kelulusan
Deklarasi
    nama, keterangan : string
    nilai : integer
Deskripsi
    read (nama, nilai)
    if nilai >= 60 then
            keterangan ← 'lulus'
    else
            keterangan ← 'tidak lulus'
    write(nama, keterangan)
```

Aturan Pseudo Code

- **Judul algoritma**
  Bagian yang terdiri atas nama algoritma dan penjelasan (spesifikasi) tentang algoritma tersebut. Nama sebaiknya singkat dan menggambarkan apa yang dilakukan oleh algoritma tersebut.
- **Deklarasi**
  Bagian untuk mendefinisikan atau mendeklarasikan semua apa yang digunakan atau dibutuhkan dalam pemrograman.
- **Deskripsi**
  Bagian ini berisi uraian langkah-langkah penyelesaian masalah.

Operator Aritmetik

- /, *, div, mod → level tinggi
- +, - → level rendah
- Mod dan div hanya untuk bilangan bulat!

- Contoh :
- 5 – 2 + 1 = ?
- 5 – 2 * 3 = ?
- (6 + 3 * 2) / 6 – 2 * 3 = ?

Tipe Data

- Bilangan bulat (Shortint , Integer, Longint, Byte, Word)
- Boolean (Boolean, ByteBool ,WordBool, LongBool)
- Bilangan real (Real, Single, Double, Extended, Comp)
- Karakter
- String

Latihan

Buatlah notasi algoritma untuk :

a. Menghitung luas dan keliling lingkaran dengan memasukkan nilai jari-jari
b. Mengidentifikasi suatu bilangan apakah bilangan tersebut ganjil atau genap

Pertemuan 2

Percabangan Sederhana

MK. Algoritma dan Struktur Data

Bekti Wulandari, M.Pd.
TE KELAS B
2014

Definisi Percabangan

- Percabangan adalah suatu suatu perintah (pernyataan) yang memungkinkan suatu perintah (pernyataan) dieksekusi jika suatu kondisi terpenuhi atau tidak terpenuhi.
- Jika suatu kondisi terpenuhi, maka perintah akan dilaksanakan.
- Jika kondisi tidak terpenuhi, maka perintah yang lainnya yang dilaksanakan.

Definisi Percabangan
(lanjutan)

- Percabangan (brancing) di dalam pemrograman digunakan oleh komputer untuk menentukan langkah kerja instruksi.
- Percabangan menggunakan operator kondisional yang akan menghasilkan nilai boolean (benar/true atau salah/false).
- Jika nilai yang dihasilkan benar, maka perintah (instruksi) akan dilaksanakan, sedangkan jika salah, maka instruksi tidak akan dilaksanakan atau melaksanakan instruksi lainnya.

Macam Percabangan

1. Satu Kasus

- <u>if</u> kondisi <u>then</u> aksi1
- Notasi algoritma :

```
if kondisi-terpenuhi (true) then
      laksanakan_aksi
endif
```

- kondisi berupa ekspresi yang menghasilkan true /false
- aksi berupa instruksi yang akan dilaksanakan jika kondisi yang dipasangkan dengan aksi yang bersangkutan bernilai benar. Bila kondisi bernilai salah, tidak ada pernyataan apapun yang dikerjakan

- Contoh

  Mencetak pesan “bilangan genap” jika bilangan tersebut genap.

Macam Percabangan

(lanjutan 1)

2. Dua Kasus

- if kondisi then aksi1 else aksi2
- Notasi Algoritma

  if kondisi-terpenuhi (true) then

  laksanakan_aksi

  else kondisi_tidak_terpenuhi (false)

  endif
- Digunakan untuk menguji sebuah kondisi dimana jika kondisi terpenuhi maka perintah yang telah ditentukan akan dijalankan, tetapi jika kondisi tidak terpenuhi maka perintah yang lain yang akan dijalankan.

- Contoh

  Mencetak pesan “bilangan genap” jika bilangan tersebut bilangan genap, atau “bilangan ganjil” jika bilangan tersebut ganjil.

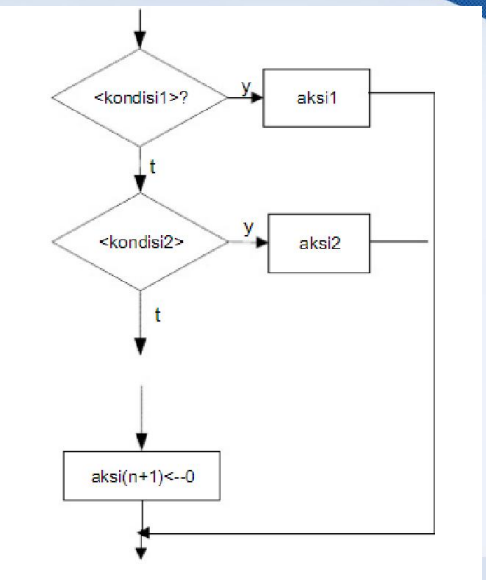
Macam Percabangan

(lanjutan 1)

3. Tiga Kasus atau Lebih

- if kondisi1 then
  aksi1
  else
  if kondisi2 then
  aksi2
  else aksi3
  endif
  endif
- Hampir sama dengan bentuk percabangan kedua tetapi kondisi yang diuji lebih dari satu.

- Flowchart



- Contoh
  Membaca temperatur air (T) pada tekanan normal (dalam satuan derajat celsius). Lalu menentukan apakah wujud air tersebut dalam keadaan padat (T≤ 0), cair (0<T<100), gas (T≥100)

```
Algoritma :
Program......
Deklarasi
    T :......
Algoritma
    Read (...)
    if T≤ 0 then
    Write ('.......')
        else if (......) and (.....) then
            Write ('.......')
             if T≥100 then
               write('.......')
            end if
       end if
    endif
```

Soal

1. Karyawan honorer di PT ABC digaji berdasarkan jumlah jam kerja selama satu minggu. Upah per jam Rp 2000,-. Bila jumlah jam kerja lebih besar dari 48 jam, maka sisanya dianggap sebagai jam lembur. Upah lembur Rp 3000,-/jam. Tulislah algoritma yang membaca nama pegawai jumlah jam kerja seorang karyawan selama satu minggu, lalu menentukan upah mingguannya.
2. Menampilkan bilangan terbesar dari tiga bilangan yang dimasukkan!
3. Buatlah algoritma untuk menentukan apakah bilangan bulat yang dimasukkan tersebut bilangan positif, bilangan negatif, atau bilangan nol!

Soal

(lanjutan)

4. Buatlah algoritma yang membaca sebuah titik P(x,y) di bidang kartesian, lalu menentukan di kuadran mana letak titik tersebut!
5. Mengurutkan tiga bilangan yang dimasukkan dari kecil ke besar.
6. Mengurutkan tiga bilangan yang dimasukkan dari kecil ke besar dimana bilangan yang dimasukkan tidak boleh ada yang sama.

Struktur CASE

- Digunakan untuk memilih jika terdapat lebih dari dua kondisi
- Case ekspresi of

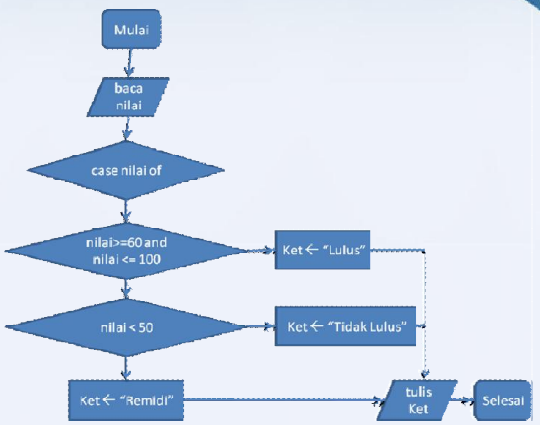
  nilai1: aksi1
  nilai2: aksi2
  nilai3: aksi3
  .....
  nilaiN:aksiN
  otherwise:aksiX

  endcase

Contoh 1

Diberikan nama dan nilai mahasiswa, jika nilai tersebut lebih besar atau sama dengan 60 maka mahasiswa tersebut dinyatakan lulus jika nilai lebih kecil dari 50 maka dinyatakan tidak lulus.

Bila nilainya 50 sampai dengan 59, maka harus mengikuti remidi.



Bagaimana Pseudo Code ?

- Pseudo code :

```
algoritma kelulusan
deklarasi
        nilai : integer
        ket : string
deklarasi
        read (nilai)
        Case nilai of
        60 ..100: ('lulus')
        50 .. 59 : ('remidi')
        0 .. 49 : ('tidak lulus')
        endcase
        Write (keterangan)
```

Contoh 2:

- Buatlah algoritma dan program yang membaca angka bulan dan tahun, lalu menuliskan jumlah hari dalam bulan tersebut. Misalnya jika dibaca bulan 8 (agustus), maka jumlah harinya adalah 31.

Bulan	Jumlah hari
1, 3, 5, 7, 8, 10, 12	31
4, 6, 9, 11	30
2	29 (jika tahun kabisat), 28 (jika bukan kabisat)

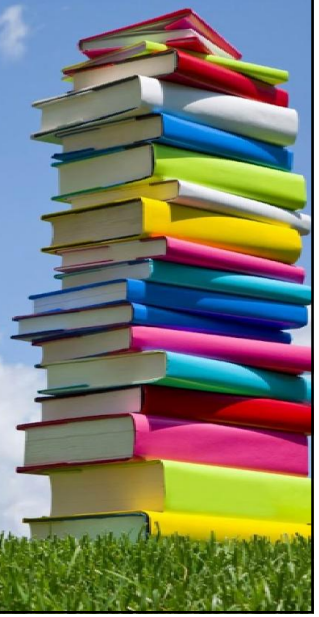
```
Algoritma JUMLAH_HARI
{ menentukan jumlah hari dalam satu bulan }

DEKLARASI
        AngkaBulan      : integer               { 1 . . 12 }
        Tahun           : integer               { > 0 }
        JumlahHari      : integer

DESKRIPSI
        read (AngkaBulan,Tahun)
        case (AngkaBulan) of
           AngkaBulan= [1, 3, 5, 7, 8, 10, 12 ] : JumlahHari←31
           AngkaBulan= [ 4, 6, 9, 11 ]          : JumlahHari←31
           AngkaBulan= 2 : case Tahun
                           Tahun mod 4 = 0 : JumlahHari←29
                           Tahun mod 4 ≠ 0 : JumlahHari←28
                           endcase
        endcase

        write(JumlahHari)
```

Pertemuan 3

Perulangan

MK. Algoritma dan Struktur Data

Bekti Wulandari, M.Pd.
TE KELAS B
2014



pendahuluan

- Digunakan untuk menjalankan satu atau beberapa pernyataan sebanyak beberapa kali.
- Terdiri dari dua bagian :

  1. kondisi pengulangan

     ekspresi boolean yang harus dipenuhi untuk melaksanakan pengulangan

  2. badan pengulangan

     aksi/pernyataan yang harus diulang selama kondisi yang ditentukan untuk pengulangan masih dipenuhi.

1. inisialisasi, yaitu aksi yang dilakukan sebelum pengulangan dilakukan pertama kali
2. terminasi, yaitu aksi yang dilakukan seteleh pengulangan selesai dilaksanakan

Struktur pengulangan

<inisialisasi>

Awal pengulangan

Badan pengulangan

Akhir pengulangan

<terminasi>

Struktur kontrol pengulangan

1 • Pernyataan FOR

2 • Pernyataan WHILE

3 • Pernyataan REPEAT

Pernyataan FOR

- Digunakan untuk menghasilkan pengulangan sejumlah (n) kali yang dispesifikasikan.
- Jumlah pengulangan diketahui (dapat ditentukan) sebelum eksekusi.
- Variabel pencacah
  - Nilainya selalu bertambah setiap kali perulangan dilakukan.
  - Jika nilainya sudah mencapai jumlah yang dispesifikasikan, maka proses perulangan akan berhenti
- Bentuk umum for :
  - Menaik (*ascending*)
  - Menurun (*descending*)

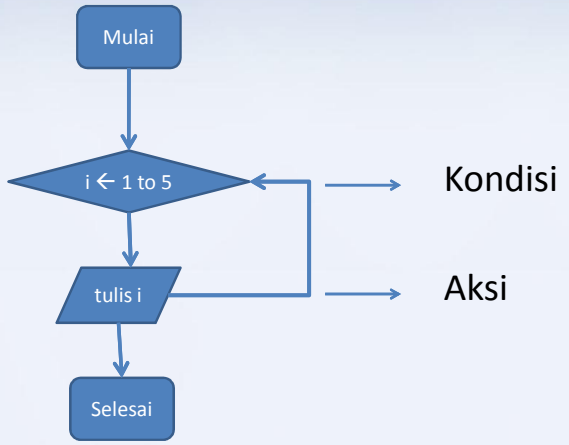
For to do

- Notasi algoritmatik

```
FOR (nama_pencacah ← nilai awal) TO (nilai_akhir) DO
        (aksi/pernyataan)
End For
```

- Ket :

1. Pencacah haruslah dari tipe data integer atau karakter, diinisialisasi dengan nilai_awal, nilai pencacah secara otomatis bertambah satu setiap kali badan pengulangan dimasuki
2. Aksi dapat berupa satu/lebih instruksi yang diulang
3. Nilai_awal harus lebih kecil dari nilai_akhir
4. Jumlah pengulangan yang terjadi = nilai_akhir – nilai_awal + 1



For downto do

- Notasi algoritma

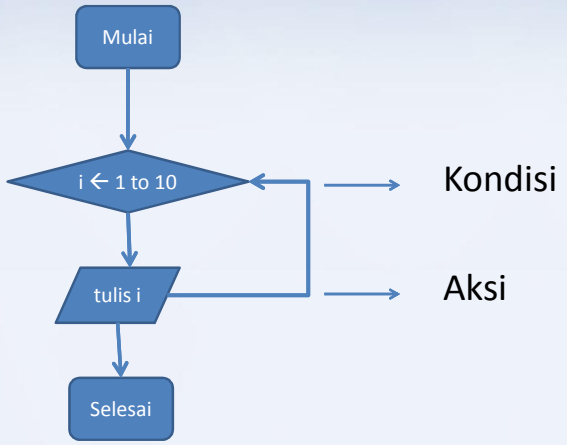
```
FOR (nama_pencacah ← nilai_akhir) DOWNTO (nilai_awal) DO
        (aksi/pernyataan)
End For
```

Contoh

1. Misalkan kita ingin mencetak angka 1 sampai 10, maka algoritmanya :

```
Program Tulis_Nomor
Deklarasi :
    i : integer
Algoritma
    For i ← 1 to 10 do      {kondisi}
      Write (i)             {aksi}
EndFor
```

2. Jika mencetak angka 1 sampai N bagaimana algoritmanya? Apa yang terjadi jika N = 0? N=-1? N = 1?



Pernyataan WHILE

- Perulangan ini dipilih jika kita tidak tahu berapa kali suatu pernyataan akan diulang-ulang.
- Banyak perulangan dilakukan melalui pemeriksaan suatu kondisi tertentu.

  Dengan demikian pemeriksaan kondisi terlebih dahulu dilakukan sebelum perulangan dijalankan.

while do

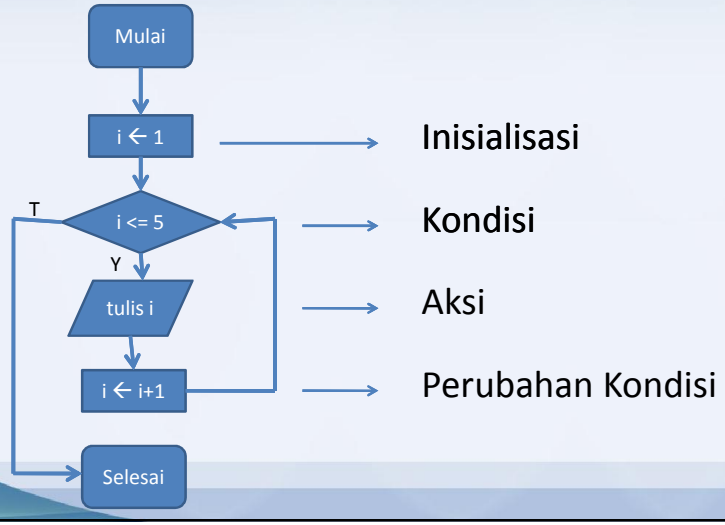
- Syntaks

```
While kondisi_pengulangan do
        Aksi
EndWhile
```

Keterangan :

Aksi akan dilakukan selama kondisi bernilai *true.*
Jika kondisi bernilai *false*, badan pengulangan tidak akan dilaksanakan, yang berarti perulangan selesai.
Yang harus diperhatikan → pengulangan harus berhenti.
Supaya kondisi bernilai *false*,
Di dalam badan pengulangan harus ada instruksi yang mengubah nilai peubah kondisi.

while do (lanjutan)



Inisialisasi

Kondisi

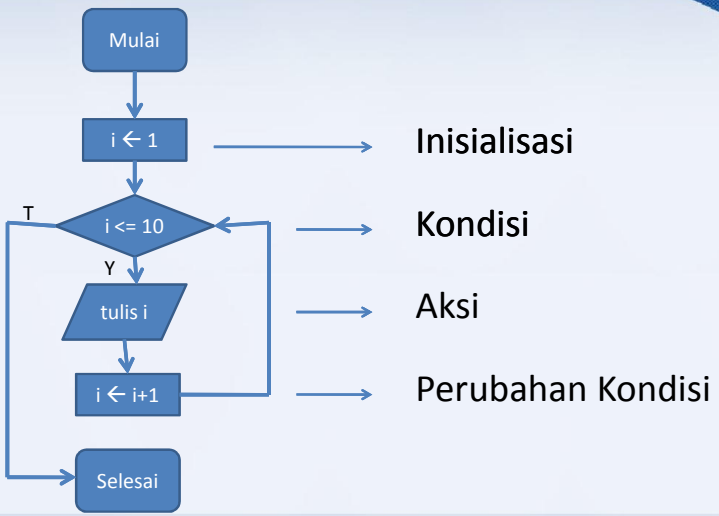
Aksi

Perubahan Kondisi

Contoh

1. Mencetak angka 1 sampai dengan 10

```
Program Tulis_Angka
Deklarasi :
    i : integer
Algoritma
   i ← 1          {inisialisasi}
   while i ≤ 10 do      {kondisi}
    Write (i)                {aksi}
  i ← 1 + 1            {perubahan kondisi}
Endwhile
{i > 10}
```

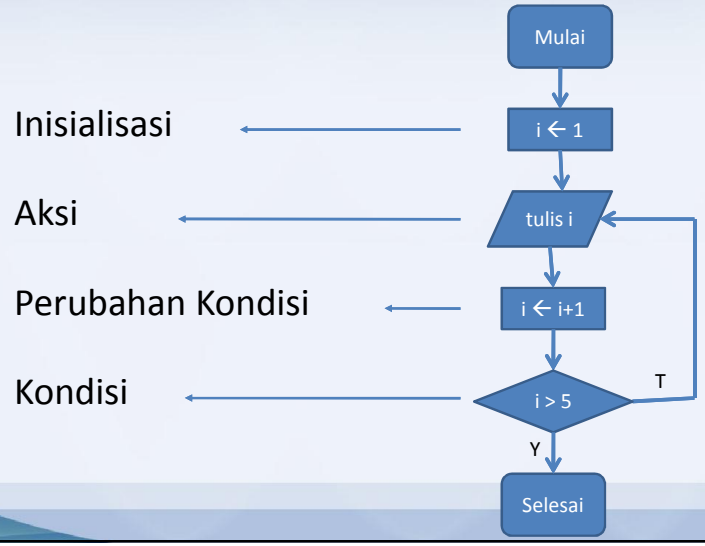


Pernyataan repeat

- Bentuk perulangan ini akan melakukan aksi terlebih dahulu (minimal dilakukan satu kali), kemudian baru melakukan pemeriksaan terhadap kondisi.
- Jika kondisi benar maka perulangan masih akan tetap dilakukan.
- Perulangan akan dilakukan sampai kondisi false.
- Tidak mengetahui berapa kali pengulangan akan dikerjakan.
- Di dalam pernyataan, harus ada instruksi yang mengubah nilai kondisi agar pengulangan berhenti.

```
repeat
        aksi
until kondisi
```

repeat until



Contoh

1. Mencetak angka 1 sampai dengan 10

```
Deklarasi :
        i : integer
Algoritma
  i ← 1        {inisialisasi}
  repeat
    Write (i)        {aksi}
    i ← 1 + 1        {perubahan kondisi}
    until i > 10     {kondisi}
```

- FOR digunakan untuk proses pengulangan yang jumlah pengulangannya dapat diketahui
- WHILE fungsinya sama seperti FOR, tetapi WHILE juga dapat digunakan untuk proses yang jumlah pengulangannya tidak dapat ditentukan sebelum eksekusi
- REPEAT fungsinya sama seperti WHILE, dapat menggunakan WHILE ataupun REPEAT untuk masalah-masalah tertentu. Tetapi pada beberapa masalah, pemilihan WHILE atau REPEAT tergantung kepada persoalaanya.

Perbedaan penggunaan REPEAT atau WHILE

- Pada REPEAT, kondisi pengulangan diperiksa pada akhir pengulangan. Jadi instruksi dalam badan pengulangan dilaksanakan dulu, baru pemeriksaan kondisi dilakukan. Konsekuensinya badan pengulangan dilakukan paling sedikit satu kali. → terlebih dahulu memanipulasi objek, baru memeriksa kondisi objek tersebut.
- Pada WHILE, kondisi pengulangan diperiksa di awal pengulangan. Jadi instruksi dalam badan pengulangan hanya dapat dilaksanakan jika pemeriksaan menghasilkan nilai true. (badan pengulangan tidak akan dilaksanakan jika kondisi pengulangan pertama kali bernilai false) → mengharuskan terlebih dahulu pemeriksaan kondisi objek sebelum objek tersebut dimanipulasi

Diskusikan

1. Buat algoritma dan flowchart untuk melakukan penjumlahan deret 1+2+3+4+... N
2. 1 2 3 4 5

   6 7 8 9 10

   11 12 13 14 15

   16 17 18 19 20

   Dengan satu loop

3. Menghitung bilangan faktorial.
5. Menghitung perpangkatan, a ^ n dimana a dan n adalah bilangan bulat.

Nested Looping

- Perulangan yang terdiri dari lebih dari satu perulangan.
- Disebut perulangan bersarang (Nested Looping).
- Nested Loop pada For ..... to ..... Do

  Syntax For....to.....do

  Instruksi

  For....to....do

  Instruksi...

  end

  end

  Proses perulangan yang akan diselesaikan terlebih dahulu adalah perulangan yang terletak pada bagian dalam

- Nesterd Loop Pada While .....Do

  Pada prinsip kerjanya nested loop while ...do sama dengan for..to..do dimana proses perulangan yang akan diselesaikan terlebih dahulu adalah perulangan yang terletak bagian dalam

  Syntax :

  While.....do

  Instruksi

  While .....do

  Instruksi

  End

  End

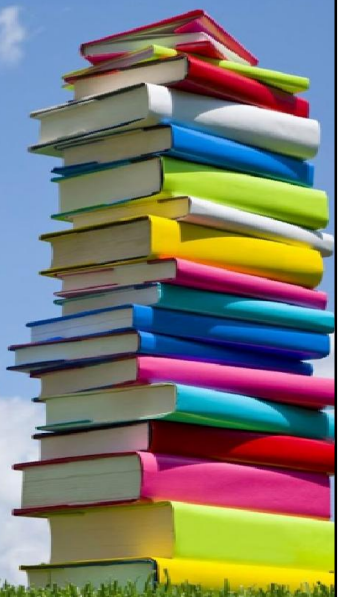
  Proses nested repeat until hampir sama dengan proses yang ada pada nested for dan nested while. Tetapi disini masing-masing perulangan pada repeat ...until satu kali proses pasti akan dilakukan sesuai dengan keterangan yang ada pada perulangan dengan repeat until

Pertemuan 4

ARRAY

MK. Algoritma dan Struktur Data

Bekti Wulandari, M.Pd.
TE KELAS B
2014



Pengertian

- Struktur data statis yang menyimpan sekumpulan elemen yang bertipe sama, dimana setiap elemen memiliki nilai indeks.
- Salah satu tipe data terstruktur, dibutuhkan untuk menyimpan serangkaian elemen bertipe sama.
- Struktur yang dapat diakses secara acak karena semua elemen array dapat diacu secara acak dengan urutan tertentu, yaitu mengetahui nomor urutnya yang disebut indeks.

Ilustasi

- Sebuah asrama mahasiswa, yang setiap kamar mempunyai nomor urut dan dihuni seorang mahasiswa.
- Seorang mahasiswa dapat dibedakan dengan nomor urut kamarnya.
- Asrama mahasiswa seperti array, elemen-elemen asrama bertipe sama, yaitu mahasiswa
- Nomor urut kamar adalah indeksnya.
- Untuk menampilkan nilai array tinggal menyebutkan indeks-nya. Misalkan untuk menampilkan nilai variabel x yang ke 5 dituliskan dengan x(5).

Pendeklarasian Array

Untuk dapat membuat variabel array maka terlebih dahulu harus didefinisikan nama variabel array dan berapa jumlah maksimalnya

Type namaarray = **array** [I] **of** tipe-basis;

I adalah jangkauan array, mulai dari indeks terkecil ke terbesar. Bertipe tertentu disebut tipe indeks (index type) -> integer dan char.

a. Sebagai variabel

Type arrhuruf : array [1..20] of char;

Type arrangka : array [1..100] of integer;

b. Sebagai tipe baru atau tipe bentukan (record)

```
type Mhs = record
        <NPM: integer,
        Nama : String [20],
        Alamat : String [30]>
type DataMhs = array [1..20] of Mhs
A:DataMhs
```

dalam Pascal :

```
type Mhs = record
        NPM: integer;
        Nama : String [20];
        Alamat : String [30];
end;
type DataMhs = array [1..20] of Mhs
A:DataMhs;
```

c. Mendefinisikan ukuran maksimum elemen array sebagai konstanta

```
Const Nmaks = 20
Type Huruf = array [1..Nmaks] of char
A:Huruf
```

Contoh penggunaan array adalah pada penyimpanan nilai seorang mahasiswa selama 10 kali mengikuti tes. Ilustrasinya sebagai berikut :

A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]	A[10]
86	90	97	100	98	79	76	55	90	100

- Nama variable array di atas adalah A , memiliki 10 elemen. Nilai 1,2,3 ... dst adalah nilai indeks untuk menunjuk elemen tertentu. Range yang digunakan pada array berdimensi satu di atas adalah 1 sampai dengan 10.
- Nilai mahasiswa A pada tes yang ketiga ditunjuk oleh variable A pada indeks ketiga ::: A[3] = 97.

Statement Option Base

- Dalam pemakaian sebuah array kita akan memakai sistem range.
- Contoh pada array nilai di atas (array A) mengindikasikan sebuah array dengan range 1 sampai 10.
- 1 merupakan range nilai awal sedangkan 10 merupakan range nilai akhir.
- Nilai range awal sebuah array, dapat dimulai dengan angka 0 (nol) atau 1 (seperti contoh di atas).

Pendahuluan

- Sub rutin adalah suatu bagian dalam program yang dapat melakukan tugas tertentu. Jadi sub rutin merupakan "program kecil" yang menjadi bagian dari suatu program yang besar.

1. prosedur
2. fungsi

   Perbedaan antara keduanya adalah setelah dipanggil prosedur tidak mengembalikan suatu nilai sedangkan fungsi selalu mengembalikan suatu nilai.

Procedure

PROSEDUR

Prosedur adalah suatu program yang terpisah dalam blok sendiri yang berfungsi sebagai subprogram (program bagian). Prosedur diawali dengan kata cadangan PROCEDURE di dalam bagian deklarasi prosedur. Prosedur dipanggil dan digunakan di dalam blok program lainnya dengan menyebutkan judul prosedurnya.

Procedure adalah sekumpulan instruksi yang dibungkus yang akan dipanggil dalam program utama.

```
Bentuk penulisan :
PROGRAM judul-program;
  PROCEDURE judul-prosedur;
  BEGIN
    ...
  END;

BEGIN
   ....
END.
```

```
Contoh :
PROGRAM prosedur;
USES WINCRT;
PROCEDURE garis;
BEGIN
  WRITELN('----------------');
END;

BEGIN
  CLRSCR;
  garis;
  WRITELN('BELAJAR PROSEDUR');
  garis;
  READLN;
END.


Hasil :
-------------------------------
BELAJAR PROSEDUR
-------------------------------
```

Punya parameter dan tidak punya parameter

Parameter →nama peubah yang dideklarasikan pada bagian header prosedur.

Parameter dibagi 2, yaitu :

a. Value Parameter (berfungsi sebagai input pada program prosedur)
b. Variabel Parameter (digunakan oleh prosedur untuk berkomunikasi dengan pemanggilnya →sebagai output → untuk pass by reference)

Parameter di prosedur disebut juga formal parameter.

(parameter yang dideklarasikan di bagian header). Untuk pemanggil disebut actual parameter.

PARAMETER DALAM PROSEDUR

Nilai di dalam suatu modul program Pascal sifatnya adalah lokal, artinya hanya dapat digunakan pada modul atau unit program yang bersangkutan saja, tidak dapat digunakan pada modul atau unit program yang lain.

PENGIRIMAN PARAMETER SECARA NILAI

Bila parameter dikirim secara nilai (by value), parameter formal di prosedur akan berisi nilai yang dikirimkan yang kemudian bersifat lokal di prosedur. Bila nilai parameter formal di dalam prosedur tersebut berubah, tidak akan mempengaruhi nilai parameter nyata.

Pengiriman nilai ini merupakan pengiriman searah yang tidak dikirimkan balik dari parameter formal ke parameter nyata. Parameter yang digunakan dengan pengiriman secara nilai ini disebut dengan parameter nilai (value parameter)

```
Contoh :
PROGRAM parameter_value;
USES WINCRT;
PROCEDURE hitung(A,B,C : INTEGER);
BEGIN
   C:=A+B;
   WRITELN('Nilai di Prosedur');
   WRITELN('NILAI A= ',A,' NILAI B= ',B,' NILAI C= ',C);
END;

VAR
  X,Y,Z : INTEGER;
BEGIN
  CLRSCR;
  WRITE('NILAI X= '); READLN(X);
  WRITE('NILAI Y= '); READLN(Y);
  WRITE('NILAI Z= '); READLN(Z);
  hitung(X,Y,Z);
  WRITELN('Nilai setelah Prosedur');
   WRITELN('NILAI X= ',X,' NILAI Y= ',Y,' NILAI Z= ',Z);
  READLN;
END.
```

Hasil :

NILAI X= 3

NILAI Y= 4

Nilai parameter Z sebelum dan sesudah prosedur sama

NILAI Z= 5 Nilai di Prosedur

NILAI A= 3 NILAI B= 4 NILAI C= 7

Nilai setelah Prosedur

NILAI X= 3 NILAI Y= 4 NILAI Z= 5

PENGIRIMAN PARAMETER SECARA ACUAN

Bila pengiriman parameter secara acuan (by reference), maka perubahan-perubahan yang terjadi pada nilai parameter formal di prosedur akan mempengaruhi nilai parameter nyata. Parameter-parameter ini disebut dengan *variabel parameter* serta dideklarasikan di deklarasi prosedur dengan menggunakan kata cadangan VAR.

```
Contoh :
PROGRAM parameter_reference;
USES CRT;
PROCEDURE hitung(VAR A,B,C : INTEGER);
BEGIN
   C:=A+B;
   WRITELN('Nilai di Prosedur');
   WRITELN('NILAI A= ',A,' NILAI B= ',B,' NILAI C= ',C);
END;
```

```
VAR
  X,Y,Z : INTEGER;
BEGIN
  CLRSCR;
  WRITE('NILAI X= '); READLN(X);
  WRITE('NILAI Y= '); READLN(Y);
  WRITE('NILAI Z= '); READLN(Z);
  hitung(X,Y,Z);
  WRITELN('Nilai setelah Prosedur');
  WRITELN('NILAI X= ',X,' NILAI Y= ',Y,' NILAI Z= ',Z);
  READLN;
END.
```

Hasil :

```
NILAI X= 3
NILAI Y= 4
NILAI Z= 0
Nilai parameter Z sebelum dan sesudah prosedur berbeda
Nilai di Prosedur NILAI A= 3 NILAI B= 4 NILAI C= 7
Nilai setelah Prosedur
NILAI X= 3 NILAI Y= 4 NILAI Z= 7
```

(lanjutan)

```
procedure lingkaran;
  const phi = 3.14;
var
  r,luas,keliling : real;
begin
  write('Masukkan nilai jari-jari = '); readln(r);
  luas:=phi*sqr(r);
  keliling:=2*phi*r;
  writeln('Luas Lingkaran = ',luas:2:2);
  writeln('Keliling Lingkaran = ',keliling:2:2);
 end;
```

Variabel lokal

Main program
lingkaran; → **pemanggil sub rutin**

Formal VS Actual Parameter

- Actual parameter dan formal parameter harus digunakan secara berurutan.
- Jumlah kedua parameter harus sama.
- Tipe data juga harus sama
- Actual parameter yang berhubungan dengan value parameter dapat mempunyai banyak bentuk.

→ Sub rutin hendaknya jangan memakai variabel global!!!!

(Contoh 1)

Procedure contoh(G:real; var H:integer; I:char);

Cara memanggilnya :

- contoh(R,S,T);
- contoh(3.4,S,'A');
- contoh(sqr(R)+2,S,T);
- contoh(3,S,T);

(Contoh 2)

```
Procedure contoh(Z:integer);
Begin
Z:=Z+1;
Writeln(Z);
end;
```

- Dipanggil dengan :

```
A:=3;
Contoh(A);
Writeln(A);
```

- Hasilnya????

(Contoh 3)

```
Procedure tukar(satu,dua:integer);
Var Temp:integer;
Begin
Temp:=satu;
Satu:=dua;
Dua:=temp;
End;
```

- Pemanggilnya:

```
Readln(C,D);
Tukar(C,D);
Writeln(C,D);
```

- Output?????

(Contoh 4)

```
Procedure conto_maning(P:integer);
Begin
   if P>5 then P:=1
   Else P:=0;
   Writeln(P);
End;
```

- Pemanggilnya:

```
E:=5;
Writeln(E);
Conto_maning(E);
Writeln(E);
```

(Contoh 5)

```
Procedure latih(N:integer; P:char);
Begin
Writeln('Output 1',N,P);
N:=1; P:='Z';
Writeln('Output2',N,P);
End;
```

- Pemanggilnya:

```
I:=0; C:='A';
Latih(I,C);
Writeln('Output3',I,C);
```

PROSEDUR TERSARANG

Prosedur tersarang (nested procedure) adalah prosedur yang berada di dalam prosedur yang lainnya.

```
Contoh :
PROGRAM nested_prosedur;
USES CRT;

PROCEDURE kesatu;
  PROCEDURE kedua;
  BEGIN
     WRITELN('Prosedur KEDUA ini ada di dalam prosedur KESATU');
  END;

 PROCEDURE ketiga;
  BEGIN
     WRITELN('Prosedur KETIGA ini juga ada di dalam prosedur KESATU');
  END;
```

```
BEGIN
   WRITELN('Ini Prosedur KESATU akan memanggil Prosedur
  KEDUA & KETIGA');
   kedua;
   ketiga;
END;

BEGIN
  CLRSCR;
  WRITELN('Ini program di modul utama akan memanggil
  Prosedur KESATU');
  kesatu;
  READLN;
END.
```

Hasil :
Ini program di modul utama akan memanggil Prosedur KESATU
Ini Prosedur KESATU akan memanggil Prosedur KEDUA & KETIGA
Prosedur KEDUA ini ada di dalam prosedur KESATU
Prosedur KETIGA ini juga ada di dalam prosedur KESATU

Function

- Merupakan sub rutin yang selain menerima masukan juga memberikan output kepada pemanggilnya.
- Bentuk :
  1. Function nama_fungsi : tipe_data;
  2. Function nama_fungsi(parameter) : tipe_data;
- Parameter pada fungsi dapat diubah dengan memberi var.
- Biasanya tidak perlu diubah-ubah!!!!

(lanjutan)

```
Function contoh(X,Y:integer): integer;
Begin
Contoh:=X+Y;
End;
```

- Pemanggilnya:

```
A:=contoh(B,C);
B:=contoh(A+B,C+2*D);
C:=contoh(contoh(A,4),contoh(B,-2));
```

PROSEDUR MEMANGGIL DIRINYA SENDIRI

Prosedur memanggil dirinya sendiri merupakan suatu prosedur yang memanggil atau menggunakan prosedur itu juga. Proses dari suatu program yang memanggil dirinya sendiri dikenal dengan nama *recursion*. Proses ini membutuhkan lebih banyak memori dalam komputer.

Contoh :

```
PROGRAM rekursi_prosedur;
USES winCRT;
VAR
  I : INTEGER;

PROCEDURE rekursi;
BEGIN
   I:=I+1;
   WRITELN('PASCAL');
   IF I < 10 THEN rekursi;
END;

BEGIN
  CLRSCR;
  I:=0; rekursi;
  READLN;
END.
```

Hasil :

```
PASCAL
PASCAL
PASCAL
PASCAL
PASCAL
PASCAL
PASCAL
PASCAL
PASCAL
PASCAL
```

LATIHAN

Buat program untuk menghitung gaji harian PT. XYZ dengan ketentuan:

- Gaji pokok karyawan Rp. 20000/jam
- Bila karyawan bekerja lebih dari 7 jam/hari maka kelebihannya dihitung lembur yang besarnya 1.5 dari gaji pokok
- Untuk karyawan yang bekerja 8 jam/hari atau lebih akan mendapat tambahan uang makan sebesar Rp. 5000
- Karyawan yang bekerja 9 jam/hari atau lebih akan mendapat uang transport lembur sebesar Rp. 4000

Program ini akan terdiri dari 4 buah prosedur : pokok, lembur, makan, jasa

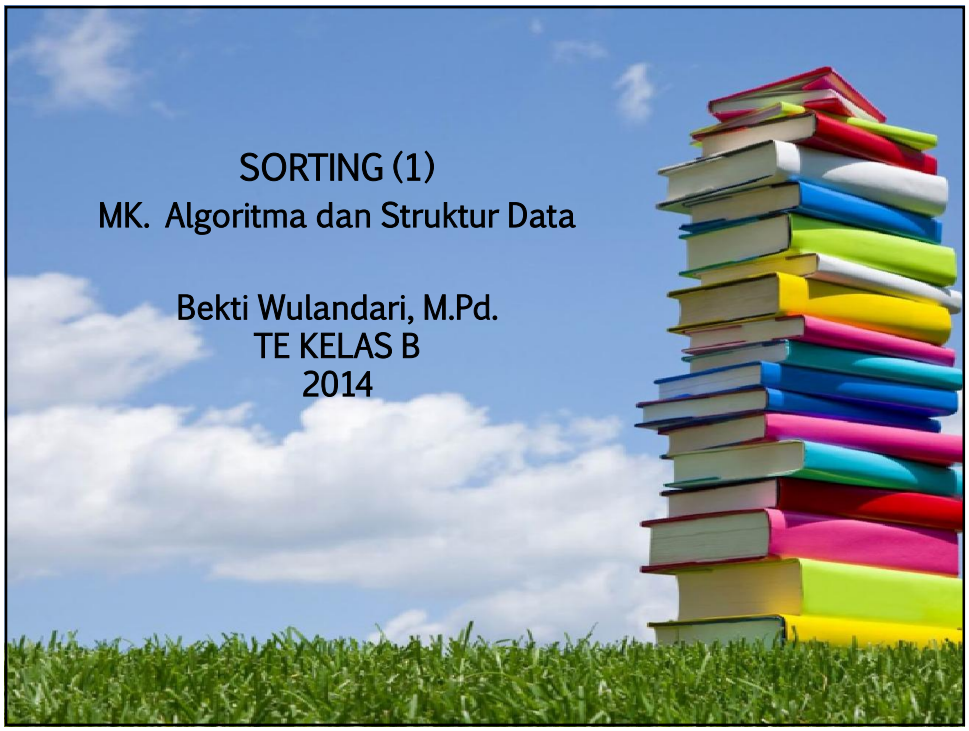
Input : NIP, Nama, Jumlah jam kerja

Output :

LAPORAN GAJI HARIAN KARYAWAN
PT. XYZ

Bulan : April 2011

NIP	Nama	Gaji Pokok	Lembur	Uang makan	Transport Lembur



Pendahuluan

- Pengurutan data dalam struktur data sangat penting, baik untuk data yang bertipe data numerik maupun karakter.
- Pengurutan dapat dilakukan secara ascending (naik) maupun descending (turun).
- Pengurutan adalah proses menyusun kembali data yang acak menjadi susunan yang teratur menurut aturan tertentu.

Pengertian

- *Sorting* = pengurutan
- *Sorted* = terurut menurut kaidah/aturan tertentu
- Data pada umumnya disajikan dalam bentuk *sorted*.
- Contoh:
  - Data Mahasiswa
  - Kata-kata dalam kamus
  - File-file di dalam sebuah directory
  - Indeks sebuah buku
  - Data mutasi rekening tabungan
  - dll

TUJUAN

- Mudah dalam Membaca data
- Mudah dalam menemukan data
- Penyajian data lebih teratur
- dll

Metode Sorting

- Exchange Sort
- Selection Sort
- Insertion Sort
- Bubble Sort
- Quick Sort
- Shell Sort
- Binary Insertion Sort

Metode Pengurutan Data

- Pengurutan berdasarkan perbandingan (*comparison-based sorting*)
  - Bubble sort, exchange sort
- Pengurutan berdasarkan prioritas (*priority queue sorting method*)
  - Selection sort, heap sort (menggunakan tree)
- Pengurutan berdasarkan penyisipan dan penjagaan terurut (*insert and keep sorted method*)
  - Insertion sort, tree sort
- Pengurutan berdasarkan pembagian dan penguasaan (*devide and conquer method*)
  - Quick sort, merge sort
- Pengurutan berkurang menurun (*diminishing increment sort method*)
  - Shell sort (pengembangan insertion)

Bubble Sort

- Metode sorting termudah
- Bubble Sort mengurutkan data dengan cara membandingkan elemen sekarang dengan elemen berikutnya.
- Ascending :
  if elemen sekarang **>** dari elemen berikutnya then kedua elemen **ditukar**
- Descending:
  if elemen sekarang **<** dari elemen berikutnya then kedua elemen **ditukar**

Bubble Sort

- Membandingkan data ke-1 dengan data ke-2, jika data ke-1 **lebih besar**, maka kedua **data ditukar**.
- Kemudian membandingkan data ke-2 dengan data ke-3, jika data ke-2 **lebih besar**, kedua **data ditukar** lagi.
- Demikian seterusnya sampai data terakhir, sehingga data kedudukannya akan bergeser-geser.
- Untuk proses 2, pembandingan (pergeseran data) hanya sampai pada data terakhir dikurangi satu.
- Bubble sort berhenti jika seluruh array telah diperiksa dan tidak ada pertukaran lagi yang bisa dilakukan, serta tercapai perurutan yang telah diinginkan.

Bubble Sort

- Algoritma:
  banyaknya data: n
  Data diurutkan/disorting dari yang bernilai besar

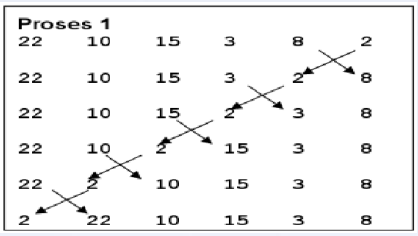
<u>Proses</u>

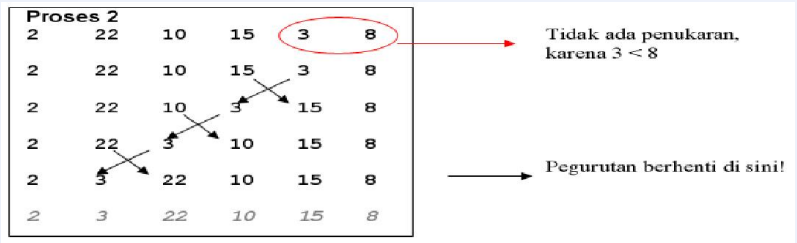
step 1 :
Periksalah nilai dua elemen mulai dari urutan ke-n sampai urutan ke-1. Jika nilai kiri<kanan, tukarkan kedua data itu.

step 2 :
Periksalah nilai dua elemen mulai dari urutan ke-n sampai urutan ke-2. Jika nilai kiri<kanan, tukarkan kedua data itu.

step n-1 :
Periksalah nilai dua elemen mulai dari urutan ke-n sampai urutan ke-n-1. Jika nilai kiri<kanan, tukarkan kedua data itu.

Bubble Sort





Bubble Sort

Proses 3

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

→ Pegurutan berhenti di sini!

Proses 4

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

→ Tidak ada penukaran, karena 10 < 15

→ Pegurutan berhenti di sini!

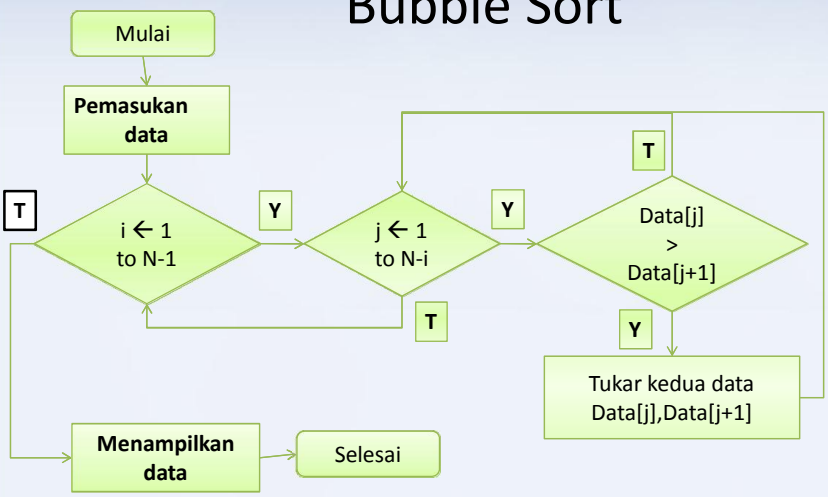
Bubble Sort

Proses 5

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

→ Pegurutan berhenti di sini!

Bubble Sort



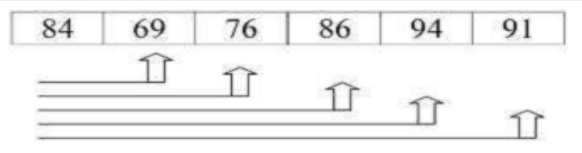
Exchange Sort

- Semua data dibandingkan dengan data pembanding, dimana pada akhir proses data terbesar akan berada pada akhir urutan.
- **Pada proses 1**: data ke-1 dibandingkan dengan data ke-2 **jika data ke-1 lebih besar** maka kedua data **langsung ditukar**. Kemudian data ke-1 dibandingkan lagi dengan data ke-3, lebih besar? Tukar! Demikian seterusnya.
- **Pada proses 2**: data ke-2 dibandingkan dengan data ke-3 jika data ke-2 lebih besar maka kedua data langsung ditukar. Kemudian data ke-2 dibandingkan lagi dengan data ke-4, lebih besar? Tukar! Demikian seterusnya.
- Dan seterusnya…..

Exchange Sort

- Pebedaan Exchange dengan Buble: dalam hal bagaimana membandingkan antar elemen-elemennya.
  - Exchange sort membandingkan **suatu elemen** dengan **elemen-elemen lainnya** dalam array tersebut, dan melakukan pertukaran elemen jika perlu. Jadi ada elemen yang selalu menjadi elemen **pusat (pivot).**
  - Sedangkan Bubble sort akan membandingkan **elemen pertama/terakhir** dengan **elemen sebelumnya/sesudahnya**, kemudian elemen tersebut itu akan menjadi **pusat (pivot)** untuk dibandingkan dengan elemen sebelumnya/sesudahnya lagi,

Exchange Sort



Proses 1

Pivot (Pusat)

84	69	76	86	94	91
84	69	76	86	94	91
84	69	76	**86**	94	91
86	69	76	**84**	**94**	91
94	69	76	84	**86**	91
94	69	76	84	86	91

Exchange Sort

Proses 2

Pivot (Pusat)

94	69	76	84	86	91
94	76	69	84	86	91
94	84	69	76	86	91
94	86	69	76	84	91
94	91	69	76	84	86

Proses 3

Pivot (Pusat)

94	91	69	76	84	86
94	91	76	69	84	86
94	91	84	69	76	86
94	91	86	69	76	84

Exchange Sort

Proses 4

Pivot (Pusat)

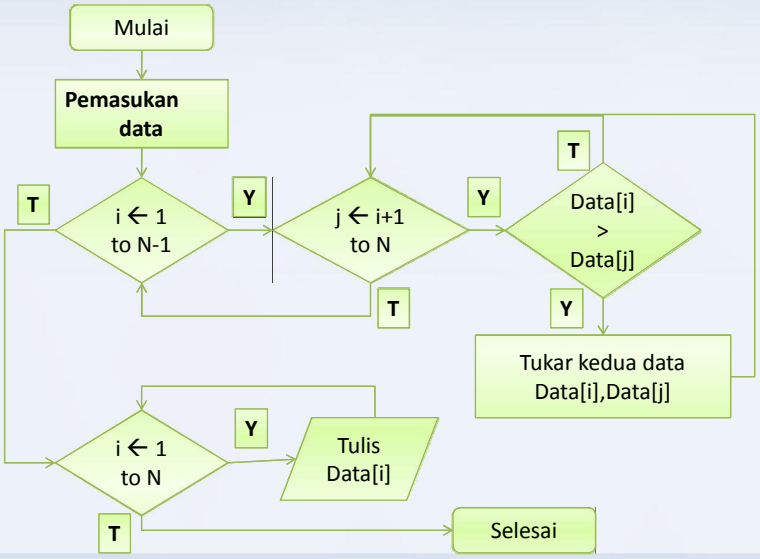
94	91	86	69	76	84
94	91	86	76	69	84
94	91	86	84	69	76

Proses 5

Pivot (Pusat)

94	91	86	84	69	76
94	91	86	84	76	69

Exchange Sort



Selection Sort

- Hampir sama dengan Exchange Sort, bedanya yang ditukar adalah **indeknya**.
- Penukaran data dilakukan di akhir proses.
- **Pada proses 1:** data ke-1 dibandingkan dengan data ke-2 jika data ke-1 lebih besar maka **indek kedua data ditukar**. Kemudian data ke-1 dibandingkan lagi dengan data ke-3, lebih besar? Indek ditukar! Demikian seterusnya, setelah selesai data ditukar.
- **Pada proses 2:** data ke-2 dibandingkan dengan data ke-3 jika data ke-2 lebih besar maka indek kedua data ditukar. Kemudian data ke-2 dibandingkan lagi dengan data ke-4, lebih besar? Indek ditukar! Demikian seterusnya, setelah selesai data ditukar.
- Dan seterusnya…..

Selection Sort

Proses 1

0	1	2	3	4	5
32	75	69	58	21	40

Pembanding	Posisi
32 < 75	0
32 < 69	0
32 < 58	0
32 > 21 (tukar idx)	4
21 < 40	4

Tukar data ke-0 (**32**) dengan data ke-4 (**21**)

0	1	2	3	4	5
21	75	69	58	32	40

Proses 2

0	1	2	3	4	5
21	**75**	69	58	32	40

Pembanding	Posisi
75 > 69 (tukar idx)	2
69 > 58 (tukar idx)	3
58 > 32 (tukar idx)	4
32 < 40	4

Tukar data ke-1 (**75**) dengan data ke-4 (**32**)

0	1	2	3	4	5
21	32	69	58	75	40

Proses 3

0	1	2	3	4	5
21	32	**69**	58	75	40

Pembanding	Posisi
69 > 58 (tukar idx)	3
58 < 75	3
58 > 40	5

Tukar data ke-2 (**69**) dengan data ke-5 (**40**)

0	1	2	3	4	5
21	32	40	58	75	69

Proses 4

0	1	2	3	4	5
21	32	40	**58**	75	69

Pembanding	Posisi
58 < 75	3
58 < 69	3

Tukar data ke-3 (**58**) dengan data ke-3 (**58**)

0	1	2	3	4	5
21	32	40	58	75	69

Proses 5

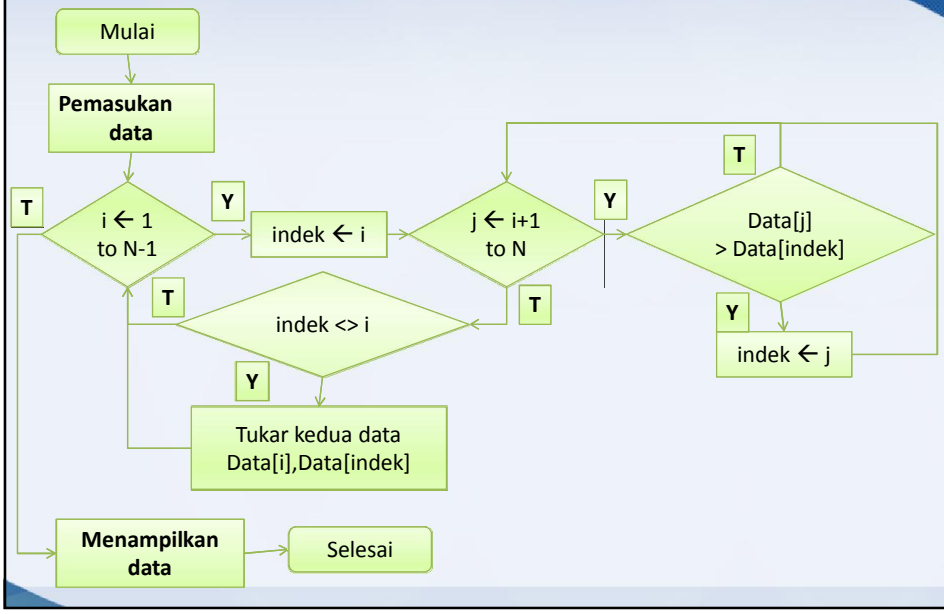
0	1	2	3	4	5
21	32	40	58	**75**	69

Pembanding	Posisi
75 > 69	5

Tukar data ke-4 (**75**) dengan data ke-5 (**69**)

0	1	2	3	4	5
21	32	40	58	69	75

Selection Sort



Insertion Sort

- Mirip dengan cara orang mengurutkan kartu selembar demi selembar kartu diambil dan disisipkan (**insert**) ke tempat yang seharusnya.
- Pengurutan dimulai dari data ke-2 sampai dengan data terakhir, jika ditemukan data yang lebih kecil atau lebih besar, maka akan ditempatkan (diinsert) diposisi yang seharusnya.
- Pada penyisipan elemen, maka elemen-elemen lain akan bergeser ke belakang

Insertion Sort

Proses 1

0	1	2	3	4	5
22	10	15	3	8	2

Temp	Cek	Geser
10	Temp<22?	Data ke-0 ke posisi 1

Temp menempati posisi ke -0

0	1	2	3	4	5
10	22	15	3	8	2

Proses 2

0	1	2	3	4	5
10	22	15	3	8	2

Temp	Cek	Geser
15	Temp<22	Data ke-1 ke posisi 2
15	Temp>10	-

Temp menempati posisi ke-1

0	1	2	3	4	5
10	15	22	3	8	2

Proses 3

0	1	2	3	4	5
10	15	22	3	8	2

Temp	Cek	Geser
3	Temp<22	Data ke-2 ke posisi 3
3	Temp<15	Data ke-1 ke posisi 2
3	Temp<10	Data ke-0 ke posisi 1

Temp menempati posisi ke-0

0	1	2	3	4	5
3	10	15	22	8	2

Proses 4

0	1	2	3	4	5
3	10	15	22	8	2

Temp	Cek	Geser
8	Temp<22	Data ke-3 ke posisi 4
8	Temp<15	Data ke-2 ke posisi 3
8	Temp<10	Data ke-1 ke posisi 2
8	Temp>3	-

Temp menempati posisi ke-1

0	1	2	3	4	5
3	8	10	15	22	2

Insertion Sort

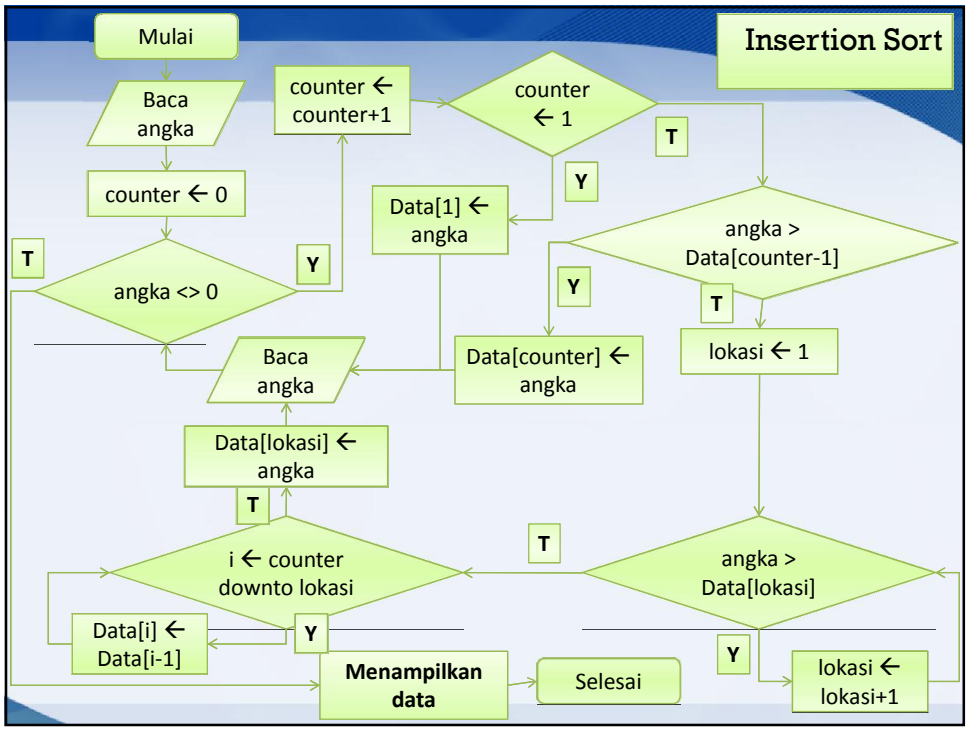
Proses 5

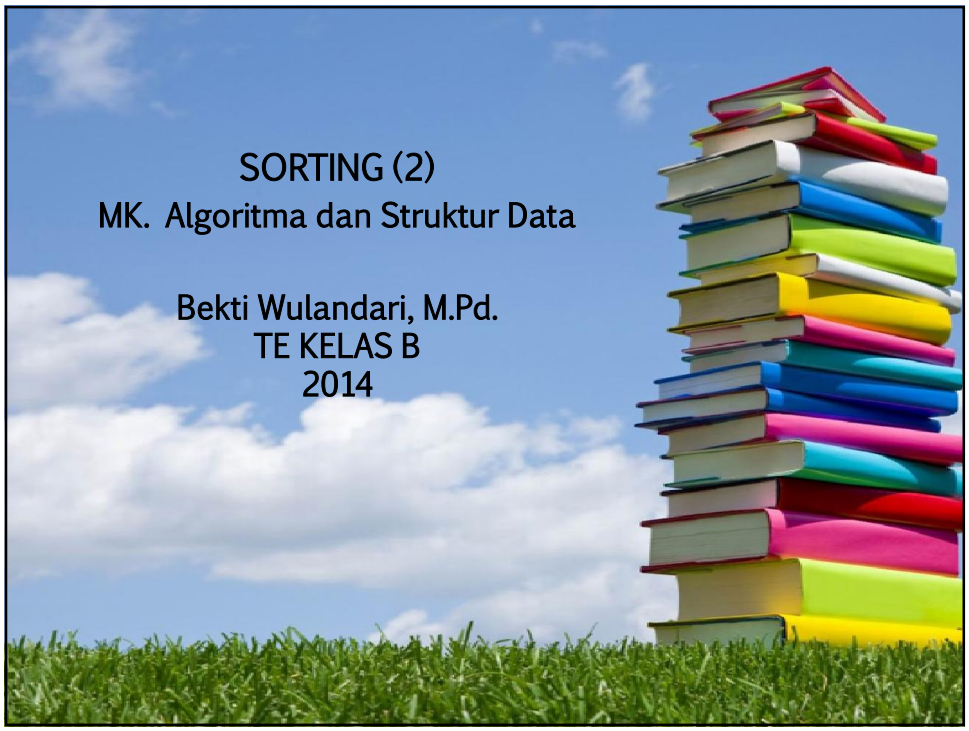
0	1	2	3	4	5
3	8	10	15	22	**2**

Temp	Cek	Geser
2	Temp<22	Data ke-4 ke posisi 5
2	Temp<15	Data ke-3 ke posisi 4
2	Temp<10	Data ke-2 ke posisi 3
2	Temp<8	Data ke-1 ke posisi 2
2	Temp<3	Data ke-0 ke posisi 1

Temp menempati posisi ke-0

0	1	2	3	4	5
2	3	8	10	15	22





Shell Sort

- Penemu : Donald Shell
- Metode perbandingan dan pertukaran
- Perbandingan dimulai dari separuh array yang akan disortir dengan separuh bagian yang lain.
- Contoh :
  - Jika terdapat 100 elemen, diperbandingkan elemen 1 dan elemen 51, elemen 2 dan elemen 52 dst. Selanjutnya algoritma akan membandingkan elemen 1 dan elemen 26, elemen 2 dan elemen 27 dst.

Penjelasan algoritma

- Sebelum masuk putaran ditentukan range dan target
- Pada putaran ke-`1`, `range = banyak/2`
- Dilakukan pengulangan dari 1 sampai dengan `banyak`/2.

  (Tiap putaran dimulai dari `counter=1` sampai dengan `counter=target )`

Penjelasan algoritma

- Pada tiap counter dilakukan proses : `kiri = counter` dan selanjutnya,
- `item(kiri)` dibandingkan dengan `item(kanan)` dimana : `kanan = kiri + range`
- Jika `item(kiri) >= item(kanan)` maka proses selesai dan dilanjutkan counter atau mungkin putaran berikutnya

Penjelasan algoritma

- Jika `item(kiri) < item(kanan)` maka terjadi pertukaran, selanjutnya :
- jika `item kiri < range` maka proses selesai dan dilanjutkan counter berikutnya
- jika `kiri > range` maka `kiri = kiri – range` dan proses dimulai dari awal pembandingan `item(kiri)` dan `item(kanan)` lagi
- Jika semua `counter` pada suatu putaran telah selesai maka range akan dihitung kembali yaitu : `range = range/2`.
- Jika `range <> 0` maka program akan dijalankan sampai `range = 0` berarti data telah terurut

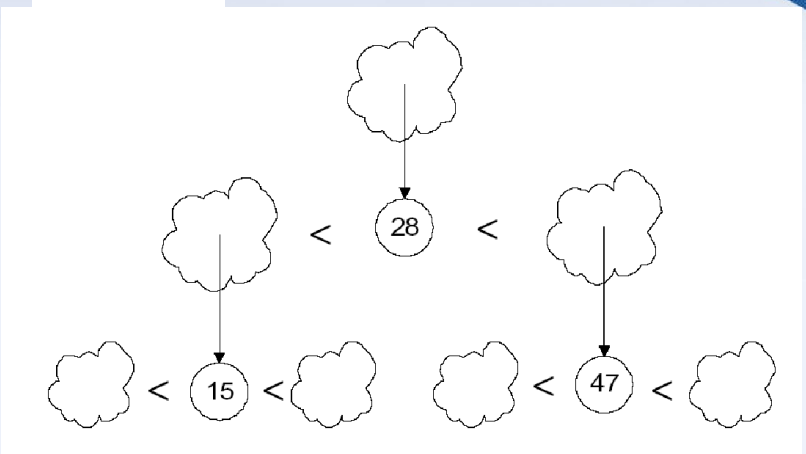
Shell Sort

Elemen ke	1	2	3	4	5	6	7	8	9	10
Nilai awal	14	6	23	18	7	47	2	83	16	38
Untuk Range 5										
Setelah putaran ke-1	47	6	23	18	7	14	2	83	16	38
Setelah putaran ke-2	47	6	23	18	7	14	2	83	16	38
Setelah putaran ke-3	47	6	83	18	7	14	2	23	16	38
Setelah putaran ke-4	47	6	83	18	7	14	2	23	16	38
Setelah putaran ke-5	47	6	83	18	38	14	2	23	16	7
Untuk Range 2										
Setelah putaran ke-6	83	6	47	18	38	14	2	23	16	7
Setelah putaran ke-7	83	18	47	6	38	14	2	23	16	7
Setelah putaran ke-8	83	18	47	6	38	14	2	23	16	7
Setelah putaran ke-9	83	18	47	14	38	6	2	23	16	7
Setelah putaran ke-10	83	18	47	14	38	6	2	23	16	7
Setelah putaran ke-11	83	23	47	18	38	14	2	6	16	7
Setelah putaran ke-12	83	23	47	18	38	14	16	6	2	7
Setelah putaran ke-13	83	23	47	18	38	14	16	7	2	6
Untuk Range 1										
Setelah putaran ke-14	83	23	47	18	38	14	16	7	2	6
Setelah putaran ke-15	83	47	23	18	38	14	16	7	2	6
Setelah putaran ke-16	83	47	23	18	38	14	16	7	2	6
Setelah putaran ke-17	83	47	38	23	18	14	16	7	2	6
Setelah putaran ke-18	83	47	38	23	18	14	16	7	2	6
Setelah putaran ke-19	83	47	38	23	18	16	14	7	2	6
Setelah putaran ke-20	83	47	38	23	18	16	14	7	2	6
Setelah putaran ke-21	83	47	38	23	18	16	14	7	2	6
Setelah putaran ke-22	83	47	38	23	18	16	14	7	6	2

Simulasi

Tentukan data acuan (data paling tengah).

Ide Quicksort

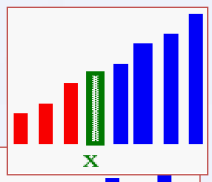


Tentukan “pivot”. Bagi Data menjadi 2 Bagian yaitu Data kurang dari dan Data lebih besar dari pivot. Urutkan tiap bagian tersebut secara rekursif.

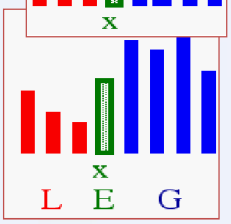
Ide Quicksort

1. **Tentukan pivotnya**

2. **Divide (Bagi):** Data disusun sehingga x berada pada posisi akhir E



3. **Recur and Conquer**: Diurutkan secara rekursif



Quicksort

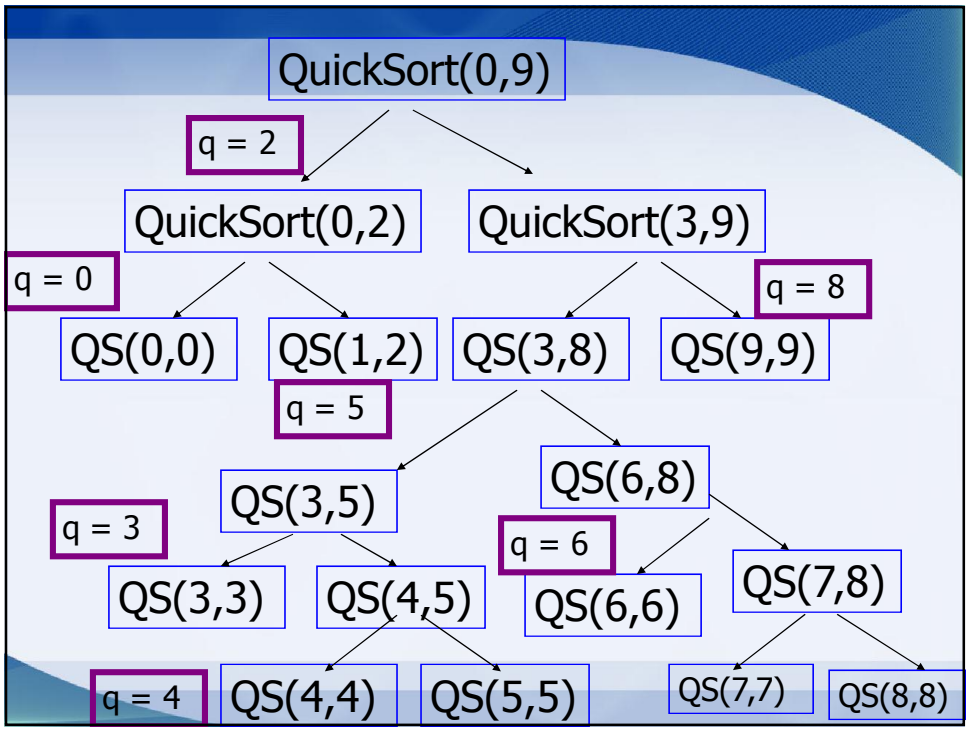
- Algoritma divide-and-conquer
  - array A[p..r] is *dipartisi* menjadi dua subarray yang tidak empty A[p..q] and A[q+1..r]
    - Invariant: Semua elemen pada A[p..q] lebih kecil dari semua elemen pada A[q+1..r]
  - Subarray diurutkan secara rekursif dengan memanggil quicksort

Partisi

- Jelas, semua kegiatan penting berada pada fungsi **`partition()`**
  - Menyusun kembali subarray
  - Hasil akhir :
    - Dua subarray
    - Semua elemen pada subarray pertama ≤ semua nilai pada subarray kedua
  - Mengembalikan indeks pivot yang membagi subarray

Partisi

- Partition(A, p, r):
  - Pilih elemen sebagai "pivot" (*which?*)
  - Dua bagian A[p..i] and A[j..r]
    - Semua element pada A[p..i] <= pivot
    - Semua element pada A[j..r] >= pivot
  - Increment i sampai A[i] >= pivot
  - Decrement j sampai A[j] <= pivot
  - Swap A[i] dan A[j]
  - Repeat Until i >= j
  - Return j



12	35	9	11	3	17	23	15	31	20

QuickSort(0,9)

- X = PIVOT merupakan indeks ke –0
- PIVOT = 12
- terdapat variabel i dan j , i=0 , j=9
- variabel i untuk mencari bilangan yang lebih besar dari PIVOT. Cara kerjanya : selama Data[i] < PIVOT maka nilai i ditambah.
- variabel j untuk mencari bilangan yang lebih kecil dari PIVOT. Cara kerjanya : selama Data[j] > PIVOT maka nilai j dikurangi

q = Partition(0,9)

12	35	9	11	3	17	23	15	31	20

PIVOT = 12
i = 0 j = 4
i < j maka SWAP

SWAP

3	35	9	11	12	17	23	15	31	20

PIVOT = 12
i = 1 j = 3
i < j maka SWAP

SWAP

3	11	9	35	12	17	23	15	31	20

3	11	9	35	12	17	23	15	31	20

PIVOT = 12
i = 3 j = 2
i < j (False) NO SWAP
Return j = 2
Q = Partisi = 2

QuickSort(0,9)

QuickSort(0,2) QuickSort(3,9)

QuickSort(0,2)

3	11	9

35	12	17	23	15	31	20

PIVOT = 3

i = 0 j = 0

i < j (False) NO SWAP

Return j = 0

Q = Partisi = 0

QuickSort(0,0) QuickSort(1,2)

QuickSort(1,2)

PIVOT = 11

i = 1 j = 2

i < j SWAP

3	9	11

35	12	17	23	15	31	20

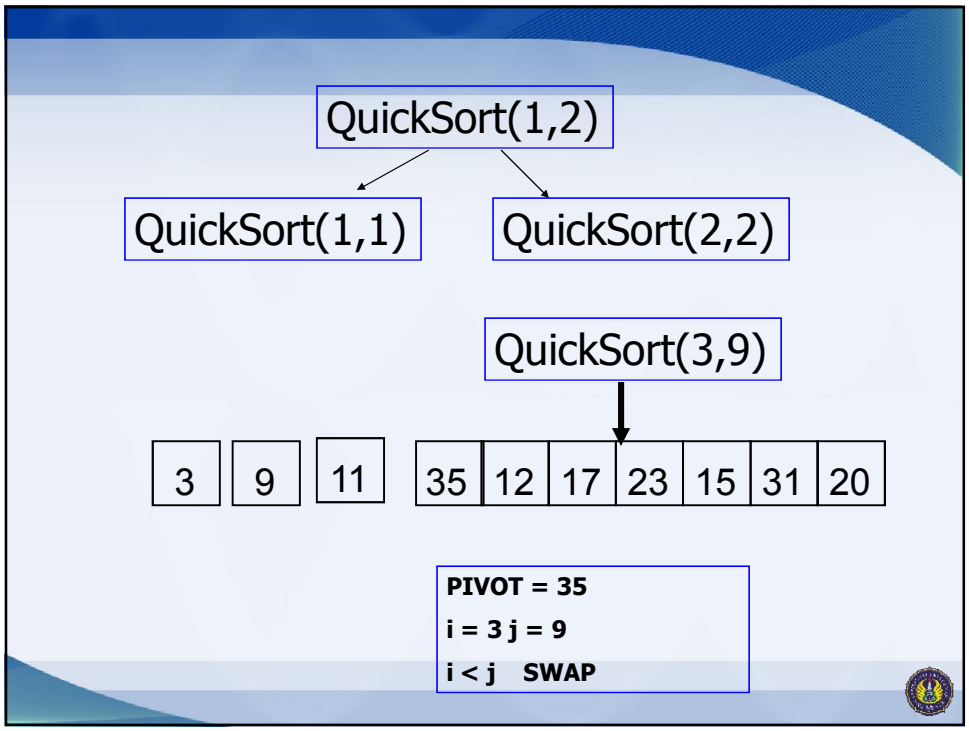
PIVOT = 11

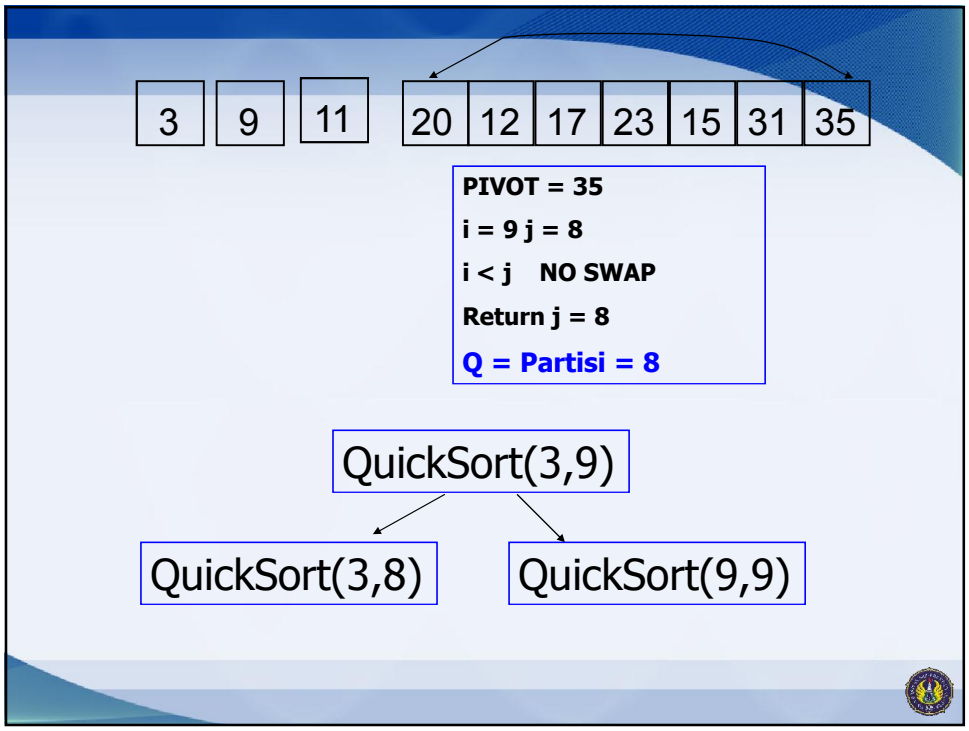
i = 2 j = 1

i < j NO SWAP

Return j = 1

Q = Partisi = 1


QuickSort(1,2)
QuickSort(1,1)
QuickSort(2,2)
QuickSort(3,9)
3
9
11
35
12
17
23
15
31
20
PIVOT = 35
i = 3 j = 9
i < j SWAP


3
9
11
20
12
17
23
15
31
35
PIVOT = 35
i = 9 j = 8
i < j NO SWAP
Return j = 8
Q = Partisi = 8
QuickSort(3,9)
QuickSort(3,8)
QuickSort(9,9)

3	9	11	20	12	17	23	15	31	35

QuickSort(3,8)

PIVOT = 20
i = 3 j = 7
i < j SWAP

3	9	11	15	12	17	23	20	31	35

PIVOT = 20
i = 6 j = 5
i < j NO SWAP
Return j = 5
Q = Partisi = 5

QuickSort(3,8)

QuickSort(3,5) QuickSort(6,8)

PIVOT = 15
i = 3 j = 4
i < j SWAP

3	9	11	15	12	17	23	20	31	35

PIVOT = 15
i = 4 j = 3
i < j NO SWAP
Return j = 3
Q = Partisi = 3

QS(3,5)

q = 3

QS(3,3) QS(4,5)

3 9 11 12 15 17 23 20 31 35

QS(4,5)

PIVOT = 15
i = 4 j = 4
i < j NO SWAP
Return j = 4
Q = Partisi = 4

→

q = 4

QS(4,5)

QS(4,4) QS(5,5)

3 9 11 12 15 17 23 20 31 35

QuickSort(6,8) →

PIVOT = 23
i = 6 j = 7
i < j SWAP

3 9 11 12 15 17 20 23 31 35

PIVOT = 23
i = 7 j = 6
i < j NO SWAP
Return j = 6
Q = Partisi = 6

→ QS(6,8)

q = 6

QS(6,6) QS(7,8)

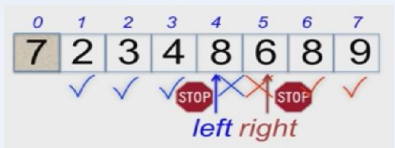
3	9	11	12	15	17	20	23	31	35

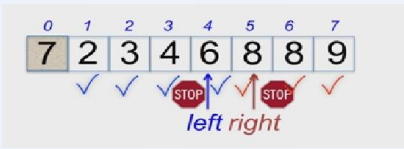
QS(7,8)

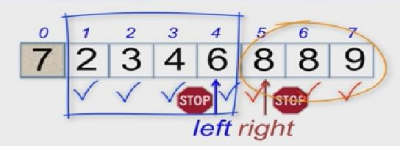
PIVOT = 23
i = 7 j = 7
i < j NO SWAP
Return j = 7
Q = Partisi = 7

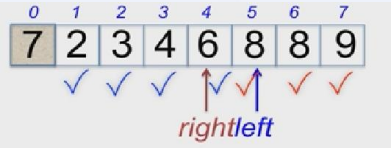
QS(7,8)

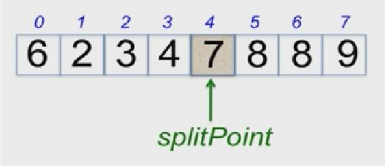
QS(7,7) QS(8,8)

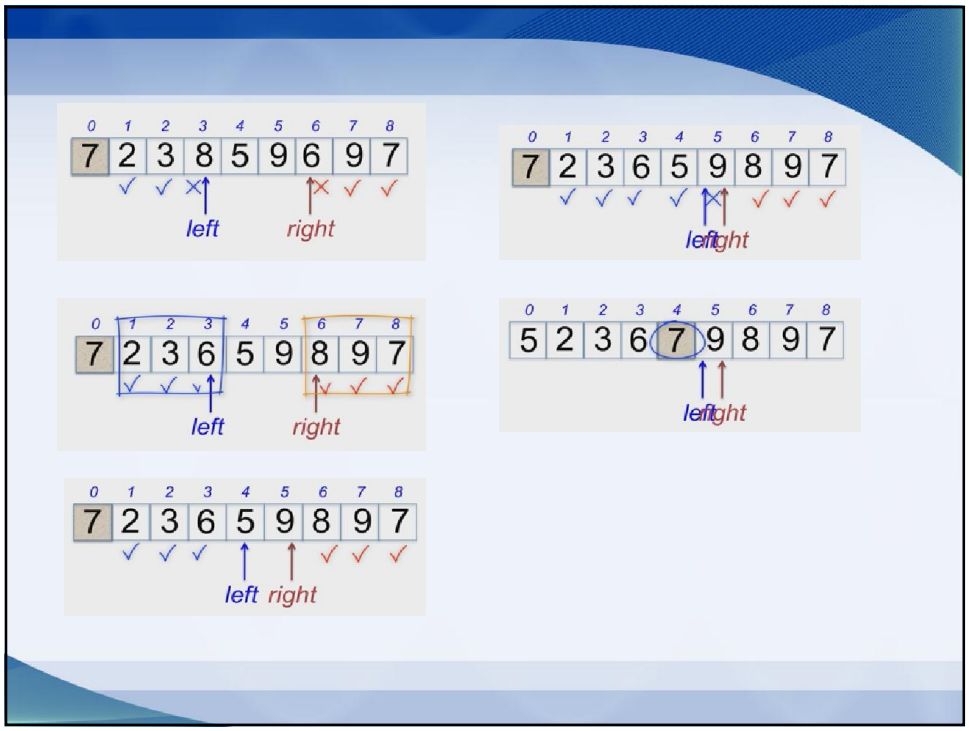


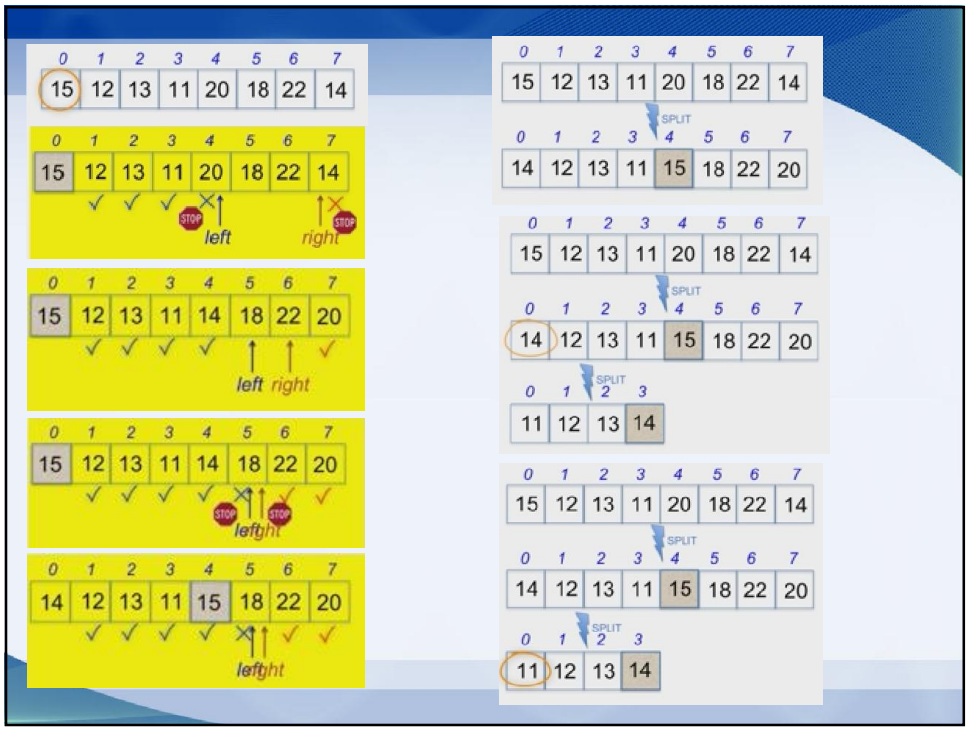


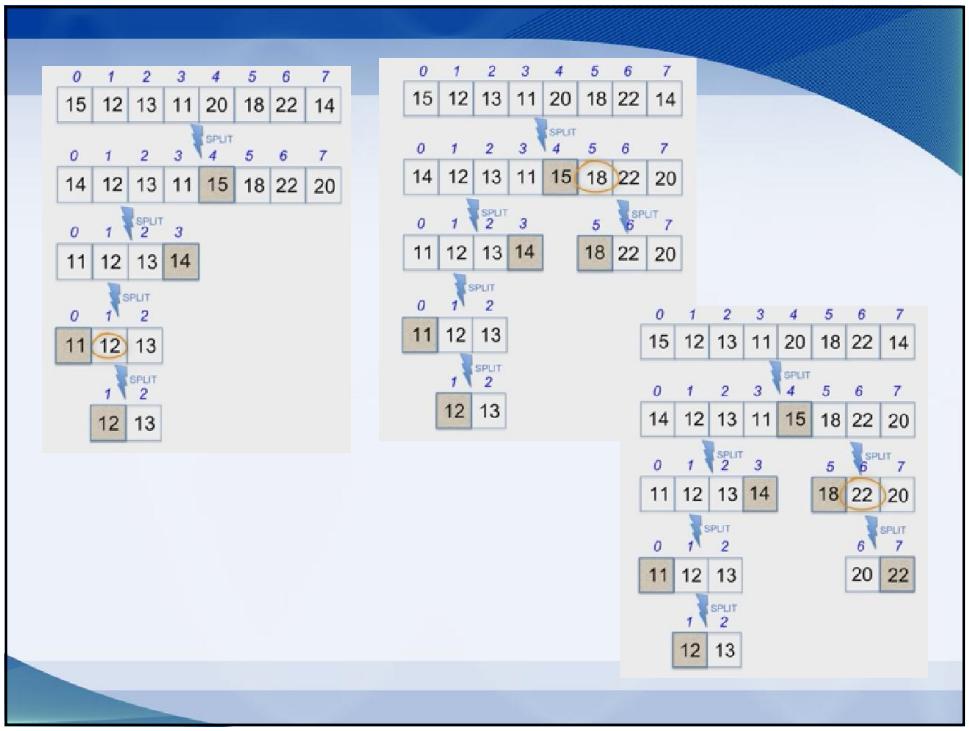







0 1 2 3 4 5 6 7 8
7 2 3 8 5 9 6 9 7
left
right
0 1 2 3 4 5 6 7 8
7 2 3 6 5 9 8 9 7
left
right
0 1 2 3 4 5 6 7 8
7 2 3 6 5 9 8 9 7
left
right
0 1 2 3 4 5 6 7 8
5 2 3 6 7 9 8 9 7
left
right
0 1 2 3 4 5 6 7 8
7 2 3 6 5 9 8 9 7
left
right


0 1 2 3 4 5 6 7
15 12 13 11 20 18 22 14
STOP
left
right
15 12 13 11 14 18 22 20
left
right
STOP
left
right
14 12 13 11 15 18 22 20
left
right
SPLIT
14 12 13 11 15 18 22 20
SPLIT
11 12 13 14
SPLIT
14 12 13 11 15 18 22 20
SPLIT
11 12 13 14


0 1 2 3 4 5 6 7
15 12 13 11 20 18 22 14
SPLIT
0 1 2 3 4 5 6 7
14 12 13 11 15 18 22 20
SPLIT
0 1 2 3
11 12 13 14
SPLIT
0 1 2
11 12 13
SPLIT
1 2
12 13
0 1 2 3 4 5 6 7
15 12 13 11 20 18 22 14
SPLIT
0 1 2 3 4 5 6 7
14 12 13 11 15 18 22 20
SPLIT
0 1 2 3
11 12 13 14
SPLIT
0 1 2
11 12 13
SPLIT
1 2
12 13
SPLIT
5 6 7
18 22 20
0 1 2 3 4 5 6 7
15 12 13 11 20 18 22 14
SPLIT
0 1 2 3 4 5 6 7
14 12 13 11 15 18 22 20
SPLIT
0 1 2 3
11 12 13 14
SPLIT
0 1 2
11 12 13
SPLIT
1 2
12 13
SPLIT
5 6 7
18 22 20
SPLIT
6 7
20 22

ALGORITMA DAN STRUKTUR DATA

SEARCHING

Bekti Wulandari, M.Pd
TE Kelas B
2014

Pencarian (Searching)

- Pada suatu data seringkali dibutuhkan pembacaan kembali informasi (*information retrieval*) dengan cara searching.
- Searching adalah menemukan nilai (data) tertentu di dalam sekumpulan data yang bertipe sama
- Tahapan paling penting pada searching: memeriksa jika data yang dicari sama dengan data yang ada pada deret data.

Algoritma Pencarian

1. Input x (x=data yang dicari)
2. Bandingkan x dengan deret data
3. Jika ada data yang sama cetak pesan “Ada”
4. Jika tidak ada data yang sama cetak pesan “tidak ada”.

Algoritma Pencarian

- Macam algoritma pencarian :
  - Sequantial Search
  - Binary Search

(1) Sequential Search

- Disebut juga linear search atau Metode pencarian beruntun.
- Adalah proses membandingkan setiap elemen larik satu per satu secara beruntun, mulai dari elemen pertama sampai elemen yang dicari ditemukan atau seluruh elemen sudah diperiksa.
- Data awal = tidak harus dalam kondisi terurut.
- Pencarian beruntun terbagi dua:
  1. Pencarian beruntun pada larik tidak terurut;
  2. Pencarian beruntun pada larik terurut

Algoritma Sequential Search

1. Input x (data yang dicari)
2. Bandingkan x dengan data **ke-i sampai n**
3. Jika ada data yang sama dengan x maka cetak pesan “Ada”
4. Jika tidak ada data yang sama dengan x cetak pesan “tidak ada”

Ilustrasi Sequential searching:

Data	12	5	7	3	14	28	9	40	6	21
Indeks ke-	0	1	2	3	4	5	6	7	8	9

→ Misal nilai yang dicari adalah X = 28, maka elemen yang diperiksa :
- → Data[0]=12 <> 28,
- → Data[1]= 5 <> 28,
- → Data[2]= 7 <> 28,
- → Data[3]= 3 <> 28,
- → Data[4]= 14 <> 28,
- → Data[5]= 28 (DITEMUKAN!)

Indeks larik yang dikembalikan: iX = 5 (Posisi ke-6)

→ Misal nilai yang dicari adalah X = 17, maka elemen yang diperiksa : 12, 5, 7, 3, 14, 28, 9, 40, 6, dan 21 (TIDAK DITEMUKAN!)

Q & A

- **Problem:** Apakah cara di atas efisien? Jika datanya ada 10000 dan semua data dipastikan unik?
- **Solution**: Untuk meningkatkan efisiensi, seharusnya jika data yang dicari sudah ditemukan maka perulangan harus dihentikan!
  - **Hint**: Gunakan **break**!
- **Question**: Bagaimana cara menghitung ada berapa data dalam array yang tidak unik, yang nilainya sama dengan data yang dicari oleh user?
  - **Hint**: Gunakan variabel counter yang nilainya akan selalu bertambah jika ada data yang ditemukan!

Pencarian pada larik terurut

Pada contoh kasus sebelumnya, data dapat diurutkan terlebih dahulu sehingga diperoleh urutan :

Data	3	5	6	7	9	12	14	21	28	40
Indeks ke-	0	1	2	3	4	5	6	7	8	9

→ Misal nilai yang dicari adalah X = 10, maka pencarian dilakukan dengan membandingkan data yang dicari dengan elemen data indeks ke-0 sampai dengan data indeks ke-5,

--> karena data indeks ke-5, yaitu 12 ternyata sudah bernilai lebih besar daripada data yang dicari, yaitu 10. --> Sehingga tidak perlu membandingkan lagi untuk data berikutnya.

Kesimpulannya, data 10 ternyata TIDAK DITEMUKAN di dalam larik.

Sentinel Sequential Search

- Yang dimaksud dengan sentinel adalah elemen fiktif yang sengaja ditambahkan sesudah elemen terakhir larik.
- Jika elemen larik terakhir L[N], maka sentinel dipasang pada elemen L[N+1].
- Sentinel ini harganya sama dengan elemen yang dicari.
- Akibatnya proses pencarian selalu berhasil menemukan data yang dicari. Walaupun demikian harus diperiksa lagi letak data tersebut ditemukan, apakah:
  1. Di atara elemen-elemen larik sesungguhnya, yaitu L[1]…L[N]
  2. Pada elemen fiktif (L[N+1]) berarti X sesungguhnya tidak terdapat di dalam larik L.

Sequential Search with Sentinel

- Perhatikan array data berikut ini:

0	1	2	3	4	5	6	indeks
3	12	9	-4	21	6		value

- Terdapat 6 buah data dalam array (dari indeks 0 s/d 5) dan terdapat 1 indeks array tambahan (indeks ke 6) yang belum berisi data (disebut sentinel)
- Array pada indeks ke 6 berguna untuk menjaga agar indeks data berada pada indeks 0 s/d 5 saja. Bila pencarian data sudah mencapai array indeks yang ke-6 maka berarti data TIDAK ADA, sedangkan jika pencarian tidak mencapai indeks ke-6, maka data ADA.

Best & Worst Case

- **Best case** : jika data yang dicari terletak di depan sehingga waktu yang dibutuhkan minimal.
- **Worst case** : jika data yang dicari terletak di akhir sehingga waktu yang dibutuhkan maksimal.
- Contoh :

  DATA = 5 6 9 2 8 1 7 4

  bestcase ketika x = 5

  worstcase ketika x = 4

  *x = key/data yang dicari

Latihan

- Buatlah flowchart dari algoritma Sequential Search!

(2) Binary Search

- Teknik pencarian = data dibagi menjadi dua bagian untuk setiap kali proses pencarian.
- Adalah metode pencarian yang diterapkan pada sekumpulan data yang sudah terurut (terurut menaik atau terurut menurun).
- Harus dilakukan proses sorting terlebih dahulu untuk data awal.
- Mencari posisi tengah :

Posisi tengah = (posisi awal + posisi akhir) div 2

Algoritma Binary Search

Misalkan indek kiri adalah L dan indek kanan adalah R.

Kondisi awal L = 1 dan R = N.

LANGKAH 1 :

1. Data diambil dari posisi awal 1 dan posisi akhir N
2. Bagi dua elemen larik sehingga ditemukan elemen tengahnya dengan rumus = (L+R) div 2
3. Elemen tengah (data[m]) membagi larik menjadi dua bagian, yaitu bagian kiri data[L..m-1] dan bagian kanan data[m+1..R]

Algoritma Binary Search

LANGKAH 2 :

1. Periksa apakah data[m] = X.
2. Jika data[m] = X, pencarian dihentikan sebab X sudah ditemukan.
3. Tetapi, jika data[m] ≠ X, harus ditentukan apakah pencarian akan dilakukan di larik bagian kiri atau di bagian kanan.
4. Jika data[m] < X, maka pencarian dilakukan pada bagian kanan. Sebaliknya, jika data[m] > X, pencarian dilakukan pada larik bagian kiri.

LANGKAH 3 :

→ Ulangi langkah 1 sampai X ditemukan atau L > R (ukuran larik sudah nol).

Ilustrasi

Contoh Data:

Misalnya data yang dicari **15**

L = 0, N = 8, indeks tengah= (0+8) div 2 = 4

0	1	2	3	4	5	6	7	8
3	**9**	**11**	**12**	**15**	**17**	**23**	**31**	**35**
kiri								**kanan**

L [4] = 15?Ya, X ditemukan dan proses pencarian selesai

Misal data yang dicari adalah 17

0	1	2	3	4	5	6	7	8
3	**9**	**11**	**12**	**15**	**17**	**23**	**31**	**35**
	kiri				**kanan**			

L [4] = 17? Tidak, sehingga harus diputuskan pencarian dilakukan di sebelah kanan ato kiri.

Karena 17 > 15 (data tengah), maka: awal = tengah + 1

5	6	7	8
17	**23**	**31**	**35**

langkah selanjutnya :

1. L = 5, N = 8, indeks tengah= (5+8) div 2 = 6

5	6	7	8
17	**23**	**31**	**35**
kiri		**kanan**	

2. L [6] = 17? Tidak, sehingga harus diputuskan pencarian dilakukan di sebelah kanan ato kiri.

   Karena 17 < 23 (data tengah), maka: akhir = tengah – 1

5
17

3. Karena 17 = 17 (data tengah), maka KETEMU! Proses pencarian selesai

Best & Worst Case

- **Best case** : jika data yang dicari terletak di posisi tengah.
- **Worst case** : jika data yang dicari tidak ditemukan.
- Contoh :

  DATA = 5 6 9 2 8 1 7 4 3

  bestcase ketika x = 8 (T(n)=1)

  worstcase ketika x = 25 (T(n) = 5 atau n/2)

  *x = key/data yang dicari

Latihan

- Buatlah flowchart dari algoritma binary search!

Pakai yang mana?
Sequential Search atau Binary Search

1. SS dapat digunakan untuk data terurut dan data belum terurut sedangkan BS digunakan untuk data terurut
2. Ditinjau dari kinerja pencarian : SS memerlukan waktu yang sebanding dengan n (banyaknya data), BS membutuhkan waktu sebanding dengan $^{2}\log(n)$.

   Karena $^{2}\log(n) < n$ untuk $n > 1$, maka jika n semakin besar waktu pencarian dengan BS lebih sedikit dari pada SS

Contoh :

Larik yang berukuran n = 256 elemen

Algoritma SS melakukan perbandingan elemen sebanyak 256 kali.

Algoritma BS melakukan perbandingan sebanyak $^{2}\log(256) = 8$ kali

ALGORITMA DAN STRUKTUR DATA

Stack dan Queue

Bekti Wulandari, M.Pd
TE Kelas B
2014

Struktur Data Linear

- Adalah kumpulan komponen-komponen yang tersusun membentuk satu garis linear.
- Stack: struktur data linear dimana penambahan atau pengurangan komponen dilakukan di satu ujung saja.
- Queue: struktur data linear dimana penambahan komponen dilakukan di satu ujung, sementara pengurangan dilakukan di ujung lain (yang satu lagi).
- Kedua struktur tersebut merupakan struktur data abstrak dimana implementasi pada tingkat lebih rendah dapat menggunakan struktur *sequential* (array) atau struktur berkait (linear linked-list).

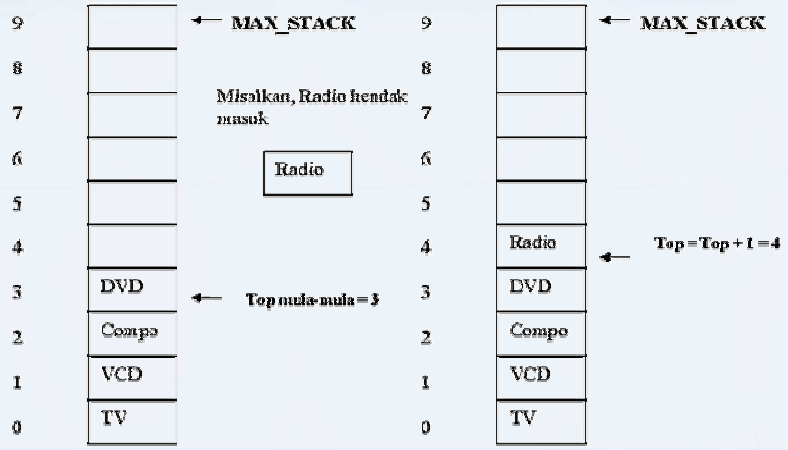
STACK (TUMPUKAN)

Stack (tumpukan) sebenarnya secara mudah dapat diartikan sebagai ***list (urutan) dimana penambahan dan pengambilan elemen hanya dilakukan pada satu sisi yang disebut <u>top (puncak)</u> dari stack.***

Dengan melihat definisi tersebut maka jelas bahwa pada stack berlaku aturan LIFO (Last In First Out), yaitu elemen yang terakhir masuk akan pertama kali diambil atau dilayani.

STACK (TUMPUKAN)

Ilustrasi stack



Operasi dalam stack

Operasi dasar pada stack adalah membuat tumpukan, memeriksa apakah sebuah tumpukan kosong, **PUSH** (operasi pemasukan elemen ke dalam stack) dan **POP** (operasi pengambilan satu elemen dari dalam stack).

- **Push** : digunakan untuk menambah item pada stack pada tumpukan paling atas
- **Pop** : digunakan untuk mengambil item pada stack pada tumpukan paling atas

Operasi tambahan

- **Clear** : digunakan untuk mengosongkan stack
- **IsEmpty** : fungsi yang digunakan untuk mengecek apakah stack sudah kosong
- **IsFull** : fungsi yang digunakan untuk mengecek apakah stack sudah penuh

Operasi dalam Stack

- Elemen paling kanan adalah elemen yang ada pada TOS (Top Of the Stack)
- stack yang dipakai bernama S
- PUSH(B,S) berarti memasukkan elemen B ke dalam stack S
- POP(B,S) berarti mengambil elemen dari stack S dan menaruhnya ke dalam variabel B

Operasi yang dilakukan	Isi Stack	Keterangan
Kondisi Awal	kosong	-
PUSH('A',S)	A	-
PUSH('B',S)	AB	-
PUSH('C',S)	ABC	-
POP(Data,S)	AB	Variabel Data berisi 'C'
PUSH('D',S)	ABD	-
POP(Data,S)	AB	Data berisi 'D'
POP(Data,S)	A	Data berisi 'B'

Operasi Stack

Implementasi dalam bahasa Pascal dapat dilakukan dengan memanfaatkan struktur data record dan array.

Array dipergunakan untuk menyimpan elemen-elemen yang dimasukkan. Selain itu diperlukan pula suatu variabel untuk mencatat banyaknya elemen yang ada di dalam array yang sekaligus menunjukkan TOS (Top of Stack)

- konstanta *maxelm* menyatakan banyaknya elemen maksimum yang dapat ditampung oleh stack
- *typeelemen* adalah tipe data yang akan disimpan di dalam stack (bisa integer, word, real, boolean, char , string atau lainya)
- *atas* adalah field yang menyatakan banyaknya elemen dalam stack saat itu, yang sekaligus menyatakan TOS

Implementasi dalam Pascal

Deklarasi tipe untuk tumpukan (stack):

```
type tumpukan = record
   atas :0..maxelm;
   isi : array[1..maxelm] of
typeelemen;
end;
```

Implementasi dalam Pascal

Selain prosedur untuk POP dan PUSH, kita dapat pula menambahkan sejumlah fungsi untuk membantu penanganan kesalahan diantaranya adalah fungsi **PENUHS** (untuk mengecek apakah stack penuh)
fungsi **KOSONGS** (untuk mengecek apakah stack kosong) dan fungsi **SIZES** (untuk mengetahui banyaknya elemen di dalam stack).

Implementasi dalam Pascal

Fungsi SIZES

```
Function SIZES(S : tumpukan):
integer;
begin
 SIZES := S.atas;
end;
```

Fungsi PENUHS

```
Function PENUHS(S : tumpukan):
boolean;
begin
 Jika S.atas = maxelm then
   PENUHS := true
 else
   PENUHS :=false ;
end;
```

Fungsi KOSONGS

```
Function KOSONGS(S :
tumpukan):boolean;
begin
 If S.atas = 0 then
   KOSONGS := true;
 else
   KOSONGS := false;
end;
```

Implementasi dalam Pascal

Procedure Push

```
procedure PUSH( var T:tumpukan; x:integer);
begin
     T.atas:=T.atas+1;
     T.isi[T.atas]:=x;
end;
```

Implementasi dalam Pascal

Procedure Push

```
procedure PUSH( var T:tumpukan; var penuh:boolean;
x:integer);
begin
   if T.atas = maxElm then penuh:=true
   else
   begin
     penuh := false;
     T.atas:=T.atas+1;
     T.isi[T.atas]:=x;
   end;
end;
```

Implementasi dalam Pascal

Procedure Pop

```
procedure POP( var T:tumpukan; var
habis:boolean;x:integer);
begin
   if T.banyak = 0 then habis:=true
   else
   begin
      habis := false;
      T.atas:=T.atas-1;
      X:=T.isi[T.atas];
   end;
end;
```

QUEUE (ANTRIAN)

QUEUE

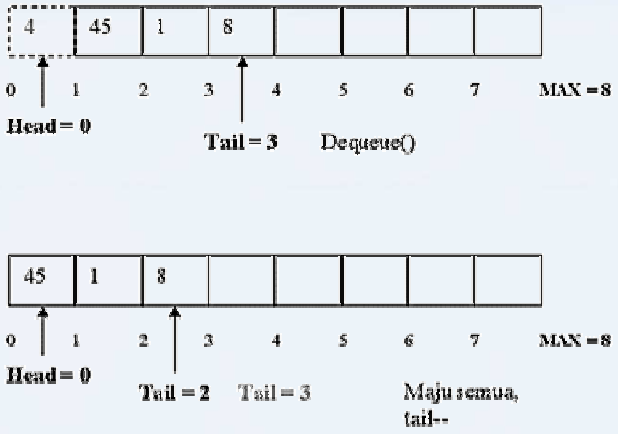
Queue atau antrian sebenarnya juga merupakan suatu list. Salah satu sifat yang membedakan queue dengan stack adalah bahwa pada queue penambahan elemen dilakukan pada salah satu ujung (ujung depan) dan pengambilan dilakukan pada ujung yang lain (ujung belakang).

Dengan demikian queue menggunakan prinsip **FIFO** (First In First Out), yaitu elemen yang pertama masuk akan pertama kali pula dikeluarkan.

Seperti pada stack, operasi-operasi dasar pada queue adalah operasi penambahan elemen (sebut "ADDQ") dan operasi pengambilan elemen (sebut DELQ).

QUEUE

Ilustrasi Queue



QUEUE

- elemen paling kanan adalah elemen yang ada pada ujung belakang (yang terakhir kali masuk)
- queue yang dipakai bernama Q
- ADDQ(Q,B) berarti memasukkan elemen B ke dalam queue Q
- DELQ(B,Q) berarti mengambil elemen dari queue Q dan menaruhnya ke dalam variabel B

Operasi yang dilakukan	Isi Queue	Keterangan
Kondisi Awal	kosong	-
ADDQ('A',Q)	A	-
ADDQ('B',Q)	AB	-
ADDQ('C',Q)	ABC	-
DELQ(Data,Q)	BC	Variabel Data berisi 'A'
ADDQ('D',Q)	BCD	-
DELQ(Data,Q)	CD	Data berisi 'B'
DELQ(Data,S)	D	Data berisi 'C'

Implementasi Queue

Implementasi dalam bahasa Pascal dapat dilakukan dengan memanfaatkan struktur data record dan array. Array dipergunakan untuk menyimpan elemen-elemen yang dimasukkan. Selain itu diperlukan pula suatu variabel untuk mencatat banyaknya elemen yang ada di dalam array. Pada implementasi

- konstanta *maxelm* menyatakan banyaknya elemen maksimum yang dapat ditampung oleh queue
- *typeelemen* adalah tipe data yang akan disimpan di dalam queue(bisa integer, word, real, boolean, char , string atau lainya)
- *Depan, belakang* adalah field yang menyatakan banyaknya elemen depan dan belakang dalam queue saat itu
- queue diimplementasikan sebagai array linier dengan memandang bahwa elemen terdepan selalu berada pada sel pertama (implementasi fisik), sehingga bila terjadi pengambilan satu elemen maka semua elemen di belakang elemen terambil (bila ada) harus digeser ke depan satu langkah

Deklarasi tipe untuk antrian (queue):

```
type antrian= record
   depan,belakang :0..maxelm;
   isi : array[1..maxelm] of typeelemen;
end;
```

Implementasi Queue dalam Pascal

Selain prosedur untuk ADDQ dan DELQ, kita dapat pula menambahkan sejumlah fungsi untuk membantu penanganan kesalahan diantaranya adalah fungsi **PENUHQ** (untuk mengecek apakah antrian penuh) fungsi**KOSONGQ** (untuk mengecek apakah antrian kosong) dan fungsi **SIZEQ** (untuk mengetahui banyaknya elemen di dalam queue).

Fungsi PENUHQ

```
Function PENUHQ(q : antrian): boolean;
begin
 Jika Q.banyak = maxelm then PENUHQ := true
 else PENUHQ := false;
end;
```

Fungsi KOSONGQ

```
Function KOSONGQ(q : antrian):boolean;
begin
 If Q.banyak = 0 then KOSONGQ := true
 else KOSONGQ := false;
end;
```

Implementasi Queue dalam Pascal

Procedure ADDQ

```
procedure ADDQ(data:integer; var q:queue);
var sisip :boolean;
i,j,pos:integer;
begin
    sisip:=false;
    i:=q.depan;
    while (q.isi[i]<>0) and (data>=q.isi[i]) do inc(i);
    if data<q.isi[i] then
       begin
          pos:=i;
          for j:=q.belakang downto pos do
          q.isi[j+1]:=q.isi[j];
          q.isi[pos]:=data;
          inc(q.belakang);
       end
    else
       if q.belakang<max then
       begin
          inc(q.belakang);
          q.isi[q.belakang]:=data;
       end;
end;
```

Implementasi Queue dalam Pascal

Prosedur DELQ

```
Procedure DeQueue(var q:queue; var hsl:integer);
var
   i:integer;
begin
   if q.belakang>0 then
   begin
     hsl:=q.isi[q.depan];
     dec(q.belakang);
     for i:=1 to q.belakang do
         q.isi[i]:=q.isi[i+1] ;
   end;
end;
```

Latihan

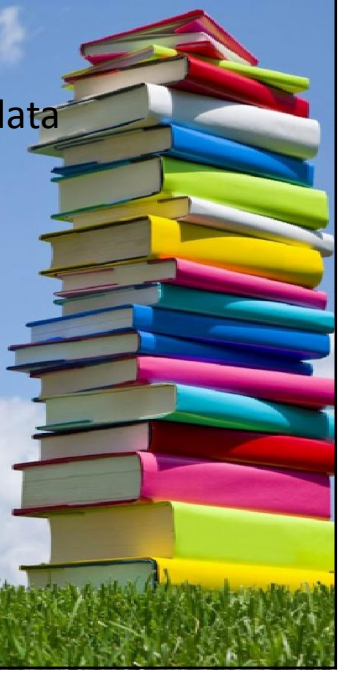
Aplikasi Stack

MK Algoritma dan Struktur data

Bekti Wulandari, M.Pd

TE kelas B

2014



Aplikasi Stack

- Untuk kalkulator Scientific.
- 3+4-2, ((2+4)*7)+3*(9–5), dan sebagainya.
- Ada 2 istilah: operand (angka) dan operator (+, -, *, /).
- Operasi aritmatika seperti 3+4-2 dan ((2+4)*7)+3*(9–5) disebut dengan infix.
- Infix: operator yang diletakkan di antara 2 operand.
- Postfix: operator yang mengikuti operand.
- Prefix: kebalikan dari postfix.
- Infix harus diterjemahkan ke dalam Postfix untuk membuat kalkulator Scientific.

Operator Aritmetik

- /, *, div, mod → level tinggi
- +, - → level rendah
- Mod dan div hanya untuk bilangan bulat!!!

- Contoh :
- 5 – 2 + 1 = ?
- 5 – 2 * 3 = ?
- (6 + 3 * 2) / 6 – 2 * 3 = ?

Infix dan Postfix

Infix : 3+4 →Postfix : 34+ → Prefix : +34

Infix	Postfix
A+B–C	AB+C–
A*B/C	AB*C/
A+B*C	ABC*+
A*B+C	AB*C+
A*(B+C)	ABC+*
A*B+C*D	AB*CD*+
(A+B)*(C–D)	AB+CD–*
((A+B)*C)–D	AB+C*D–
A+B*(C–D/(E+F))	ABCDEF+/–*+

Pengkalkulasian Infix oleh Manusia

1. Baca infix satu persatu dari kiri ke kanan.
2. Tugas utamanya adalah menemukan 2 operand dan 1 operator.

   Perhatikan juga apakah ada operator selanjutnya atau tidak. Jika tidak ada langsung dikalkulasi. Jika ada, perhatikan derajatnya, jika sederajat atau lebih rendah, kalkulasi. Jika lebih tinggi, tunda kalkulasi.
3. Lanjutkan pembacaan dan lakukan kalkulasi jika memungkinkan.

Pengkalkulasian Infix oleh Manusia

(lanjutan 1)

Infix: 3 + 4 - 5

Item Read	Expression Parsed So Far	Comments
3	3	
+	3+	
4	3+4	
–	7	When you see the –, you can evaluate 3+4.
	7–	
5	7–5	
End	2	When you reach the end of the expression, you can evaluate 7–5.

Pengkalkulasian Infix oleh Manusia

(lanjutan 2)

Infix: 2+4*5

Item Read	Expression Parsed So Far	Comments
3	3	
+	3+	
4	3+4	
*	3+4*	You can't evaluate 3+4 because * is higher precedence than +.
5	3+4*5	When you see the 5, you can evaluate 4*5.
	3+20	
End	23	When you see the end of the expression, you can evaluate 3+20.

Pengkalkulasian Infix oleh Manusia

(lanjutan 3)

Infix: 3*(4+5)

Item Read	Expression Parsed So Far	Comments
3	3	
*	3*	
(	3*(	
4	3*(4	You can't evaluate 3*4 because of the parenthesis.
+	3*(4+	
5	3*(4+5	You can't evaluate 4+5 yet.
)	3*(4+5)	When you see the), you can evaluate 4+5.
	3*9	After you've evaluated 4+5, you can evaluate 3*9.
	27	
End		Nothing left to evaluate.

Penerjemahan Infix ke Postfix

Infix: A+B-C

Character Read from Infix Expression	Infix Expression Parsed So Far	Postfix Expression Written So Far	Comments
A	A	A	
+	A+	A	
B	A+B	AB	
–	A+B–	AB+	When you see the –, you can copy the + to the postfix string.
C	A+B–C	AB+C	
End	A+B–C	AB+C–	When you reach the end of the expression, you can copy the –.

Penerjemahan Infix ke Postfix

(lanjutan 1)

Infix: A+B*C

Character Read from Infix Expression	Infix Expression Parsed So Far	Postfix Expression Written So Far	Comments
A	A	A	
+	A+	A	
B	A+B	AB	
*	A+B*	AB	You can't copy the + because * is higher precedence than +.
C	A+B*C	ABC	When you see the C, you can copy the *.
	A+B*C	ABC*	
End	A+B*C	ABC*+	When you see the end of the expression, you can copy the +.

Penerjemahan Infix ke Postfix

(lanjutan 2)

Infix: A*(B+C)

Character Read from Infix Expression	Infix Expression Parsed so Far	Postfix Expression Written So Far	Comments
A	A	A	
*	A*	A	
(	A*(	A	
B	A*(B	AB	You can't copy * because of the parenthesis.
+	A*(B+	AB	
C	A*(B+C	ABC	You can't copy the + yet.
)	A*(B+C)	ABC+	When you see the), you can copy the +.
	A*(B+C)	ABC+*	After you've copied the +, you can copy the *.
End	A*(B+C)	ABC+*	Nothing left to copy.

Penerjemahan Infix ke Postfix

(lanjutan 3)

Infix: A+B*(C-D)

Character Read from Infix Expression	Infix Expression Parsed So Far	Postfix Expression Written So Far	Stack Contents
A	A	A	
+	A+	A	+
B	A+B	AB	+
*	A+B*	AB	+*
(	A+B*(	AB	+*(
C	A+B*(C	ABC	+*(
–	A+B*(C–	ABC	+*(–
D	A+B*(C–D	ABCD	+*(–
)	A+B*(C–D)	ABCD–	+*(
	A+B*(C–D)	ABCD–	+*(
	A+B*(C–D)	ABCD–	+*
	A+B*(C–D)	ABCD–*	+
	A+B*(C–D)	ABCD–*+	

Penerjemahan Infix ke Postfix

(lanjutan 4)

Infix: A+B-C

Character Read from Infix	Infix Parsed So Far	Postfix Written So Far	Stack Contents	Rule
A	A	A		Write operand to output.
+	A+	A	+	If stack empty, push opThis.
B	A+B	AB	+	Write operand to output.
–	A+B–	AB		Stack not empty, so pop item.
	A+B–	AB+		opThis is –, opTop is +, opTop>=opThis, so output opTop.
	A+B–	AB+	–	Then push opThis.
C	A+B–C	AB+C	–	Write operand to output.
End	A+B–C	AB+C–		Pop leftover item, output it.

Penerjemahan Infix ke Postfix

(lanjutan 5)

Infix: A+B*C

Character Read From Infix	Infix Parsed So Far	Postfix Written So Far	Stack Contents	Rule
A	A	A		Write operand to postfix.
+	A+	A	+	If stack empty, push opThis.
B	A+B	AB	+	Write operand to output.
*	A+B*	AB	+	Stack not empty, so pop opTop.
	A+B*	AB	+	opThis is *, opTop is +, opTop<opThis, so push opTop.
	A+B*	AB	+*	Then push opThis.
C	A+B*C	ABC	+*	Write operand to output.
End	A+B*C	ABC*	+	Pop leftover item, output it.
	A+B*C	ABC*+		Pop leftover item, output it.

Penerjemahan Infix ke Postfix

(lanjutan 6)

A*(B+C)

Character Read From Infix	Infix Parsed So Far	Postfix Written So Far	Stack Contents	Rule
A	A	A		Write operand to postfix.
*	A*	A	*	If stack empty, push opThis.
(	A*(	A	*(	Push (on stack.
B	A*(B	AB	*(	Write operand to postfix.
+	A*(B+	AB	*	Stack not empty, so pop item.
	A*(B+	AB	*(	It's (, so push it.
	A*(B+	AB	*(+	Then push opThis.
C	A*(B+C	ABC	*(+	Write operand to postfix.
)	A*(B+C)	ABC+	*(	Pop item, write to output.
	A*(B+C)	ABC+	*	Quit popping if (.
End	A*(B+C)	ABC+*		Pop leftover item, output it.

Algoritma dan Struktur Data
Pointer

Bekti Wulandari, M.Pd
Kelas B TE
2014

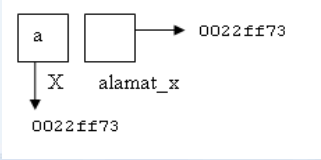
Pendahuluan

Cakupan bahasan

1. Mengenal tipe data Pointer.
2. Manipulasi memori lewat Pointer bertipe dan tak bertipe.
3. Linked List; meliputi operasi inisialisasi, menambah node baru, menyisipkan node baru, menghapus node yang berisi data, membaca data dari node.

Pendahuluan

Pointer merupakan suatu tipe data dalam Pascal yang berfungsi untuk menunjuk dan menyimpan alamat memori. Dalam penulisan pointer biasa digambar dengan panah, sedangkan bagian memori yang ditunjuk digambar dengan kotak, dan isinya ditulis di dalam kotak.



Ilustrasi Pointer

- Kita memiliki variabel X yang berisi nilai karakter 'a'
- Oleh kompiler pascal, nilai 'a' ini akan disimpan di suatu alamat tertentu di memori.
- Alamat variabel X dapat diakses dengan menggunakan statemen **&X**.
- Jika kita ingin menyimpan alamat dari variabel X ini, kita dapat menggunakan suatu variabel
  misalnya **char alamat_x = &X;**
- alamat_x adalah suatu variabel yang berisi alamat dimana nilai X, yaitu 'a' disimpan.
- Variabel alamat_x disebut variabel pointer atau sering disebut **pointer** saja.

Deklarasi Pointer

Bentuk umum dari deklarasi tipe pointer:

Untuk pointer **bertipe**:

```
<nama_var> : ^<tipe_data>;
```

Untuk pointer **tidak bertipe**:

```
<nama_var> : pointer;
```

Suatu pointer dapat menunjuk ke data **bertipe** *elementer*, *terstruktur*, *pointer yang lain*, atau *tidak bertipe*. Jika suatu pointer *tidak menunjuk ke mana-mana*, pointer itu dinamakan *dangling*, sedangkan bagian memori yang tidak dapat diakses karena tidak ada pointer yang menunjuk dinamakan *garbage* (sampah)

Deklarasi Pointer

Dalam Pascal, pointer dapat diisi dengan nilai yang berasal dari:

1. NIL
2. Fungsi Ptr
3. Operator @
4. Prosedur New dan GetMem
5. Pointer yang lain

NIL

Reserved word NIL

NIL merupakan reserved word dalam Pascal, di mana pointer yang bernilai NIL dianggap tidak menunjuk alamat memori manapun.NIL biasa digambarkan dengan lambang ground

Fungsi Ptr

Fungsi Ptr mengembalikan pointer dari segmen dan offset yang dimasukkan.

Sintaks:

```
Function Ptr(Seg, Ofs : word) : pointer;
```

dengan *Seg* : segmen memori.

Ofs : offset memori.

Operator @

Operator @ digunakan untuk mengambil alamat variabel yang akan ditunjuk.

Sintaks:

```
<nama_var>:=@<variabel_yang_alamatnya_diambil>;
```

Prosedur New dan GetMem

Prosedur **New** digunakan untuk memesan memori untuk *pointer bertipe*, sedangkan prosedur **GetMem** untuk *pointer tidak bertipe*. Kedua prosedur ini akan membentuk suatu variabel dinamik yang diletakkan dalam *Heap*.

Sintaks:

```
New(var P : pointer);
GetMem(var P : pointer, size : word);
```

Dengan *P* : pointer yang akan diisi.

Size : ukuran yang dipesan.

- Pointer yang belum digunakan sebaiknya diisi dengan NIL, dan untuk pointer yang telah menunjuk sebuah alamat yang sudah dipesan memorinya, isinya dapat dimanipulasi melalui pointer.

Contoh Penggunaan Pointer

```
program deklarasi;
      uses wincrt;
var
   p : ^integer;
   nilai : integer;
begin
clrscr;
   nilai:=12;
   p:=@nilai;
   writeln(p^);
   p^:=100;
   writeln(p^);
   writeln(nilai);
   readln;
end.
```

Variabel nilai diisi dengan nilai 12.

variabel p menunjuk alamat dari variabel nilai dengan operator @, sehingga variable p berisi nilai 12

variabel p diberi nilai 100

variabel nilai juga bernilai 100 karena sudah ditunjuk oleh variabel p

Pembahasan

Pada contoh program deklarasi ini, pertama-tama dideklarasikan variabel *p* sebagai pointer yang bertipe integer. Dibuat sebuah variabel lagi yang diberi nama *nilai* dan bertipe integer.

Variabel nilai diisi dengan nilai 12. Kemudian variabel p menunjuk alamat dari variabel nilai dengan operator @, sehingga variable p berisi nilai 12, dan ditampilkan outputnya di layar. Kemudian variabel p diberi nilai 100, dan secara otomatis variabel nilai juga bernilai 100 karena sudah ditunjuk oleh variabel p. Kemudian isi dari variabel p yang baru dan variabel nilai ditampilkan di layar.

LinkedList

PENDAHULUAN

- Dalam suatu linear list kita dapat melakukan operasi penyisipan atau penghapusan atas elemen-elemennya pada sembarang posisi.
- Misalkan ada 1500 item yang merupakan elemen dari suatu linear list.
- Jika elemen ke-56 akan kita keluarkan, maka elemen ke-1 s/d elemen ke-55 tidak akan berubah posisinya pada linear list tersebut. Tetapi elemen ke-57 akan menjadi elemen ke-56, elemen ke-58 akan menjadi elemen ke-57 dst. Selanjutnya, jika kita sisipkan satu elemen pada posisi setelah elemen ke-41, maka elemen ke-42 s/d elemen ke-1500 akan berubah posisinya.
- Untuk menyatakan keadaan diatas diperlukan suatu konsep yang berbeda dengan konsep sekuensial sebelumnya.
- Linked list merupakan suatu cara non-sekuensial yang digunakan untuk merepresentasikan suatu data.

DEFINISI

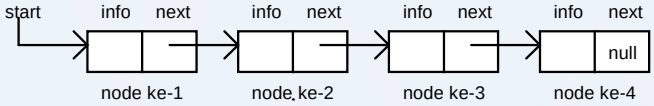
Linked list (one way list) adalah suatu kumpulan **elemen data** (yang disebut sebagai **node**) dimana urutannya ditentukan oleh suatu pointer.

Setiap elemen (node) dari suatu linked list terdiri atas dua bagian, yaitu :

1. INFO, berisi informasi tentang elemen data yang bersangkutan.
2. NEXT (link field/next pointer field), berisi alamat dari elemen node) selanjutnya yang dituju.

DEFINISI (1)

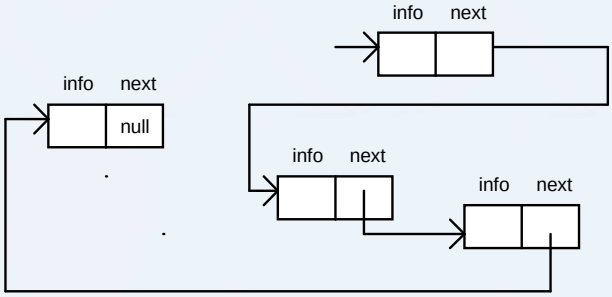
Berikut ini sebuah contoh linked list yang terdiri atas 4 node :



Pada node ke-4 field NEXT-nya berisi NULL, artinya node ke-4 tersebut adalah node terakhir

DEFINISI (2)

Node-node dalam linked list tidak harus selalu digambarkan paralel seperti pada gambar diatas. Linked list pada contoh diatas dapat pula digambarkan seperti berikut ini :



DEFINISI (3)

Ada dua hal yang menjadi kerugian dengan representasi suatu data dengan linked list ini yaitu :

1. Diperlukan ruang tambahan untuk menyatakan/tempat field pointer.
2. Diperlukan waktu yang lebih banyak untuk mencari suatu node dalam linked list.

Keuntungannya adalah :

1. Jenis data yang berbeda dapat di-link.
2. Operasi REMOVE atau INSERT hanya dilakukan dengan mengubah pointer-nya saja.

OPERASI DASAR PADA LINKED LIST

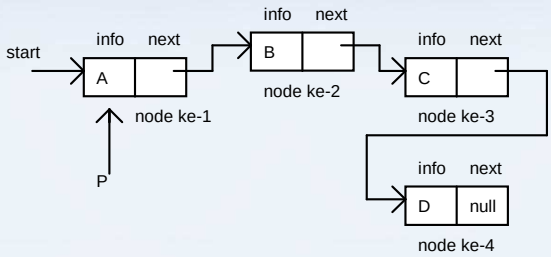
Ada beberapa aturan yang didefinisikan pada operasi didalam linked list, yaitu :

1. Jika P adalah suatu variabel pointer, maka nilainya adalah alamat atau lokasi dari variabel lain yang dituju.
2. Operasi yang didefinisikan pada suatu variabel pointer adalah :
    1. Test apakah sama dengan NULL.
    2. Test untuk kesamaan dengan variabel pointer lain.
    3. Menetapkan sama dengan NULL.
    4. Menetapkan menuju ke node lain.

Notasi yang didefinisikan sehubungan dengan operasi diatas adalah :

1. NODE(P), artinya node yang ditunjuk oleh pointer P.
2. INFO(P), artinya nilai INFO dari node yang ditunjuk pointer P.
3. NEXT(P), artinya hubungan (link) selanjutnya dari node yang ditunjuk oleh pointer P

OPERASI DASAR PADA LINKED LIST



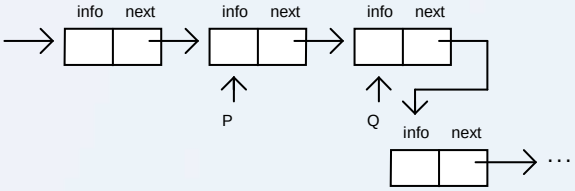
NODE(P) = node yang ditunjuk oleh P yaitu node pertama.
INFO(P) = A
NEXT(P) = node ke-dua
INFO(NEXT(NEXT(P))) = C

MENGHAPUS SUATU NODE DARI LINKED LIST (REMOVE)

- Untuk menghapus node dalam linked list digunakan procedure FREENODE.
- Jika Q adalah suatu variabel pointer, maka FREENODE(Q) akan menyebabkan node yang ditunjuk oleh variabel pointer Q dihapus dari linked list.
- Perhatikan linked list berikut :

langkah ke-1 :

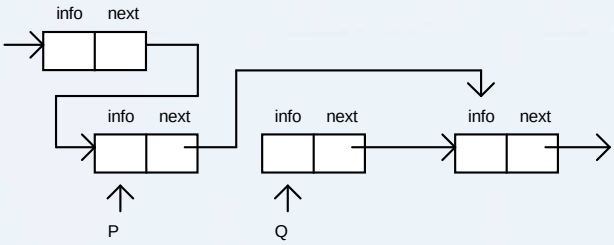
Q := Next(P)



MENGHAPUS SUATU NODE DARI LINKED LIST (REMOVE)

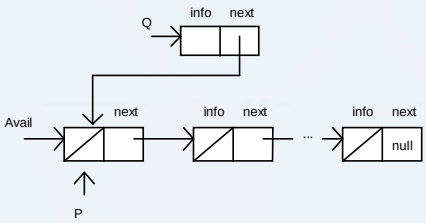
langkah ke-2 :

Next(P) := Next(Q)

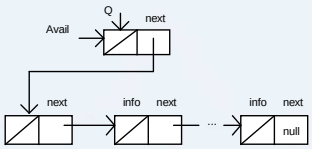


procedure Freenode(Q)

(a)Next(Q) := Avail



(c) Avail := Q



(b)Info(Q) := Null

MENYISIPKAN SUATU NODE KE DALAM LINKED LIST

- Untuk menyisipkan node dalam linked list digunakan procedure GETNODE.
- Jika NEW adalah suatu variabel pointer, maka GETNODE(NEW) akan menyebabkan node yang ditunjuk oleh variabel pointer NEW disisipkan ke dalam linked list.

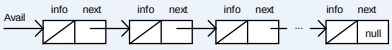
procedure Getnode(NEW)

if Avail = Null

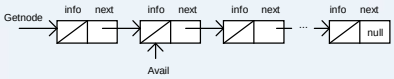
then out-of-free-space

(a) else begin

Getnode := Avail;



(b) Avail := Next(Avail);



(c) Next(Getnode) : = Null;

end;

Aplikasi Stack

Bekti Wulandari, M.Pd
TE kelas B
2014

Aplikasi Stack

- Untuk kalkulator Scientific.
- 3+4-2, ((2+4)*7)+3*(9–5), dan sebagainya.
- Ada 2 istilah: operand (angka) dan operator (+, -, *, /).
- Operasi aritmatika seperti 3+4-2 dan ((2+4)*7)+3*(9–5) disebut dengan infix.
- Infix: operator yang diletakkan di antara 2 operand.
- Postfix: operator yang mengikuti operand.
- Prefix: kebalikan dari postfix.
- Infix harus diterjemahkan ke dalam Postfix untuk membuat kalkulator Scientific.

Operator Aritmetik

- /, *, div, mod → level tinggi
- +, - → level rendah
- Mod dan div hanya untuk bilangan bulat!!!

- Contoh :
- 5 – 2 + 1 = ?
- 5 – 2 * 3 = ?
- (6 + 3 * 2) / 6 – 2 * 3 = ?

Infix dan Postfix

Infix : 3+4 →Postfix : 34+ → Prefix : +34

Infix	Postfix
A+B–C	AB+C–
A*B/C	AB*C/
A+B*C	ABC*+
A*B+C	AB*C+
A*(B+C)	ABC+*
A*B+C*D	AB*CD*+
(A+B)*(C–D)	AB+CD–*
((A+B)*C)–D	AB+C*D–
A+B*(C–D/(E+F))	ABCDEF+/–*+

Pengkalkulasian Infix oleh Manusia

1. Baca infix satu persatu dari kiri ke kanan.
2. Tugas utamanya adalah menemukan 2 operand dan 1 operator.

   Perhatikan juga apakah ada operator selanjutnya atau tidak. Jika tidak ada langsung dikalkulasi. Jika ada, perhatikan derajatnya, jika sederajat atau lebih rendah, kalkulasi. Jika lebih tinggi, tunda kalkulasi.
3. Lanjutkan pembacaan dan lakukan kalkulasi jika memungkinkan.

Pengkalkulasian Infix oleh Manusia

(lanjutan 1)

Infix: 3 + 4 - 5

Item Read	Expression Parsed So Far	Comments
3	3	
+	3+	
4	3+4	
–	7	When you see the –, you can evaluate 3+4.
	7–	
5	7–5	
End	2	When you reach the end of the expression, you can evaluate 7–5.

Pengkalkulasian Infix oleh Manusia
(lanjutan 2)

Infix: 2+4*5

Item Read	Expression Parsed So Far	Comments
3	3	
+	3+	
4	3+4	
*	3+4*	You can't evaluate 3+4 because * is higher precedence than +.
5	3+4*5	When you see the 5, you can evaluate 4*5.
	3+20	
End	23	When you see the end of the expression, you can evaluate 3+20.

Pengkalkulasian Infix oleh Manusia
(lanjutan 3)

Infix: 3*(4+5)

Item Read	Expression Parsed So Far	Comments
3	3	
*	3*	
(	3*(	
4	3*(4	You can't evaluate 3*4 because of the parenthesis.
+	3*(4+	
5	3*(4+5	You can't evaluate 4+5 yet.
)	3*(4+5)	When you see the), you can evaluate 4+5.
	3*9	After you've evaluated 4+5, you can evaluate 3*9.
	27	
End		Nothing left to evaluate.

Penerjemahan Infix ke Postfix

Infix: A+B-C

Character Read from Infix Expression	Infix Expression Parsed So Far	Postfix Expression Written So Far	Comments
A	A	A	
+	A+	A	
B	A+B	AB	
–	A+B–	AB+	When you see the –, you can copy the + to the postfix string.
C	A+B–C	AB+C	
End	A+B–C	AB+C–	When you reach the end of the expression, you can copy the –.

Penerjemahan Infix ke Postfix

(lanjutan 1)

Infix: A+B*C

Character Read from Infix Expression	Infix Expression Parsed So Far	Postfix Expression Written So Far	Comments
A	A	A	
+	A+	A	
B	A+B	AB	
*	A+B*	AB	You can't copy the + because * is higher precedence than +.
C	A+B*C	ABC	When you see the C, you can copy the *.
	A+B*C	ABC*	
End	A+B*C	ABC*+	When you see the end of the expression, you can copy the +.

Penerjemahan Infix ke Postfix

(lanjutan 2)

Infix: A*(B+C)

Character Read from Infix Expression	Infix Expression Parsed so Far	Postfix Expression Written So Far	Comments
A	A	A	
*	A*	A	
(	A*(	A	
B	A*(B	AB	You can't copy * because of the parenthesis.
+	A*(B+	AB	
C	A*(B+C	ABC	You can't copy the + yet.
)	A*(B+C)	ABC+	When you see the), you can copy the +.
	A*(B+C)	ABC+*	After you've copied the +, you can copy the *.
End	A*(B+C)	ABC+*	Nothing left to copy.

Penerjemahan Infix ke Postfix

(lanjutan 3)

Infix: A+B*(C-D)

Character Read from Infix Expression	Infix Expression Parsed So Far	Postfix Expression Written So Far	Stack Contents
A	A	A	
+	A+	A	+
B	A+B	AB	+
*	A+B*	AB	+*
(	A+B*(	AB	+*(
C	A+B*(C	ABC	+*(
–	A+B*(C–	ABC	+*(–
D	A+B*(C–D	ABCD	+*(–
)	A+B*(C–D)	ABCD–	+*(
	A+B*(C–D)	ABCD–	+*(
	A+B*(C–D)	ABCD–	+*
	A+B*(C–D)	ABCD–*	+
	A+B*(C–D)	ABCD–*+	

Penerjemahan Infix ke Postfix

(lanjutan 4)

Infix: A+B-C

Character Read from Infix	Infix Parsed So Far	Postfix Written So Far	Stack Contents	Rule
A	A	A		Write operand to output.
+	A+	A	+	If stack empty, push opThis.
B	A+B	AB	+	Write operand to output.
–	A+B–	AB		Stack not empty, so pop item.
	A+B–	AB+		opThis is –, opTop is +, opTop>=opThis, so output opTop.
	A+B–	AB+	–	Then push opThis.
C	A+B–C	AB+C	–	Write operand to output.
End	A+B–C	AB+C–		Pop leftover item, output it.

Penerjemahan Infix ke Postfix

(lanjutan 5)

Infix: A+B*C

Character Read From Infix	Infix Parsed So Far	Postfix Written So Far	Stack Contents	Rule
A	A	A		Write operand to postfix.
+	A+	A	+	If stack empty, push opThis.
B	A+B	AB	+	Write operand to output.
*	A+B*	AB	+	Stack not empty, so pop opTop.
	A+B*	AB	+	opThis is *, opTop is +, opTop<opThis, so push opTop.
	A+B*	AB	+*	Then push opThis.
C	A+B*C	ABC	+*	Write operand to output.
End	A+B*C	ABC*	+	Pop leftover item, output it.
	A+B*C	ABC*+		Pop leftover item, output it.

Penerjemahan Infix ke Postfix

(lanjutan 6)

A*(B+C)

Character Read From Infix	Infix Parsed So Far	Postfix Written So Far	Stack Contents	Rule
A	A	A		Write operand to postfix.
*	A*	A	*	If stack empty, push `opThis`.
(	A*(	A	*(	Push (on stack.
B	A*(B	AB	*(	Write operand to postfix.
+	A*(B+	AB	*	Stack not empty, so pop item.
	A*(B+	AB	*(	It's (, so push it.
	A*(B+	AB	*(+	Then push `opThis`.
C	A*(B+C	ABC	*(+	Write operand to postfix.
)	A*(B+C)	ABC+	*(	Pop item, write to output.
	A*(B+C)	ABC+	*	Quit popping if (.
End	A*(B+C)	ABC+*		Pop leftover item, output it.