MODUL
STRUKTUR DATA

Agung Setiawan, S.Kom, MM, M.Kom

SPIRIT TOWAR|D THE FUTURE

2021

KATA PENGANTAR

Assalamu'alaikum wr. Wb.

Puji dan syukur kami panjatkan kehadirat Tuhan Yang maha Esa, karena atas kehendak dan izin-Nya jualah Diktat Struktur Data. Praktikum Struktur Data pada perkuliahan ini menggunakan aplikasi Java - Netbeans. Dengan menggunakan diktat Struktur Data ini diharapkan mahasiswa/i bisa terbantu dalam memahami materi dan latihan yang diberikan oleh dosen, sehingga apa yang kita harapkan bisa membuat aplikasi dengan menggunakan struktur data yang baik dan benar.

Akhir kata penulis mengucapkan banyak terima kasih kepada semua rekan-rekan yang telah membantu dalam pembuatan diktat Struktur Data ini. Dan demi kesempurnaan dalam penyusunan diktat berikutnya, kami sangat mengharapkan kritik dan saran yang bersifat membangun dari rekan-rekan demi terwujudnya Tujuan Pendidikan Nasional "Mencerdaskan bangsa".

Rokan Hulu, 20 Februari 2017
Wassalam

Agung Setiawan

DAFTAR ISI

BAB I
DATA DAN ALGORITMA

Data adalah sekumpulan character yang berupa huruf atau angka yang dapat berdiri sendiri atau dapat pula membentuk satu kesatuan yang utuh dan belum memiliki arti secara definitif.

Logaritma adalah himpunan langkah-langkah instruksi untuk melaksana-kan suatu pekerjaan tertentu, dengan beberapa kriteria :

1. Ada input
2. Ada Output
3. Jelas dan tidak meragukan (definiteness)
4. Ada terminasi (Finiteness)
5. Efektif dan dapat dilaksanakan

> **Suatu *Struktur Data adalah suatu koleksi atau kelompok data yang dapat dikarakteristikan oleh organisasi serta operasi yang didefinisikan terhadapnya.***

Tujuan Mempelajari Struktur Data

1. Kompleksitas domain masalah yang perlu ditangani. :
   - Kebutuhan yang saling kontradiktif dan bertolak belakang
   - Terjadi perbedaan pandangan antara pemakai dan pengembang sistem
   - Kebutuhan sistem software sering perubahan (bisa selama masa pengembangan) mengalami
2. Kesulitan dalam pengelolaan proses pengembangan
   - Jumlah pengembangan yang banyak, semakin kompleks dan koordinasi yang semakin sulit. Penambahan tenaga ditengah penbgembangan proyek, justru menghambat proses penyelesaian proyek.
3. Fleksibilitas software
   - Pengembang harus mengembangkan blok-blok promitif dasar.
4. Masalah karakteristik kelakuan sistem diskrit.
   - Sistem diskrit semua kejadian eksternal dpt mempengaruhi suatu bagian dari state internal sistem.
   - Belum memiliki tool matematika atau kapasitas intelektual untuk memodelkan kelakuan sistem diskrit secara lengkap.

Salah satu metode yang digunakan dalam struktur data ini di Divide and Conquer (Pecah dan taklukan), Pembuatan program akan menjadi lebih sederhana jika masalah dapat dipecah menjadi submasalah-submasalah yang dapat dikelola. Metode yang digunakan :

- Top - Down
- Bottom - Up

Tipe data digunakan untuk mendeskripsikan suatu jenis data tertentu ini melibatkan representasi dan sekumpulan operasi yang sah untuk mengakses dan melibatkan data itu. Tipe data menjadi komoditas yang diketahui secara jelas serta melindungi dari penggunaan yang tidak seharusnya.

Operasi aritmatika spt. penambahan, pengurangan, perkalian dan

pembagian tidak diperkenankan pada tipe data karakter.

Pada garis besarnya, Data dapat dikategorikan menjadi :

A. Tipe Data Sederhana atau Data Sederhana, terdiri atas :
   - Data Sederhana Tunggal, misalnya Integer, Real, Boolean dan Character.
   - Data Sederhana Majemuk, misalnya String

B. Struktur Data, meliputi :
   - Struktur Data Sederhana, misalnya Array dan Record
   - Struktur Data Majemuk, terdiri atas :
     - Linier, misalnya : Stack, Queue dan Linear Linked List.
     - Nonlinier, misalnya : Pohon Biner (Binary Tree), Pohon Cari Biner (Binary Search Tree), Pohon Cari M-Way (M-Way Search Tree), Trie, General Tree serta Graph.

Pemakaian Struktur Data yang tepat didalam proses pemrogramman, akan menghasilkan Algoritma yang lebih jelas dan tepat sehingga menjadikan program secara keseluruhan lebih sederhana.

C. Tipe Data Sederhana

1. Tipe Data Integer

   Yang dimaksud dengan Integer adalah Bilangan Bulat
   Misal ...-3,-2,-1,0,1,2,3,....
   Tipe data Integer tidak mengandung pecahan, dalam memory komputer sebagai Bilangan Bulat.
   Rumusnya adalah $N <= 2^{n-1}-1$

   Tipe data Integer terbagi atas beberapa macam

Type	Range	Ukuran Byte	Format
ShortInt	-128 .. 127	1	8 bit
Integer	-32768 .. 32767	2	16 bit
LongInt	-2147483648 .. 2147483647	4	32 bit
Byte	0 .. 255	1	8 bit
Word	0 .. 65535	2	16 bit

OPERASI DALAM INTEGER

Nama Operasi	Operator	Level
Pemangkatan	SQRT	I
Pembagian (Hasil Bagi)	DIV	II
Pembagian (Sisa Hasil Bagi)	MOD	II
Perkalian	*	II
Penjumlahan	+	III
Pengurangan	-	III

2. Pemetaan (Mapping) ke Storage Integer

   Struktur data secara logik dapat mempunyai beberapa Storage Mapping, salah satu yang dapat dilakukan Mapping dalam Integer adalah bentuk *Sign – and – Magnitude*, dimana Integer Positif di definisikan dengan tanda Plus serta sebarisan digit yang menyatakan Magnitude atau Besaran. Juga untuk Integer Negatif, Bila sehari-hari Bilangan dinyatakan dalam Basis 10 (Desimal), tetapi untuk memori komputer dinyatakan dalam Basis 2 (Sistem Binary), Demikian untuk Bentuk Operand kita menggunakan *"Complement",* misalnya diketahui 3 bilang Integer non negatif : X, X' dan R, kita definisikan X' adalah Complement dari X terhadap R. Maka X disebut bentuk "True" dan X' disebut bentuk Complement dan dalam Binary R dipilih menjadi Pangkat dari 2. sehingga : $R = 2^N$

   Mapping dengan $R = 2^N$ dikenal dengan sistem *"Two's Complement (* menggunakan Nilai N = 4 dari bilangan Integer –7 s/d +7) , contoh :

Integer	Binary Sign and Magnitude	Two's Compl.
-7	-111	1001
-6	-110	1010
-5	-101	1011
-4	-100	1100
-3	-011	1101
-2	-010	1110
-1	-001	1111
+0	+000	0000
+1	+001	0001
+2	+010	0010
+3	+011	0011
+4	+100	0100
+5	+101	0101
+6	+110	0110
+7	+111	1111

3. Tipe Data Real

   Data numerik yang bukan termasuk Integer seperti Bilangan Pecahan dan Bilangan Takrasional, digolongkan dalam jenis Data Real. Jenis Data Real ditulis menggunakan titik (atau koma) desimal.

   Misalnya : 0.32 43,00 -131.128 dan sebagainya.

   Rumusnya adalah : $\mathbf{X = M * R^n}$

   Dimana M dijadikan Pecahan, R adalah Radixnya dan E merupakan Exponennya. Tipe data Real terbagi atas beberapa macam :

Type	Range	Ukuran Byte	Format
Real	$\pm\ 2.9\text{x}10^{-38}$ .. $1.7\text{x}10^{38}$	6	11-12 bit
Single	$\pm\ 1.5\text{x}10^{-45}$ .. $3.4\text{x}10^{38}$	4	7-8 bit
Double	$\pm\ 5\text{x}10^{-324}$ .. $1.7\text{x}10^{308}$	8	15-16 bit
Extended	$\pm\ 3.4\text{x}10^{-4932}$ .. $1.1\text{x}10^{4932}$	10	19-20 bit
Comp	$\pm\ 9.2\text{x}10^{18}$ .. $9.2\text{x}10^{18}$	8	19-20 bit

Operasi dalam real :

Nama Operasi	Operator	Level
Pemangkatan	SQRT	I
Perkalian	*	II
Pembagian	/	II
Penjumlahan	+	III
Pengurangan	-	III

4. Tipe Data Boolean atau Logikal

   Suatu Tipe data Boolean atau Logical hanya mempunyai dua bentuk keluaran yaitu nilai True dan False (Benar dan Salah) yang kerap kali dinyatakan sebagai 1 dan 0. Oleh karena itu satuan data yang terpakai cukup berisi satu bit saja. Operator yang digunakan adalah And, Or, Not, Nand, Nor, Xor. Nilai True dan False dapat juga dihasilkan dari apa yang dikenal sebagai Operator Relational atau Relational Operator (Relop).
   Operator Relational tidak mempunyai operand boolean, tetapi menghasilkan type boolean. Operator Relational tersebut adalah **=, <, >, <=, >=, <>.** Tipe data Boolean terbagi atas beberapa macam, yaitu :

Type	Range	Ukuran Byte	Format
Boolean	Byte Size	1	8 bit
Byte Bool	Byte Size	1	8 bit
Word Bool	Word Size	2	16 bit
Long Boot	Longint Size	4	32 bit

5. Tipe Data Character dan String

   Tipe Data Charater adalah type data yang elemennya terdiri dari aksara (simbol) seperti : 0,1,2,3,4,5,6,7,8,9,A,B,C,D,....,X,Y,Z,a,b,c,d,...., x,y,z,/,?,*,-,_,....) meliputi *digit numerik*, *character alfabetik* dan *spesial character* atau *String* saja. Type Data String adalah type data majemuk yang terbentuk dari kumpulan (himpunan hingga) character. Himpunan hingga character yang digunakan untuk membentuk String dinamakan *alfabet.*

6. Pemetaan ke Storage Character dan String

   Aturan Storage Mapping dalam Karakter, menggunakan Kode "EBCDIC" *Extended Binary Coded Decimal Interchange Code* dan *"ASCII" American Standar Code for Information Interchange.* EBCDID, dikembangkan oleh IBM dengan kode 8 bit (8 binary digit) yang terdapat 2^8 (= 256).

OPERASI STRING

a. LEN(String)

Operasi untuk menghitung Panjang String. Hasil dari operasi Length adalah suatu type Integer (bil.bulat).

Misal : S = ‘AGUNG SH’
Length(S) = 8

b. CONCAT(S1,S2)

Operasi untuk menyambung atau menggabungkan 2 buah String. Hasil dari operasi Concatenation adalah String.

Misal : S1 = a1,a2,...,aN
S2 = b1,b2,...,bK
CONCAT(S1,S2) = a1,a 2,.,a N ,b1 b2,.,bK

c. SUBSTR(S,i,j)

Operasi untuk membuat suatu String baru dengan cara mengambil beberapa Character berurutan dari Suatu String yang diketahui. Dimana :

S adalah String semula
i adalah posisi Character awal yang diambil
j adalah banyaknya Character yang diambil
I dan j berTipe data Integer

Misal : S = ‘AGUNG SETIAWAN(S,8,4) = ‘TIAW’
Z = y_1,y_2,y_3,y_4,y_5
SUBSTR(Z,2,2) = y_2,y_3

d. INSERT(S_1,S_2,J)

Operasi untuk menyisipkan String S_2 didalam String S_1, sedemikian sehingga Character pertama dari S_2 menggantikan posisi Character ke J dari S_1, selesai dengan seluruh String S_2 dilanjutkan lagi dengan sisa String S_1.Untuk operasi INSERT dibutuhkan dua operand String dan sebuah operand Integer dan Hasilnya adalah String.

Misal : S1 = a_1,a_2,a_3,a_4,a_5
S2 = b_1,b_2,b_3,b_4,b_5
INSERT(S1,S2,3) = $a_1,a_2,b_1,b_2,b_3,b_4,b_5,a_3,a_4,a_5$
S3 = AGUNG
S4 = SETIAWAN
INSERT(S3,S4,5) = AGUNGSETIAWAN

e. DELETE (S,I,J)

Operasi untuk menghapus suatu String yang panjangnya J, bermula pada posisi ke I dari String S. Operator DELETE mengandung sebuah operand String dan 2 operand Integer.

Misal : S = U_1,U_2,U_3,U_4,U_5
DELETE(S,2,3) = U_4,U_5

f. LEFT(S,I,J)

Operasi untuk mengambil suatu *String dari kiri* yang panjangnya J,

bermula pada posisi ke I dari String S. Operator LEFT mengandung sebuah operand String dan 2 operand Integer..

Misal : $S = U_1,U_2,U_3,U_4,U_5$

$LEFT(S,2,3) = U_2,U_3,U_4$

g. RIGHT(S,I,J)

Operasi untuk mengambil suatu *String dari kanan* yang panjangnya J, bermula pada posisi ke I dari String S. Operator RIGHT mengandung sebuah operand String dan 2 operand Integer.

Misal : $S = U_1,U_2,U_3,U_4,U_5$

$RIGHT(S,2,2) = U_3,U_4$

Dalam pembuatan tipe dengan object oriented program, pembuatan tipe data menggunakan metode User Defined Type (UDT). Penggunaan UDT mempunyai kemampuan sebagai berikut :

- Berorientasi pada cara dimana data disimpan dimesin komputer.
- Programmer dapat mengekspresikan tipe data yang didefinisikan sendiri sesuai dengan domain persoalan yang dihadapi.
- Dengan kemampuan pembuatan tipe data oleh programmer, memungkinkan tipe data yang mendekati abstraksi tipe-tipe dunia eksternal (persoalan)
- Contoh (bhs java):

```
Import java.util.*
Class person
  string nama;
  string alamat;
  string kota;
End;
Class app { aperson:person; …}
```

- UDT dpt meningkatkan daya ekspresi dari bahasa pemrograman

Penggunaan UDT dimaksudkan untuk mengelola data yang bersifat konkret, sedang data yang bersifat abstrak menggunakan metode Abstract Data Type (ADT).

Konsep ini perluasan konsep UDT dengan penambahan encapsulation/ pembungkus (Pembungkusan merupakan suatu karakteristik OOP yang dimana program terbungkus menjadi satu data (property) dan perilaku (method), artinya memperhatikan aspek eksternal daripada aspek internal)

- Berisi representasi dari operasi-operasi terhadap tipe data.
- ADT memberikan antarmuka publik hanya lewat operasi yang dibolehkan.
- Implementasinya disebut class atau module atau package/
- Programmer dapat mendefinisikan variabel Date, Screen, CustomerOrder, Part, Company dsb. Mendekati tipe-tipe dunia nyata.

Penggunaan data dengan ADT, sangat baik digunakan untuk:

- Blok bentukan terpenting pd pendekatan berorientasi objek.
- Kelas adl ADT yang dilengkapi dengan imlepementasi parsial atau total.

Menurut Bertrand Meyer :

Class adalah tipe data abstrak (abstract data type) yang implementasinya telah direalisasikan atau data ditunda.

CONTOH PROGRAM 1 :

```
/**
 *
 * @author Agung Setiawan
 */
class TipeData {
    public static void main(String[] args) {
              // Tipe data primitif
    long    data1 = 226531;
    int     data2 = 6641;
    short   data3 = 714;
    byte    data4 = 34;
    float   data6 = (float) 1.733;            // tipe data pecahan
    double  data5 = 4.967;                    // tipe data pecahan
    char    data7 = 'C';
    boolean data8 = true;

    System.out.println("Nilai Long    : "+ data1);
    System.out.println("Nilai Int     : "+ data2);
    System.out.println("Nilai Short   : "+ data3);
    System.out.println("Nilai Byte    : "+ data4);
    System.out.println("Nilai Double  : "+ data5);
    System.out.println("Nilai Float   : "+ data6);
    System.out.println("Nilai Char    : "+ data7);
    System.out.println("Nilai Boolean : "+ data8);


   }
 }
```

CONTOH PROGRAM 2 :

```
/**
 *
 * @author Agung Setiawan
 */
public class Hello {
  public static void main (String args[]) {
     System.out.println ("Halloo, Coba Java.");
}
}
```

CONTOH PROGRAM 3 :

```
/**
 *
 * @author Agung Setiawan
```

```
 */
public class Latihan {

public static void main(String args[]) {
int var1; // this declares a variable
int var2; // this declares another variable
var1 = 1024; // this assigns 1024 to var1
System.out.println("var1 Berisi " + var1);
var2 = var1 / 2;
System.out.print("var2 Berisi var1 / 2: ");
System.out.println(var2);
   }
}
```

CONTOH PROGRAM 4 :

```
/**
 *
 * @author Agung Setiawan
 */
public class alamat {
   String name;
   String addressLine1;
   String addressLine2;
   String Hp;
   int age;

   /** Creates a new instance of PersonToy */
   public alamat() {
         name = "";
         addressLine1 = "";
         addressLine2 = "";
         Hp = "";
         age = 0;
   }

   public alamat(String newName, String newAddressLine1, String
newAddressLine2, String newHp, int newAge) {
         name = newName;
         addressLine1 = newAddressLine1;
         addressLine2 = newAddressLine2;
         Hp = newHp;
         age = newAge;
   }

   public void setName (String newName){
         name = newName;
   }
```

```
public void setAddressLine1 (String newAddressLine1){
        addressLine1 = newAddressLine1;
}

public void setAddressLine2 (String newAddressLine2) {
        addressLine2 = newAddressLine2;
}

public void setHp (String newHp) {
        Hp = newHp;
}

public void setAge (int newAge) {
        age = newAge;
}

public String getName (){
        return name;
}

public String getAddressLine1 (){
        return addressLine1;
}

public String getAddressLine2 (){
        return addressLine2;
}

public String getHp (){
        return Hp;
}

public int getAge (){
        return age;
}

public String toString(){
    String str =
        "Nama      : "+ name + "\n"+
        "Alamat 1  : "+ addressLine1 + "\n"+
        "Alamat 2  : "+ addressLine2 + "\n"+
        "No HP.    : "+Hp + "\n"+
        "Umur      : "+age + "\n";
    return str;
}

static void test(){
```

```
        alamat t = new alamat("Agung Setiawan, S.Kom, MM, M.kom","Perumahan PTPN 5 Kandir - Pantai Raja","Riau","081383461440/083187447807/087790928909",40);

        System.out.println("Agung Setiawan S.Kom, MM, M.kom sebagai Person:");
        System.out.println(t.getName());
        System.out.println(t.getAddressLine1());
        System.out.println(t.getAddressLine2());
        System.out.println(t.getHp());
        System.out.println(t.getAge());
        System.out.println(t);
    }

    public static void main(String[] args) {
        test();
    }
}
```

BAB II
ABSTRACTION DATA TYPE (ADT)

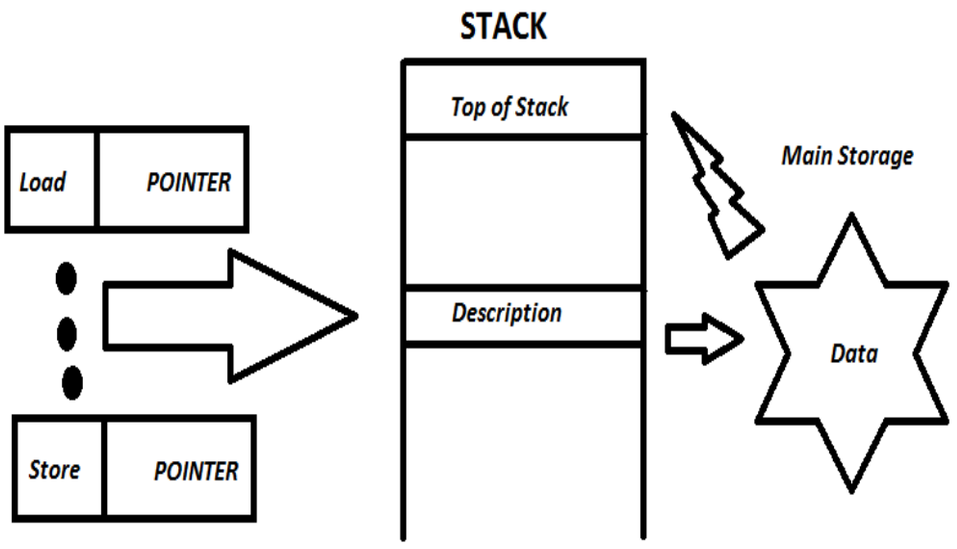
Untuk memisahkan struktur penyimpanan (lokasi memori) dari kelakuan struktur data yang abstrak spt : stack dan Queue. → information hiding / encapsulation.

Stack : Lokasi memori yg dicadangkan untuk penyimpanan data yg bersifat temporer. → Dpt berbentuk list.

Queue : Daftar tunggu atau garis (line) yg dibentuk oleh item dalam sistem tunggu (waiting system) utk pelayanan, misalnya pesan yang ditransmisikan atau tugas yg dilaksanakan. Dalam menjalankan ADT, digunakan beberapa bentuk, antara lain :

a. Bentuk Formal
   ADT ({Nilai-nilai yang diizinkan}, (Operasi-operasi yang diizinkan))
b. Bentuk Tuple
   ADT (ModelmatematiKa, {Operasi-operasi terhadap Modelmatematika})
   Himpunan bilangan Integer ditambah operasi-operasi gabungan (union), irisan (intersection), pengurangan himpunan (set difference) akan membentuk **ADT HimpunanInteger**
   ADT HimpunanInteger = ({Integer}, {union, intersection, difference, operasi logika lainnya})

STACK



c. Encapsulation
   Pembungkusan merupakan suatu karakteristik OOP yang dimana program terbungkus menjadi satu data (property) dan perilaku (method), artinya memperhatikan aspek eksternal daripada aspek internal

d. Tujuan ADT Secara encapsulation
   1. Perubahan implementasi tipe data abstrak tdk mengubah teks program yg menggunakan tipe data abstrak bila interface dirancang secara

bagus.

2. Pemakaian dan pembuatan tipe data abstrak dapat dilakukan secara terpisah, yg perlu disepakati adl interface pemakaian tipe data.
3. Tipe data abstrak merupakan sarana pemrograman modular dan menjadi landasan terbentuknya tim pemrograman.

CONTOH PROGRAM 1 :

```
/**
 *
 * @author Agung Setiawan
 */
public class suhuudara {
  private double valueInCelcius;
  public suhuudara() {
   }
  public suhuudara (double newValueInCelcius) {
        valueInCelcius = newValueInCelcius;
   }
  public void setInCelcius (double newValue) {
        valueInCelcius = newValue;
   }
  public void setInReamur (double newValue) {
        valueInCelcius = (5/4) * newValue;
   }
  public void setInFahrenheit (double newValue) {
      valueInCelcius = (5/9) * (newValue - 32);
   }

   public void setInKelvin (double newValue) {
        valueInCelcius = newValue - 272;
   }

   public double getInCelcius () {
        return valueInCelcius;
   }

   public double getInReamur () {
        return ((4/5) * valueInCelcius);
   }

   public double getInFahrenheit () {
        return (((9/5) * valueInCelcius) + 32);
   }

   public double getInKelvin () {
        return (valueInCelcius + 272);
   }
```

```
   static void test(){
      suhuudara t = new suhuudara(0.0);

         System.out.println("Pengujian");
         System.out.println("Aplikasi Thermometer");
         System.out.println("Nilai Celcius Yang Dimasukkan adalah 50");


         System.out.println("Nilai di Celcius    : "+t.getInCelcius());
         System.out.println("Nilai di Reamur     : "+t.getInReamur());
         System.out.println("Nilai di Fahrenheit : "+t.getInFahrenheit());
         System.out.println("Nilai di Kelvin     : "+t.getInKelvin());

         t.setInCelcius(50.0);

         System.out.println("Nilai di Celcius    : "+t.getInCelcius());
         System.out.println("Nilai di Reamur     : "+t.getInReamur());
         System.out.println("Nilai di Fahrenheit : "+t.getInFahrenheit());
         System.out.println("Nilai di Kelvin     : "+t.getInKelvin());

         t.setInCelcius(100.0);

         System.out.println("Nilai di Celcius    : "+t.getInCelcius());
         System.out.println("Nilai di Reamur     : "+t.getInReamur());
         System.out.println("Nilai di Fahrenheit : "+t.getInFahrenheit());
         System.out.println("Nilai di Kelvin     : "+t.getInKelvin());
   }

   static public void main (String[] args){
      test();
   }
}
```

CONTOH PROGRAM 2 :

```
package latihan;

/**
 *
 * @author Agung Setiawan
 */
abstract class Bentuk {
   protected int panjang;
   protected int lebar;
   public String getBentuk() {
      return "Bentuk Dasar";
   }
```

```
    public abstract int hitungLuas();
}
class BujurSangkar extends Bentuk {
    public BujurSangkar(int panjang1, int lebar1) {
        this.panjang = panjang1;
        this.lebar  = lebar1;
    }
    public String getBentuk() {
        return "Bentuk Bujur Sangkar";
    }
    public int hitungLuas() {
        return panjang*lebar;
    }
}
class SegiTiga extends Bentuk {
    public SegiTiga(int panjang2, int lebar2) {
        this.panjang = panjang2;
        this.lebar  = lebar2;
    }
    public int hitungLuas() {
        return this.panjang*this.lebar/2;
    }
}
class Polimorfisme {
    public static void cetakLuasBentuk(Bentuk btk) {
    System.out.println(btk.getBentuk() + " dengan luas "
        + btk.hitungLuas());
    }
    public static void main(String[] args) {
        BujurSangkar bs  = new BujurSangkar(10,20);
        BujurSangkar bs1 = new BujurSangkar(10,20);
        SegiTiga st  = new SegiTiga(5,10);
        SegiTiga st1 = new SegiTiga(50,100);

        cetakLuasBentuk(bs);
        cetakLuasBentuk(bs1);
        cetakLuasBentuk(st);
        cetakLuasBentuk(st1);
    }
}
```

BAB III
ARRAY

Salah satu Struktur Data yang teramat penting adalah Array atau Larik. Array dapat didefinisikan sebagai suatu himpunan hingga elemen, terurut dan homogen. Atau Dapat didefinisikan juga sebagai pemesanan alokasi memory sementara pada komputer.

Terurut : Dapat diartikan bahwa elemen tersebut dapat diidentifikasi sebagai elemen pertama, elemen kedua dan seterusnya sampai elemen ke-n.

Homogen : Adalah bahwa setiap elemen dari sebuah Array tertentu haruslah mempunyai type data yang sama. Sebuah Array dapat mempunyai elemen yang semuanya berupa integer atau dapat pula seluruhnya berupa character atau String bahkan dapat pula terjadi suatu Array mempunyai elemen berupa Array pula.

Karakteristik Array :

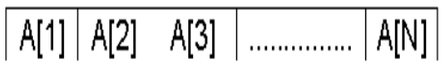
1. Mempunyai batasan dari pemesanan alokasi memory (Bersifat Statis) Mempunyai Type Data Sama (Bersifat Homogen)
2. Dapat Diakses Secara Acak

Penggunaan array antara lain :

a. Penggunaan tanpa array, hanya berisi variabel data yang dapat menampung satu buah nilai dan nilai disimpan oleh variabel data adalah nilai yang terakhir dimasukan.
b. Penggunaan array untuk menyimpan lebih dari satu nilai data sekaligus dalam sebuah variabel digunakan tipe data terstrukur.
c. Untuk dapat menyimpan data lebih dari satu nilai sekaligus, bahasa pemroghraman menyediakan array.
d. Array digunakan untuk menyimpan data yang tipenya sama
e. Setiap data dikenal melalui indeksnya.
f. Berdasarkan jumlah indeks dalam sebuah variabel, array dibedkan menjadi array berdimensi satu dan array berdimensi banyak.

Adapaun jenis Array yang akan dipelajari adalah :

1. Dimensi Satu (One Dimensional)
   Deklarasi : Var A : Array[1..N] Of Type Data;
   Penggambaran secara Logika :



   Harga minimum dari subskrip suatu Array disebut sebagai batas bawah atau lower bound (L), sedangkan harga maksimumnya disebut batas atas atau upper bound (U).
2. Pemetaan Array Dimensi Satu ke Storage
   Mapping Storage Array Dimensi Satu
   Rumus : @A[I] = B + {U - 1} * L
   Dimana @A[I] : Posisi Array Yg Dicari

B : Posisi Awal Array di Memory Komp.
U : Subkrip atau Indeks Atas Array
L : Besar Memory dari St. Type Data

Rumus menentukan jumlah elemen dalam Array :
Jumlah Elemen :

$$\prod_{i=1}^{n} (Ui - Li)$$

π = Pi (Perkalian dari statemen sebelumnya)

3. Array Dimensi Dua (Two Dimensional)
   B.U : Var A : Array[1..N,1..M] Of Type Data;

   Sering digunakan dalam menterjemahkan matriks pada pemrograman.
   Mis : Var A : Array[1..3,1..4] Of Integer;

   Terbagi Dua cara pandang (representasi) yang berbeda :
   1. Secara Kolom Per Kolom (Coloumn Major Oder)
   @M[i,j] = @M[1,1] + {(j - 1) * K + (i - 1)} * L

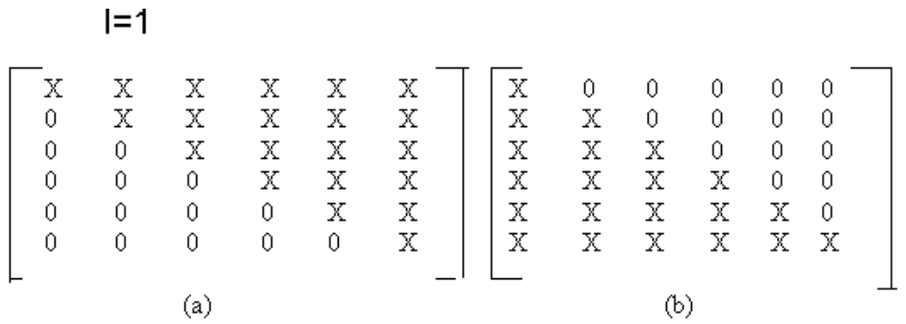
   Ket : I = Baris, j = kolom, L = panjang elemen
   K = Banyaknya kolom
   N = Banyaknya baris

4. Array Dimensi Tiga (Three Dimensional) dan Dimensi Banyak
   B.U : Var A : Array[1..N,1..M,1..P,1..R,…] Of Type Data;

   Pada dasarnya hanya pengembangan dari array dimensi tiga, yaitu :
   @M[n,m,p,r,….] = @M[1,1,1,1,….] + {(n-1)*(Ui-Li+1)+(m-1)*(Ui-Li+1)+
   (p-1)* (Ui - Li + 1)+……} * L

5. Array Segitiga (Tringular Array)
   Beberapa aspek pelinieran suatu Array yang harus ditinjau adalah Tringular Array. Tringular Array dapat merupakan Upper Tringular (seluruh elemen di bawah diagonal utama = 0), ataupun Lower Tringular (seluruh elemen di atas diagonal utama = 0).
   Dalam Array Lower Tringular dengan N baris, jumlah maksimum elemen <> 0 pada baris ke-I adalah = I, karenanya total elemen <>0, tidak lebih dari

$$\sum_{I=1}^{N} I = N(N+1)/2$$



Gambar (a) Upper Triangular Array
(b) Lower Triangular Array

Contoh berikut.

$$A = \begin{vmatrix} 1 & 2 & 3 \\ 0 & 4 & 5 \\ 0 & 0 & 6 \end{vmatrix} \quad B = \begin{vmatrix} 7 & 0 & 0 \\ 8 & 9 & 0 \\ 11 & 12 & 13 \end{vmatrix}$$

dapat disimpan sebagai Array C berorder (3 X 4)

$$\begin{vmatrix} 7 & 1 & 2 & 3 \\ 8 & 9 & 4 & 5 \\ 11 & 12 & 13 & 6 \end{vmatrix}$$

6. Array Jarang (Sparse Array)

Suatu Array yang sangat banyak elemen nol-nya, dikenal sebagai Sparse Array, contohnya adalah Array A pada Gambar berikut :

$$\begin{vmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 2 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{vmatrix}$$

LATIHAN ARRAY :

CONTOH PROGRAM 1 :

```
public class Array1 {
    public static void main(String[] args) {
int [] x;
    x = new int[5];
    x[0] = 0;
    x[1] = 1;
    x[2] = 2;
```

```
        x[3] = 3;
        x[4] = 4;
   System.out.println("Nilai x[0] : " + x[0]);
   System.out.println("Nilai x[1] : " + x[1]);
   System.out.println("Nilai x[2] : " + x[2]);
   System.out.println("Nilai x[3] : " + x[3]);
   System.out.println("Nilai x[4] : " + x[4]);
}}
```

CONTOH PROGRAM 2 :

```
public class Array2 {
    public static void main(String[] args) {
   int [] x;                // Cara 1
   x = new int[3];
   x[0] = 20 ; x[1] = 10 ; x[2] = 30;
   System.out.println("Nilai x[0] : " + x[0]);
   System.out.println("Nilai x[1] : " + x[1]);
   System.out.println("Nilai x[2] : " + x[2]);
   int [] y = new int[3];      // Cara 2
   y[0] = 20 ; y[1] = 10 ; y[2] = 30;
   System.out.println("Nilai y[0] : " + y[0]);
   System.out.println("Nilai y[1] : " + y[1]);
   System.out.println("Nilai y[2] : " + y[2]);
    }}
```

CONTOH PROGRAM 3 :

```
public class Array3 {
    public static void main(String[] args) {
   int [] x;                // Cara 1
   x = new int[3];
   x[0] = 20 ;    x[1] = 10 ;    x[2] = 30;
   System.out.println("Nilai x[0] : " + x[0]);
   System.out.println("Nilai x[1] : " + x[1]);
   System.out.println("Nilai x[2] : " + x[2]);
   int [] y = new int[3];      // Cara 2
   y[0] = 20 ;    y[1] = 10 ;    y[2] = 30;
   System.out.println("Nilai y[0] : " + y[0]);
   System.out.println("Nilai y[1] : " + y[1]);
   System.out.println("Nilai y[2] : " + y[2]);
   int[] z = {20,10,30};          // Cara 3 tdk menggunakan new
   System.out.println("Nilai z[0] : " + z[0]);
   System.out.println("Nilai z[1] : " + z[1]);
   System.out.println("Nilai z[2] : " + z[2]);
}}
```

CONTOH PROGRAM 4 :

```
public class Array4 {
      public static void main(String[] args) {
   int [] x;                // Cara 1
   x = new int[3];
   x[0] = 20 ;
   x[1] = 10 ;
   x[2] = 30;
   int [] y = new int[3];      // Cara 2
   y[0] = 20 ;
   y[1] = 10 ;
   y[2] = 30;
   int[] z = {20,10,30};          // Cara 3 tdk menggunakan new
   System.out.println("Nilai Array x, y dan z : " + x[0] + "  " + y[0] + "  " + z[0]);
   System.out.println("Nilai Array x, y dan z : " + x[1] + "  " + y[1] + "  " + z[1]);
   System.out.println("Nilai Array x, y dan z : " + x[2] + "  " + y[2] + "  " + z[2]);
   }}
```

CONTOH PROGRAM 5 :

```
public class ArrayD {
       public static void main(String[] args) {

   int[][][] arr3 = {{{10,20,30},{40,50,60}},
                     {{11,21,31},{41,51,61}},
                     {{12,22,32},{42,52,62}}};      // ukuran 3 * 6 = 18

   System.out.println("Nilai arr3[0] : " + arr3[0][0][0]);
   System.out.println("Nilai arr3[0] : " + arr3[0][0][1]);
   System.out.println("Nilai arr3[0] : " + arr3[0][0][2]);
   System.out.println("Nilai arr3[0] : " + arr3[0][1][0]);
   System.out.println("Nilai arr3[0] : " + arr3[0][1][1]);
   System.out.println("Nilai arr3[0] : " + arr3[0][1][2]);

   System.out.println("Nilai arr3[1] : " + arr3[1][0][0]);
   System.out.println("Nilai arr3[1] : " + arr3[1][0][1]);
   System.out.println("Nilai arr3[1] : " + arr3[1][0][2]);
   System.out.println("Nilai arr3[1] : " + arr3[1][1][0]);
   System.out.println("Nilai arr3[1] : " + arr3[1][1][1]);
   System.out.println("Nilai arr3[1] : " + arr3[1][1][2]);

   System.out.println("Nilai arr3[2] : " + arr3[2][0][0]);
   System.out.println("Nilai arr3[2] : " + arr3[2][0][1]);
   System.out.println("Nilai arr3[2] : " + arr3[2][0][2]);
   System.out.println("Nilai arr3[2] : " + arr3[2][1][0]);
   System.out.println("Nilai arr3[2] : " + arr3[2][1][1]);
   System.out.println("Nilai arr3[2] : " + arr3[2][1][2]); }}
```

BAB IV
STRUKTUR DAN RECORD

Sebuah Record merupakan koleksi satuan data yang heterogen, yakni terdiri dari berbagai type. Satuan data tersebut sering disebut sebagai field dari record. Field dipanggil dengan menggunakan namanya masing-masing. Suatu field dapat terdiri atas beberapa subfield.

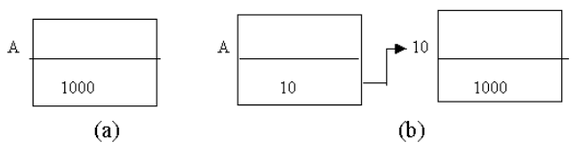
STRUKTUR Merupakan :
Kumpulan elemen-elemen data yang digangkan menjadi satu kesatuan
RECORD Merupakan :
Kumpulan item data yang saling berhubungan yang diperlakukan sebagai satu unit, misalnya transaksi yang berkaitan dengan pembeli.
LOGICAL RECORD Merupakan :
Suatu catatan yang berisikan semua item-item yang penting untuk menunjukkan beberapa fungsi khusus. Suatu catatan yang ditetapkan dalam istilah informasi yang dikandungnya bukannya catatan fisiknya. Bagian dari catatan logika akan berada dalam catatan fisik yang berbeda.

4.1. Konsep Pointer dan Linked List

Untuk mengolah data yang banyaknya tidak bisa ditentukan sebelumnya, maka disediakan satu fasilitas yang memungkinan untuk menggunakan suatu perubah yang disebut dengan perubah dinamis (Dinamic variable), yaitu ***suatu perubah yang akan dialokasikan hanya pada saat diperlukan, yaitu setelah program dieksekusi. Dengan kata lain, pada saat program dikompilasi, lokasi untuk perubah tersebut belum ditentukan***.

Pada perubah statis, isi Memory pada lokasi tertentu (nilai perubah) adalah data sesungguhnya yang akan diolah. Pada perubah dinamis, nilai perubah adalah alamat lokasi lain yang menyimpan data sesungguhnya. Dengan demikian data yang sesungguhnya dapat dimasukkan secara langsung.

Dalam hal cara pemasukkan data dapat diilustrasikan seperti Gambar (a) dan (b) dibawah ini.



(a) (b)

4.2. Deklarasi Pointer dan Alokasi Tempat

Dalam bahasa pemrogramman Pascal Type Data Pointer biasanya dideklarasikan pada bagian Deklarasi Type.

Bentuk Umum Deklarasi Pointer :

```
Type Pengenal = ^Simpul;
             Simpul   = Tipe;
```

Pengenal : Nama pengenal yang menyatakan data bertipe Pointer.
Simpul : Nama Simpul.
Tipe : Type Data dari Simpul.

Tanda ^ di depan nama Simpul harus ditulis seperti apa adanya dan menunjukkan bahwa pengenal adalah suatu Type Data Pointer.

Type Data Simpul yang dinyatakan dalam Tipe dapat berupa sembarang Type Data, misalnya Char, Integer dan Real.

Contoh :

```
Type      Str30  = String[30];   Point  = ^Data;
                   Data   = Record;
                      Nama_peg  :  Str30;
                      Alamat       :  Str30;
                      Pekerjaan    : Str30;
                   End;
```

4.3. Operasi Pointer

Dalam bahasa pemrogramman Pascal Type Data Pointer biasanya dideklarasikan pada bagian Deklarasi Type.

Pertama kali yang harus dilakukan adalah mendeklarasikan Type Pointernya, yaitu :

```
Type   Simpul  = ^Data;
       Data    =  Record;
                     Nama    : String;
                     Alamat  : String;
                     Berikut  : Simpul;
                End;
Var   T1,T2  : Simpul;
```

4.4. Menghapus Pointer

Pointer yang telah dialokasikan (dibentuk) dapat didealokasikan (dihapus) kembali pada saat program dieksekusi. Setelah suatu Pointer dihapus, maka lokasi yang semula ditempati oleh simpul yang ditunjuk oleh Pointer tersebut akan bebas, sehingga dapat digunakan oleh perubah lain.

Statemen untuk menghapus Pointer adalah **Dispose**, yang mempunyai bentuk umum :

Dispose (perubah)

Dengan perubah adalah sembarang perubah yang bertype Pointer. Sebagai contoh, dengan menggunakan deklarasi :

```
Type      Mendaftar      = (YA,TIDAK);
          Tanggal   = Record;
                         Bulan      : 1..12;
                         Tahun      : 0..99;
                      End;
          Siswa          = Record;
                         Nomor_mhs    : String[10];
                         Nama_mhs     : String[30];
                         Tgl_daftar       : Tanggal;

                    End;
     Daf_Siswa           : ^Siswa;
     Var  Murid, Murid1  : Daf_Siswa;
```

Kemudian dapat dibentuk simpul baru, yaitu :

```
     New (Murid);
     Murid1  :=  Murid;
```

Pada suatu saat, simpul yang ditunjuk oleh Pointer Murid1 tidak digunakan lagi, maka dapat dihapus dengan menggunakan statemen :

Dispose (Murid1);

Demikian penjelasan tentang perubah dinamis yang lebih dikenal denagn sebutan Pointer.

4.5. Linked List

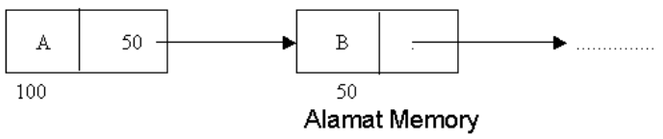
Salah satu Struktur Data Dinamis yang paling sederhana adalah Linked List atau Struktur Berkait atau Senarai Berantai, yaitu suatu kumpulan komponen yang disusun secara berurutan dengan bantuan Pointer.

Linked List (Senarai Breantai) disebut juga dengan Senarai Satu Arah (One-Way List). Masing-masing komponen dinamakan dengan Simpul (Node).

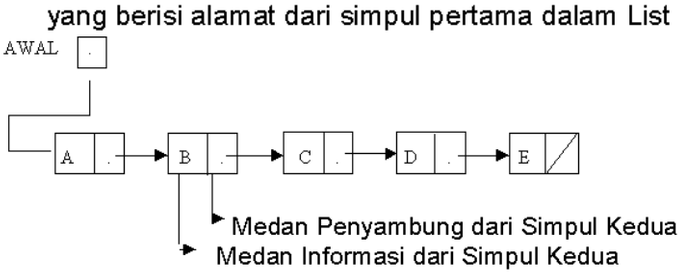
Dengan demikian, setiap simpul dalam suatu Linked List terbagi menjadi dua (2) bagian.

- Medan Informasi, berisi informasi yang akan disimpan dan diolah.
- Medan Penyambung (Link Field), berisi alamat berikutnya. Bernilai 0, Jika Link tersebut tidak menunjuk ke Data (Simpul) lainnya. Penunjuk ini disebut Penunjuk Nol.

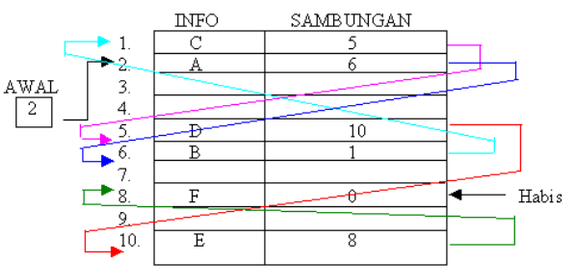
Selain jenis struktur terkait yang telah disebutkan diatas ada beberapa jenis Struktur Berkait yang lain seperti Linked-Stack, Linked-Queue, Doubly Linked-List, Linked Centralize-List dan sebagainya.



Linked List juga mengandung sebuah variabel Penunjuk List, yang biasanya diberi nama START(AWAL), yang berisi alamat dari simpul pertama dalam List



PENYAJIAN LINKED LIST DALAM MEMORY



Keterangan :

AWAL	=	2	, Maka	INFO[2]	=	'A'
SAMBUNGAN[2]	=	6	, Maka	INFO[6]	=	'B'
SAMBUNGAN[6]	=	1	, Maka	INFO[1]	=	'C'
SAMBUNGAN[1]	=	5	, Maka	INFO[5]	=	'D'
SAMBUNGAN[5]	=	10	, Maka	INFO[10]	=	'E'
SAMBUNGAN[10]	=	8	, Maka	INFO[8]	=	'F'
SAMBUNGAN[8]	=	0	, Maka	Akhir Linked List		

Dari contoh diatas diperoleh untai **'ABCDEF'**

OPERASI PADA LINKED LIST

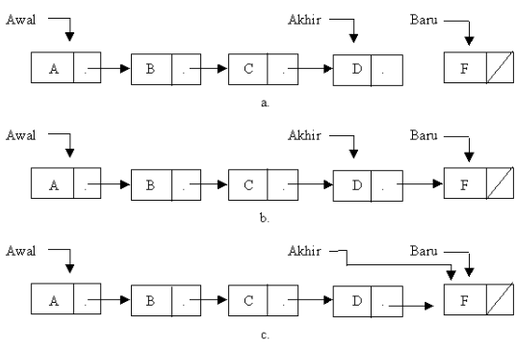
Menambah Simpul

Deklarasi :

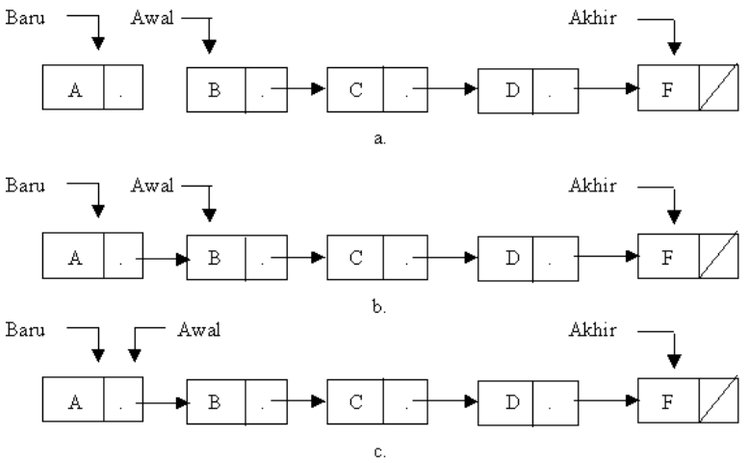
```
Type      Simpul = ^Data;
     Data    = Record
                 Info  : Char;
                 Berikut : Simpul;
               End;
Var  Elemen          : Char;
     Awal,Akhir,Baru      : Simpul;
```

```
Procedure Tambah_Belakang(Var Awal,Akhir : Simpul;
                          Elemen : Char);
    Var   Baru   : Simpul;
    Begin
          New(Baru);Baru^.Info := Elemen;
          If Awal = Nil Then     {* Linked List Masih Kosong}
                Awal := Baru;
          Else
                Akhir ^.Berikut := Baru;    {Gbr(b)}
          Akhir  := Baru;                   {Gbr(c)}
          Akhir^.Berikut  := Nil
    End;
```

A. Menambah Di Belakang

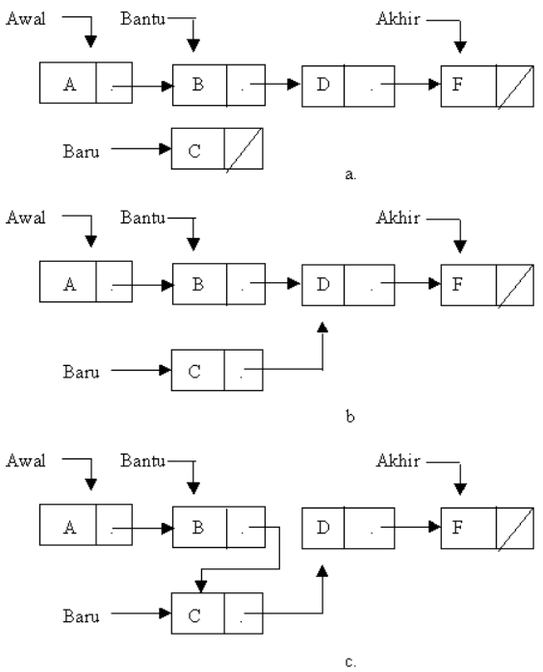


B. Menambah Di Depan



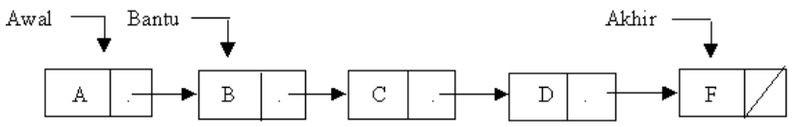

```
ProcedureTambah_Depan(VarAwal,Akhir:Simpul;Elemen : Char);
Var   Baru   : Simpul;
Begin
            New(Baru);Baru^.Info := Elemen;
            If Awal = Nil Then       {* Linked List Masih Kosong}
                  Akhir := Baru;
            Else
                  Baru ^.Berikut := awal;    {Gbr (b)}
                  Awal  := Baru;             {Gbr(c)}
End;
```

C. Menambah Di Tengah

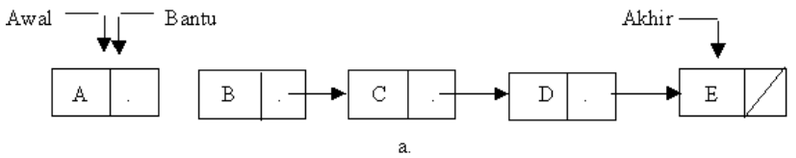


```
Procedure Tambah_tengah(Var Awal,Akhir : Simpul;Elemen : Char);
Var Baru,Bantu : Simpul;
Begin
      New(Baru);Baru^.Info := Elemen;
      If Awal = Nil Then
            Begin
                  Awal := Baru;
                  Akhir := Baru
            End
      Else
            Begin
                  {* Mencari lokasi yang sesuai *}
                  Bantu := Awal;
                  While Elemen > Baru^.Berikut   Do
                        Bantu := Bantu^.Berikut;
                  {* Menyisipkan Simpul Baru *}
                  Baru^.Berikut := Bantu^.Berikut;   {Gbr(b)}
                  Bantu^.Berikut := Baru;         {Gbr(c)}
            End;
      End;
```

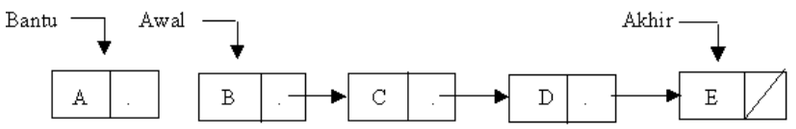
Menghapus Simpul



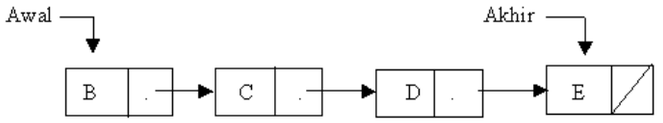
A. Menghapus Simpul Pertama



a.

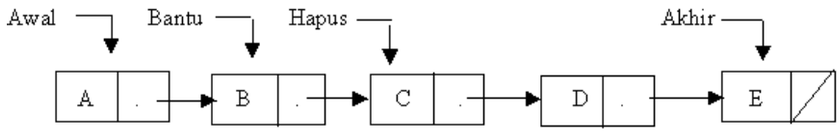


b.

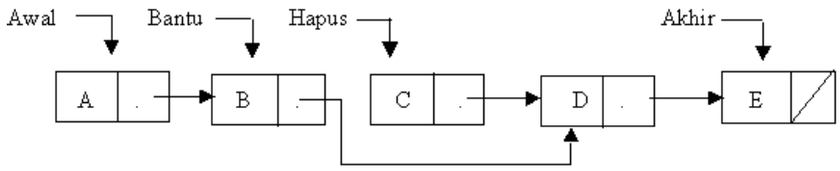


c.

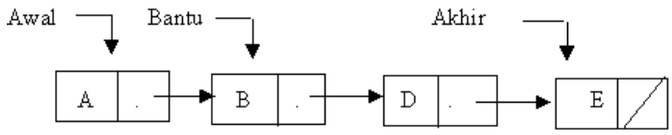
B. Menghapus Simpul di Tengah atau Terakhir



a.



b.

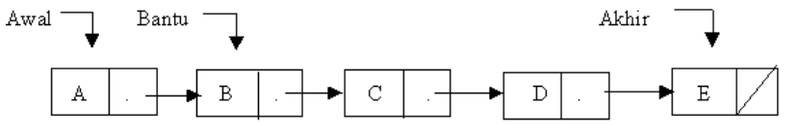


c.

```
Procedure Hapus_simpul;
Begin
        If Awal = Nil Then
                Writeln(' Linked List Masih Kosong')
        Else
                If Awal^.Info = Elemen Then
                {* Simpul pertama dihapus *}
                Begin
                        Hapus := Awal;
                        Awal := Hapus^.Berikut;
                        Dispose(Hapus);
                End
        Else
                {* Menghapus simpul tengah atau terakhir *}
        Begin
        Bantu := Awal;
        {* Mencari simpul yang akan di hapus *}
        While (Elemen <> Bantu^.Info) and (Bantu^.Berikut <> Nil) Do
                        Bantu := Bantu^.Berikut;
                        Hapus := Bantu^.Berikut;
                        If Hapus <> Nil Then
                        {* Simpul yang akan dihapus telah ketemu *}
                        Begin
                                If Hapus <> Akhir Then
                                {* Simpul tengah dihapus *}
                                Bantu^.Berikut := Hapus^.Berikut
                                Else
                          {* Simpul Terakhir dihapus *}
                                Begin
                                        Akhir := Bantu;
                                        Akhir^.Berikut := Nil
                                End;
                        Dispose(Hapus)
                        End
                        Else
                {* Simpul yang akan dihapus tidak ketemu *}
                Writeln(' Simpul yang akan dihapus tidak ada')
        End
End;
```

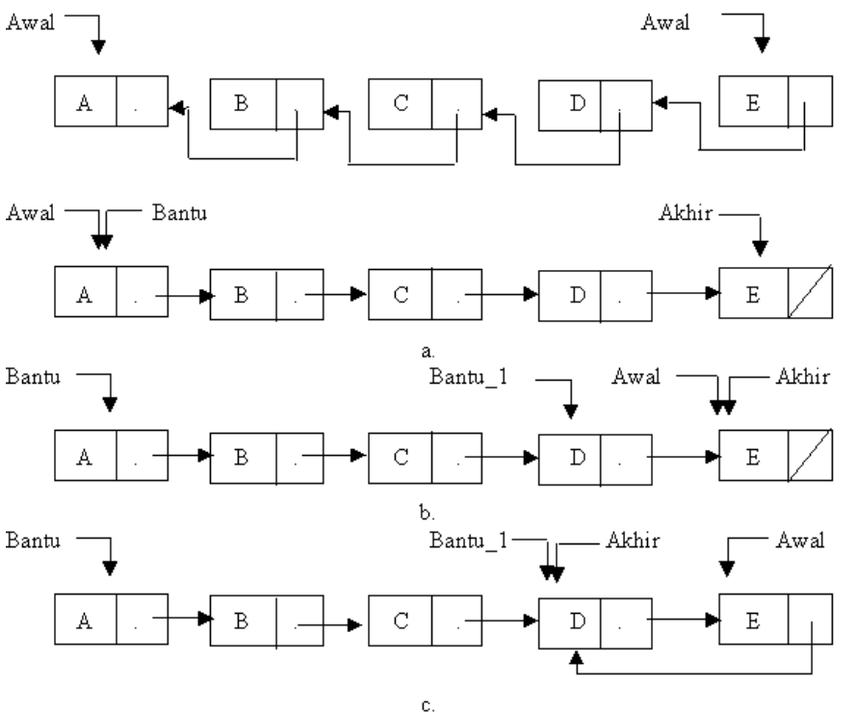
MEMBACA ISI LINKED LIST

A. Membaca Maju




```
Procedure Baca_maju(Awal,Akhir : Simpul);
Var  Bantu : Simpul;
Begin
        Bantu := Awal;
        Repeat
                Write(Bantu^.Info:2);
                Bantu := Bantu^.Berikut
        Until Bantu = Nil;
        Writeln;
End;
```

B. Membaca Mundur




```
Procedure Balik_Pointer(Var Awal,Akhir : Simpul);
Var Bantu,Bantu : Simpul;
Begin
        Bantu := Awal;
        Awal := Akhir;
        {* Proses membalik Pointer *}
        Repeat
                Bantu_1:= Bantu;
                {* Mencari simpul sebelum simpul yang di tunjuk oleh Pointer
Akhir *}
```

```
            While Bantu_1^.Berikut<> Akhir Do
                  Bantu_1:= Bantu_1^.Berikut;
            Akhir ^.Berikut := Bantu_1;
            Akhir := Bantu_1;
      Until Akhir = Bantu;
      Akhir^.Berikut := Nil;
End;
```

CONTOH PROGRAM 1 :

Membuat Record

Nama Tabel : OBAT

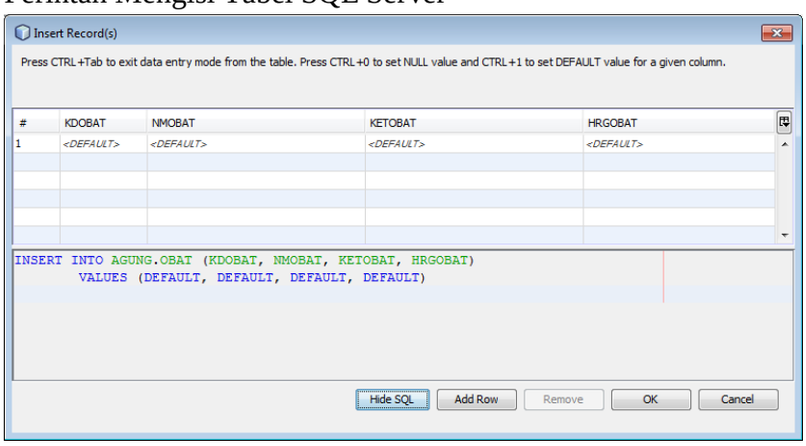
Nama Field	Tipe	Panjang
KdObat	Char	4
NmObat	Char	20
KetObat	Char	30
Harga	Double	

Perintah Membuat SQL Server

```
CREATE DATABASE RSAKIT
ON
( NAME = RSakit_dat,
  FILENAME = 'H:\Data Rsakit\RSakit.mdf ',
  SIZE = 5,
  MAXSIZE = 50,
  FILEGROWTH = 1)
```

Perintah Membuat SQL Server

Perintah Mengisi Tabel SQL Server



CONTOH PROGRAM 2 :

```
package latihan;

/**
 *
 * @author Agung Setiawan
 */
import java.io.*;
class Masukan {
    public static void main(String[] args) throws IOException {
        byte[] data = new byte[10];
        System.out.print("Ketik 10 buah karakter :");
        System.in.read(data);
        System.out.print("Karakter yang Anda ketik yaitu : ");
        for(int i=0;i<data.length;i++) {
            System.out.print((char)data[i]);
        }
    }
}
```

CONTOH PROGRAM 3 :

```
package latihan;

/**
 *
 * @author Agung Setiawan
 */
class IncrementDecrement{
    public static void main (String[] args){

    int i = 1;
                System.out.println("i   : " + i);
          System.out.println("++i : " + ++i);
          System.out.println("i++ : " + i++);
          System.out.println("i   : " + i);
          System.out.println("--i : " + --i);
          System.out.println("i-- : " + i--);
          System.out.println("i   : " + i);
  }
```

BAB V
STACK ATAU TUMPUKAN

Sebuah Record merupakan koleksi satuan data yang heterogen, yakni terdiri dari berbagai type. Satuan data tersebut sering disebut sebagai field dari record. Field dipanggil dengan menggunakan namanya masing-masing. Suatu field dapat terdiri atas beberapa subfield.

Stack atau tumpukan adalah bentuk khusus dari List Linier yang penghapusan serta pemasukan elemennya hanya dapat dilakukan pada satu posisi, yaitu posisi akhir dari List. Prinsip stack adalah terakhir masuk pertama keluar atau masuk terakhir keluar pertama yang dikenal dengan last in first out (LIFO).

Beberapa istilah dalam stack, antara lain :

1. **CREATE(S**) adalah operator yang menyebabkan Stack S menjadi suatu Stack hampa. Maka NOEL(CREATE(S)) adalah 0,dan TOP (CREATE(S)) tidak terdefinisi.
2. **ISEMPTY(S**) adalah operator yang memeriksa apakah Stack S hampa atau tidak. Operandnya adalah data bertype Stack, sedangkan hasilnya merupakan data bertype boolean. ISEMPTY(S) adalah TRUE, jika S hampa, yakni bila NOEL(S) = 0 , dan FALSE dalam hal yang lain. Jelas bahwa ISEMPTY ((CREATE(S)) adalah TRUE.
3. **PUSH(E,S)** adalah operator yang berfungsi menambahkan elemen E pada Stack S dan E ditempatkan sebagai TOP(S). Langkah :
   - Periksa dulu bahwa Top < N
   - Naikan Top dengan 1 (Top = Top + 1)
   - Isikan data kedalam elemen yg ditunjuk oleh Top

```
Procedure Push(x : Type Data);
Begin
   If Top < N Then
   Begin
         Top := Top + 1;
         S[Top] := x;
   End;
   Else Write('Stack Penuh');
End;
```

D. **POP(S)** merupakan operator yang berfungsi megeluarkan elemen TOP(S) dari dalam Stack. POP(S) akan mengurangi nilai NOEL(S) dengan 1. Suatu kesalahan akan terjadi apabila mencoba melakukan POP(S) terhadap Stack S yang hampa.

Langkah :
- Periksa bahwa Top > 0
- Copy data dari elemen yang ditunjuk Top kedalam suatu variabel
- Turunkan Top (Top = Top – 1)

```
Procedure Pop(Var X : Type Data);
Begin
   If Top > 0 Then
   Begin
         X := S[Top];
         Top := Top – 1;
   End;
   Else Write('Stack Kosong');
End;
```

Stack dikenal sangat luas pemakaiannya dalam menyelesaikan berbagai macam problema. Kompiler (Compiler), Sistem Operasi (Operating System) dan berbagai Program Aplikasi (Application Program) banyak menggunakan konsep Stack tersebut. Salah satu contoh adalah problema Penjodohan Tanda Kurung

(Matching Parantheses). Sebuah Kompiler mempunyai tugas, salah satu diantaranya adalah menyelediki apakah Pemrogram telah dengan cermat mengikuti aturan tatabahasa, atau sintaks dari bahasa pemrogramman bersangkutan.

Misalnya untuk Parenthesis kiri (kurung buka) yang diberikan harus dipastikan adanya Parenthesis Kanan (kurung tutup) yang bersangkutan. Algoritma yang dapat diberikan adalah :

- Amati barisan elemen dari kiri ke kanan
- Jika bertemu dengan suatu Parenthesis kiri, maka harus di PUSH ke dalam sebuah Stack
- Jika bertemu dengan suatu Parenthesis kanan, maka harus diperiksa apakah Stack yang ada hampa atau tidak.
  - Jika Stack Hampa berarti adanya Parenthesis kanan tanpa adanya Parenthesis kiri. Suatu kesalahan atau Error.
  - Jika Stack tidak hampa berarti tidak diperoleh sepasang Parenthesis kiri, dan kanan, POP elemen keluar Stack.
  - Jika sampai berakhirnya barisan elemen , ternyata Stack tidak hampa berarti terdapat Parenthesis kiri yang tidak tertutup dengan Parenthesis kanan, hal inipun merupakan suatu kesalahan.

NOTASI POSTFIX,INFIX DAN PREFIX

Infix	Prefix	Postfix
A+B A+B–C	+AB -+ ABC	 AB+C-
(A+B)*(C–D)	*+ AB–CD	AB+CD-*
A–B / (C*D $ E)	- A/B*C$DE	ABCDE$*/-

Berikut ini diberikan sebuah Algoritma untuk mengubah notasi infix kedalam notasi posfix. Dalam hal ini digunakan sebuah Stack kemudian diamati setiap ekspresi satu persatu dari kiri ke kanan.Pada Algoritma terdapat 4 aturan dasar, sebagai berikut :

- Jika simbol adalah "(" (kurung buka), maka ia di PUSH kedalam Stack
- Jika simbol adalah ")" (kurung tutup), POP dari Stack elemen-elemen Stack, sampai pertama kali di POP simbol "(". Semua elemen Stack yang di POP tersebut merupakan output, kecuali "(" tadi.
- Jika simbol adalah sebuah Operator, maka jika TOP Stack adalah operator dengan level lebih tinggi atau sama, maka elemen TOP dapat di POP, sekaligus keluar sebagai output, dilanjutkan proses seperti proses ini sampai TOP merupakan "(" atau Operator dengan level lebih rendah. Kalau hal ini terjadi, Operator (yang diamati) di PUSH kedalam Stack. Biasanya ditambahkan simbol ; (titik-koma) sebagai penutup ekspresi. Dalam keadaan ini di POP semua elemen Stack, sehingga Stack menjadi hampa.

Dapat dicatat bahwa terdapat 3 level operator, yaitu :

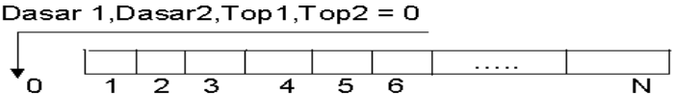
Level	Operator
Tertinggi	Pemangkatan (^)
Menengah	Perkalian (*) dan Pembagian (/)
Terendah	Penjumlahan (+) dan Pengurangan (-)

PENGGUNAAN PRINSIP MULTI STACK ARRAY
(Penggunaan Satu Buah Array untuk lebih dari satu Stack)

I. Dasar Kedua Stack Berada Pada Sisi Indeks Terkecil

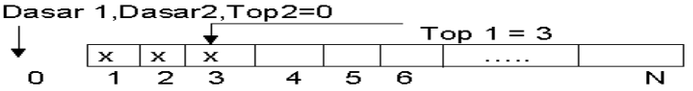
1. Kondisi Pointer Pada Saat Awal

Dasar 1,Dasar2,Top1,Top2 = 0



1. Jika Stack 1 diisi 3 elemen (x = isi satck) dan Stack 2 belum diisi maka :

Dasar 1,Dasar2,Top2=0



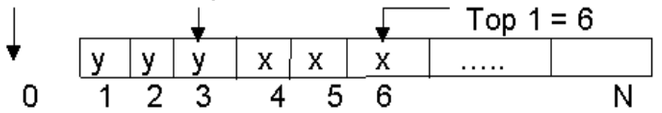
2. Kemudian Jika Stack 2 diisi 1 elemen (y) maka urutan operasi adalah :

- Geser Stack 1 (semua elemen digeser 1 langkah)
  Diperoleh : Top 1 = 4; Dasar 1 = 1
- Isi Stack 2 dengan y maka : Top 2 = 1,Dasar 2 = 0

Dasar 2 = 0 Top 2 = Dasar 1 = 1

3. Kalau Stack 2 diisi lagi, terus menerus sampai 3 elemen

Dasar 2 = 0 Top 2=Dasar 1 = 3

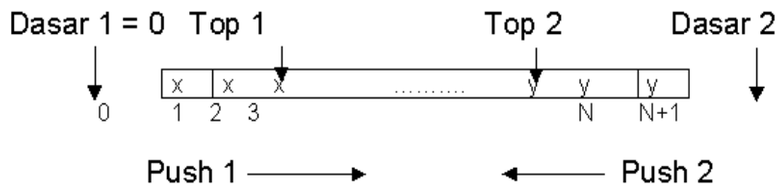


1. Demikian seterusnya sampai array penuh. Pada saat array penuh
   Top 1 = N
   Top 2 = Dasar 1
   Dasar 2 = 0
   Stack 1 dan Stack 2 tidak dapat diisi lagi.
2. Stack 1 atau Stack 2 yang akan ditempatkan didepan tergantung pada perencanaan dan siapa yang akan bergerak terlebih dahulu.

Kesimpulan :
Syarat : Top1 danTop 2 tidak boleh saling mendahului, yg lebih dulu bergerak tetap lebih dulu.

Kondisi	Stack 1	Stack 2
Awal (belum diisi)	Top1 = Dasar1 =0	Top2=Dasar2=0
Kosong	Top1 = Dasar1	Top2=Dasar2
Masih dpt diisi	Kalau array belum penuh	sda stack1
Penuh (tdk dpt diisi)	(Top1–Dasar 1) + (Top2-Dasar2)= N	sda stack1

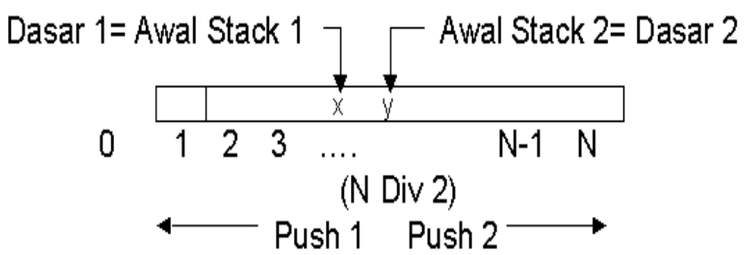
I. **Dasar Stack 1 disisi Indeks Terkecil dan Dasar Stack 2 disisi Indeks Terbesar.**



Syarat : Top 2 > Top 1
Dasar 1 Tetap = 0, Dasar 2 Tetap = N+1

Kondisi	Stack 1	Stack 2
Awal (belum diisi)	Top1 = 0	Top2 = N+1
Kosong	Top1 = 0	Top2 = N+1
Masih Dapat diisi	(Top2 – Top1) > 0	(Top2 – Top1) > 1
Penuh	(Top2 – Top1) = 1	(Top2 – Top1) = 1

II. **Dasar Kedua Stack Diletakkan Tepat di Tengah Indeks.**



Kondisi	Stack 1	Stack 2
Awal (belum diisi)	Top1 = (N Div 2)	Top2 = (N Div 2) + 1
Kosong	Top1 = (N Div 2)	Top2 = (N Div 2) + 1
Masih Dapat diisi	Top1 > 1	Top2 < N
Penuh	Top1 = 1	Top2 = N

CONTOH PROGRAM 1 :

```
public class Urut1 {
   public static void main(String[] args) {
    int y;
    int [] x;
    x = new int[2];
    x[0] = 30 ;
    x[1] = 20 ;
    if(x[0]>x[1]){
       y = x[0];
       x[0] = x[1];
       x[1] = y;}
    System.out.println("Urut  " + x[0] );
    System.out.println("Urut  " + x[1] );
   }
}
```

CONTOH PROGRAM 2 :

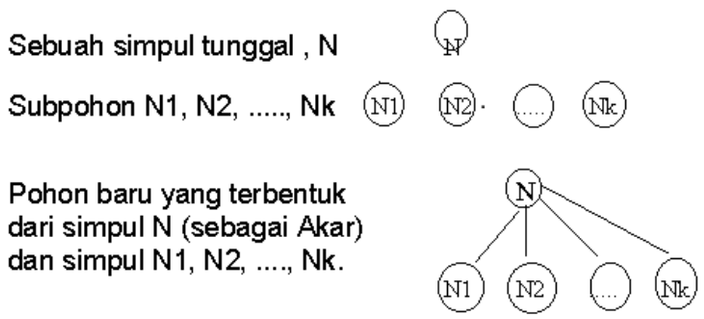
```
/**
 *
 * @author Agung Setiawan
 */
public class Urut2 {
   public static void main(String[] args) {
    int y;
    int [] x;
    x = new int[3];
    x[0] = 30 ;    x[1] = 20 ;      x[2] = 10 ;

    if(x[0]>x[1]){
       y = x[0];   x[0] = x[1];     x[1] = y;}
    if (x[0]>x[2]){
       y = x[0];   x[0] = x[2];     x[2] = y;}
    if (x[1]>x[2]){
       y = x[1];   x[1] = x[2];     x[2] = y;  }
    System.out.println("Urut " + x[0] );
    System.out.println("Urut " + x[1] );
    System.out.println("Urut " + x[2] );}}
```

BAB VI
STRUKTUR POHON (TREE)

Pohon atau Tree adalah salah satu bentuk Graph terhubung yang tidak mengandung sirkuit. Karena merupakan Graph terhubung, maka pada Pohon (Tree) selalu terdapat Path atau Jalur yang menghubungkan setiap simpul dalam dua pohon.

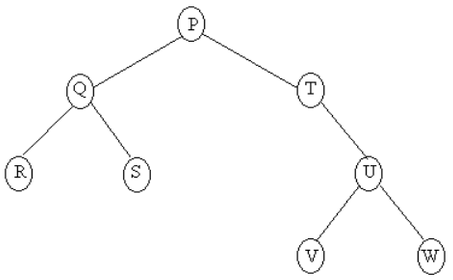
Secara sederhana Pohon (Tree) dapat juga didefinisikan sebagai kumpulan elemen yang salah satu elemennya disebut dengan Akar (Root) dan sisa elemen yang lain (Yang disebut Simpul) terpecah menjadi sejumlah himpunan yang saling tidak berhubungan satu sama lain, yang disebut dengan Subpohon (Subtree), atau disebut juga dengan cabang.

Gambar berikut ini menjelaskan tentang pembentukan awal dari Pohon (Tree).



Sifat utama Pohon Berakar adalah :

1. Jika Pohon mempunyai Simpul sebanyak n, maka banyaknya ruas atau edge adalah (n-1).
2. Mempunyai Simpul Khusus yang disebut Root, jika Simpul tersebut memiliki derajat keluar >= 0, dan derajat masuk = 0.
3. Mempunyai Simpul yang disebut sebagai Daun atau Leaf, jika Simpul tersebut berderajat keluar = 0, dan berderajat masuk = 1.
4. Setiap Simpul mempunyai Tingkatan atau Level yang dimulai dari Root yang Levelnya = 1 sampai dengan Level ke - n pada daun paling bawah.
   Simpul yang mempunyai Level sama disebut Bersaudara atau Brother atau Stribling.
5. Pohon mempunyai Ketinggian atau Kedalaman atau Height, yang merupakan Level tertinggi
6. Pohon mempunyai Weight atau Berat atau Bobot, yang merupakan banyaknya daun (leaf) pada Pohon.



Pohon Berakar T

Pohon Diatas Mempunyai :

1. Banyak Simpul adalah 8 dan banyak edge = n - 1 = 7
2. Root pada Pohon T = Simpul P
3. Mempunyai daun (Leaf) = R, S, V dan W
4. Level (tingkatan) Pohon :
   - Level 1 = Simpul P
   - Level 2 = Simpul Q dan T
   - Level 3 = Simpul R, S dan U
   - Level 4 = Simpul V dan W

5. Ketinggian atau kedalaman = level = 4
6. Weight atau berat atau bobot = 4
7. Banyaknya Simpul Maksimum sampai Level N adalah : $\mathbf{2^{(N)} - 1}$
8. Banyaknya Simpul untuk setiap Level adalah :c

$$\sum_{I=1}^{N} 2^{(I-1)}$$

9. Banyaknya Simpul yang terdapat dalam simpul Tree :

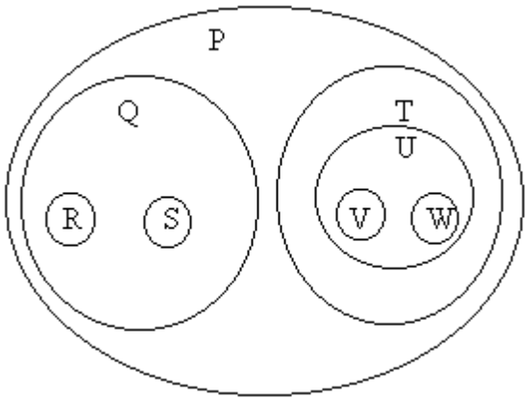
$$\mathbf{2^{(N-1)} - 1}$$

10. Banyaknya Simpul daun yang terdapat dalam Tree

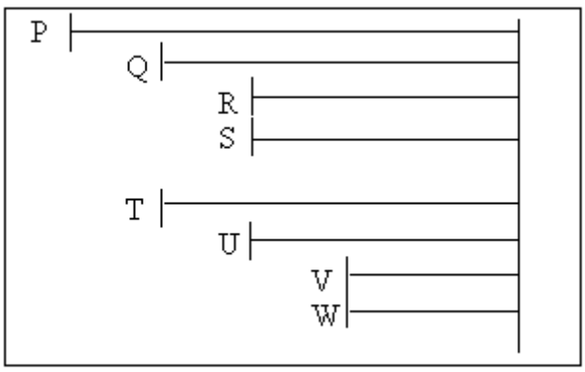
$$\mathbf{2^{(N-1)}}$$

Hutan (Forest) adalah kumpulan sejumlah Pohon yang tidak saling berhubungan, dalam gambar Pohon diatas terdapat 2 buah hutan (forest), yaitu :
Hutan 1 : Q,R,S
Hutan 2 : T,U,V,W

Ada beberapa cara untuk menggambarkan bentuk pohon, yaitu :
1. Cara Pertama Merupakan cara yang paling banyak digunakan dan paling mudah adalah dengan membuat gambar seperti pada gambar diatas.
2. Cara Kedua Dengan membuat Diagram Venn seperti dibawah ini



3. Cara Ketiga Dengan menggunakan Notasi Kurung. Berikut ini diberikan Notasi Kurung untuk Gambar pada diagram Venn diatas. Hasil :
   (P(Q(R,S)),T(U(V,W)))
4. Cara Keempat adalah menggunakan notasi Tingkat dan Notasi Garis



5.1. Pohon Biner (Binary Tree)

Dalam Struktur Data, Pohon memegang peranan yang cukup penting. Struktur ini biasangya digunakan untuk menyajikan data yang mengandung hubungan hirarkial antara elemen-elemennya.

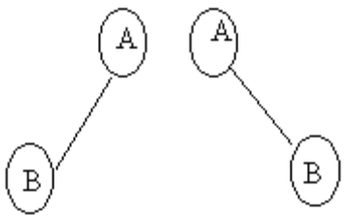
Bentuk Pohon Berakar yang Khusus yang lebih mudah dikelola dalam komputer adalah Pohon Biner (Binary Tree) yang lebih dikenal sebagai Pohon Umum (General Tree) yang dapat didefinisikan sebagai kumpulan simpul yang mungkin kosong atau mempunyai akar dan dua Subpohon yang saling terpisah yang disebut dengan Subpohon Kiri (Left Subtree) dan Subpohon Kanan (Right Subtree).

Untuk selanjutnya digunakan istilah Cabang Kiri untuk menyatakan Subpohon Kiri dan Cabang Kanan untuk Subpohon Kanan.

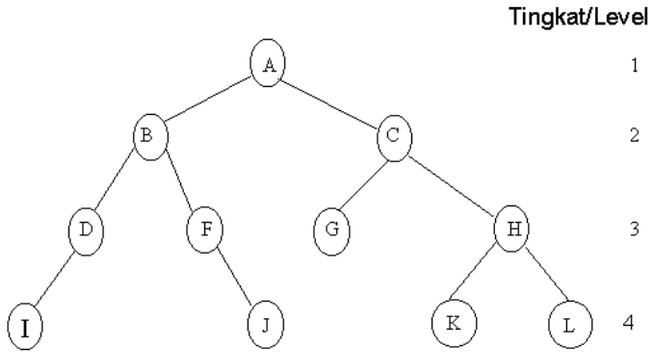
Karakteristik Pohon Biner (Binary Tree) :

1. Setiap Simpul paling banyak hanya memiliki dua buah anak.
2. Derajat Tertinggi dari setiap Simpul adalah dua.
3. Dibedakan antara Cabang Kiri dan Cabang Kanan.
4. Dimungkinkan tidak mempunyai Simpul

Berikut ini diberikan contoh gambar Pohon Biner (Binary Tree) dengan Cabang Kiri dan Cabang Kanan.



Pohon Biner Lengkap 4 Tingkat

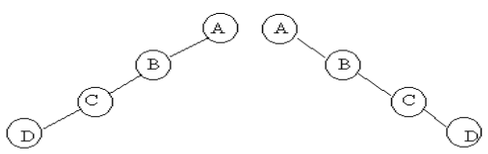


Istilah – istilah dalam Pohon Biner

1. Pohon Biner Similer : Struktur Sama, tetapi Informasi Berbeda

2. Pohon Biner Ekivalent : Struktur dan Informasi Sama

3. Pohon Biner Miring (Skewed Tree) :



Pohon Binar Miring (Skewed Binary Tree) yaitu suatu Pohon Binar yang Semua simpulnya mempunyai satu turunan, kecuali simpul Daun (Leaf).

Dengan memperhatikan gambar sembarang Pohon Biner akan didapat tambahan informasi, yaitu banyaknya simpul maksimal pada tingkat N adalah :

$2^{(N)} - 1$

5.2. Penyajian Pohon Biner (Binary Tree)

Implementasi pohon Biner dalam memory komputer dapat dilakukan dengan beberapa cara. Cara pertama adalah dengan menggunakan Linked List dan

cara yang lain adalah dengan menggunakan cara berurutan.

5.3. Deklarasi Pohon

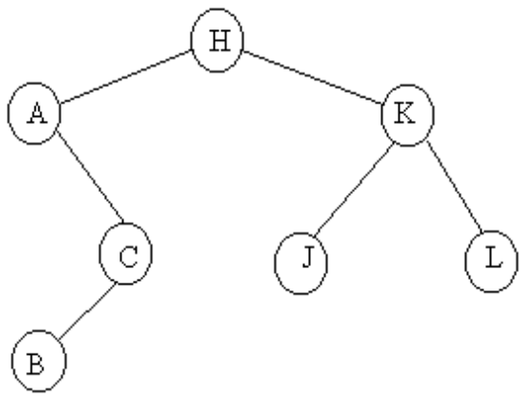
Dalam setiap simpul selalu berisi dua buah Pointer untuk menunjuk ke cabang Kiri dan cabang Kanan dan informasi yang akan disimpan dalam simpul tersebut. Deklarasi simpul yang sesuai adalah :

```
Type    Tree     = ^Simpul;
        Simpul = Record
                    Info  : Char;
                    Kiri, Kanan  : Tree;
End;
```

5.4. Membuat Pohon Biner

Untuk membuat sebuah pohon Biner dapat dilakukan dengan dua cara. Cara Pertama adalah dengan cara Non Rekursif dan cara kedua adalah dengan cara Rekursif.

Selain kedua cara diatas, perlu juga memperhatikan kapan suatu simpul akan dipasang sebagai cabang kiri dan kapan sebagai cabang kanan. Dalam Algoritma dibawah ini Cabang Kiri dan Cabang Kanan ditentukan dengan cara : Simpul yang berisi informasi yang nilainya " Lebih Besar" dari simpul yang berikut akan ditempatkan sebagai cabang Kanan, jika "Lebih Kecil" akan ditempatkan sebagai Cabang Kiri.



Proses untuk memperoleh bentuk Pohon Binar seperti terlihat pada Gambar Diatas dari untai 'HAKJCBL' adalah sebagai berikut : Karakter pertama 'H' ditempatkansebagai Akar (Root). Karakter 'K' , karena 'lebih besar' dari 'H' akan menempati cabang Kanan. Karakter 'A' karena 'lebih kecil' dari 'H' akan menempati cabang kiri dai 'H'. Kemudian karena karakter 'C' , 'lebih kecil' dari 'H' dan 'lebih besar' dari 'A', maka diletakkan sebagai cabang kanan dari 'A'. Demikian seterusnya sampai semua masukkan diproses.

```
Procedure Baru(huruf : Char) : Tree;
Var  B : Tree;
Begin
      New(B);
      B^.Info  := Huruf;
      B^.Kanan := Nil;
      B^.Kiri  := Nil;
      Baru := B;
End;
```

CONTOH PROGRAM :

```
public class Pohon1 {
   int T;
   int T1;
   int T2;
   int T3;
   int T4;

   /** Creates a new instance of PersonToy */
   public Pohon1() {
         T = 0;
         T1 = 1;
         T2 = 2;
         T3 = 3;
         T4 = 4;
   }

   public Pohon1(int newT, int newT1, int newT2, int newT3, int newT4) {
         T = newT;
         T1 = newT1;
         T2 = newT2;
         T3 = newT3;
         T4 = newT4;
   }

   public void Pohon (int newT){
         T = newT;
   }
```

```
public void setT1 (int newT1){
        T1 = newT1;
}

public void setT2 (int newT2) {
        T2 = newT2;
}

public void setT3 (int newT3) {
        T3 = newT3;
}


public void setT4 (int newT4) {
        T4 = newT4;
}

public int getT (){
        int [] x;
     x = new int[5];
     x[0] = 10;
     x[1] = 11;
     x[2] = 12;
     x[3] = 13;
     x[4] = 14;
System.out.println("Nilai x[0] : " + x[0]);
System.out.println("Nilai x[1] : " + x[1]);
System.out.println("Nilai x[2] : " + x[2]);
System.out.println("Nilai x[3] : " + x[3]);
System.out.println("Nilai x[4] : " + x[4]);
    return T;
}

public int getT1 (){
        int [] x;
     x = new int[5];
     x[0] = 20;
     x[1] = 21;
     x[2] = 22;
     x[3] = 23;
     x[4] = 24;
System.out.println("Nilai x[0] : " + x[0]);
System.out.println("Nilai x[1] : " + x[1]);
System.out.println("Nilai x[2] : " + x[2]);
System.out.println("Nilai x[3] : " + x[3]);
System.out.println("Nilai x[4] : " + x[4]);
    return T1;
```

```
}

public int getT2 (){
        int [] x;
     x = new int[5];
     x[0] = 30;
     x[1] = 31;
     x[2] = 32;
     x[3] = 33;
     x[4] = 34;
System.out.println("Nilai x[0] : " + x[0]);
System.out.println("Nilai x[1] : " + x[1]);
System.out.println("Nilai x[2] : " + x[2]);
System.out.println("Nilai x[3] : " + x[3]);
System.out.println("Nilai x[4] : " + x[4]);
    return T2;
}

public int getT3 (){
        int [] x;
     x = new int[5];
     x[0] = 40;
     x[1] = 41;
     x[2] = 42;
     x[3] = 43;
     x[4] = 44;
System.out.println("Nilai x[0] : " + x[0]);
System.out.println("Nilai x[1] : " + x[1]);
System.out.println("Nilai x[2] : " + x[2]);
System.out.println("Nilai x[3] : " + x[3]);
System.out.println("Nilai x[4] : " + x[4]);
    return T3;
}

public int getT4 (){
        int [] x;
     x = new int[5];
     x[0] = 50;
     x[1] = 51;
     x[2] = 52;
     x[3] = 53;
     x[4] = 54;
System.out.println("Nilai x[0] : " + x[0]);
System.out.println("Nilai x[1] : " + x[1]);
System.out.println("Nilai x[2] : " + x[2]);
System.out.println("Nilai x[3] : " + x[3]);
System.out.println("Nilai x[4] : " + x[4]);
    return T4;
```

```
    }

    public String toString(){
        String str =
            "Root    : "+ T + "\n"+
            "Node 1  : "+ T1 + "\n"+
            "Node 2  : "+ T2 + "\n"+
            "Node 3  : "+ T3 + "\n"+
            "Node 4  : "+ T4 + "\n";
        return str;
    }

    static void test(){
        Pohon1 t = new Pohon1(0,1,2,3,4);
        System.out.println("Pohon sebagai Tree:");
        System.out.println(t.getT());
        System.out.println(t.getT1());
        System.out.println(t.getT2());
        System.out.println(t.getT3());
        System.out.println(t.getT4());
        System.out.println(t);

    }

    public static void main(String[] args) {
        test();

    }
}
```

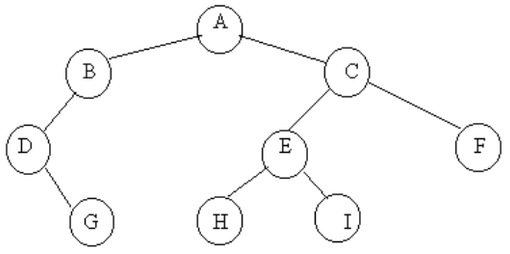
BAB VII
KUNJUNGAN PADA POHON BINER

Kunjungan pada pohon biner merupakan salah satu operasi yang sering dilakukan pada suatu pohon biner tepat satu kali atau Binary tree Traversal. Operasi ini terbagi menjadi 3 bentuk, yaitu :

1. Kunjungan secara Preorder (Depth First Order), mempunyai urutan :
   a. Cetak isi simpul yang dikunjungi (simpul akar).
   b. Kunjungi cabang kiri
   c. Kunjungi cabang kanan
2. Kunjungan secara inorder (Symetric Order), mempunyai urutan :
   a. Kunjungi cabang kiri
   b. Cetak ini simpul yang dikunjungi (simpul akar)
   c. Kunjungi cabang kanan
3. Kunjungan secara Postorder, mempunyai urutan :
   a. Kunjungi Cabang Kiri
   b. Kunjungi Cabang Kanan
   c. Cetak isi simpul yang dikunjungi (Simpul Akar)

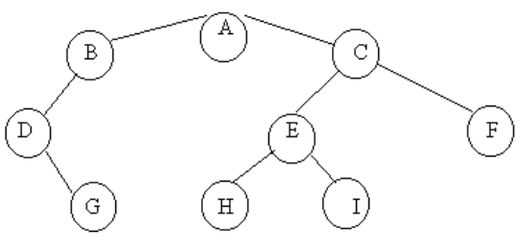
Pada ketiga cara kunjungan diatas, kunjungan ke Cabang Kiri dilakukan terlebih dahulu, baru kemudian kunjungan ke Cabang Kanan. Dengan orientasi semacam ini, Ketiga kunjungan diatas disebut dengan Left To Right Oriented (LRO). Jika kunjungan ke Cabang Kanan dilakukan lebih dahulu baru kemudian kunjungan ke Cabang Kiri, maka Orientasi semacam ini disebut Right To Left Oriented (RLO).

Kunjungan PreOrder



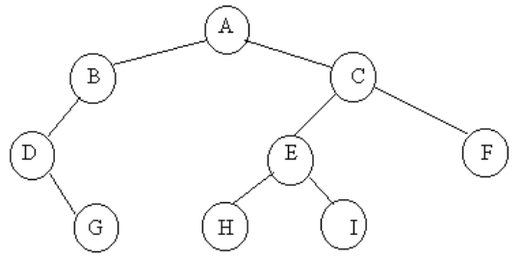
Hasil : A B D G C E H I F

Kunjungan InOrder



Hasil : D G B A H E I C F

Kunjungan PostOrder



Hasil : G D B H I E F C A

7.1. Kunjungan LevelOrder

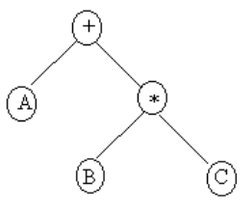
Selain kunjungan yang dijelaskan diatas, masih ada satu macam kunjungan masih ada satu macam kunjungan lagi yaitu kunjungan LevelOrder. Kunjungan dimulai dari simpul yang ada pada tingkat 1 (Akar), diteruskan pada simpul di tingkat 2, tingkat 3 dan seterusnya.

Secara singkat kunjungan Level Order ini dapat dijelaskan sebagai berikut. Dimulai dengan memasukkan Akar kedalam antrean. Kemudian mengeluarkan Akar tersebut keluar dari antrean. Pada saat Akar tersebut dikeluarkan dari antrean, cabang kiri dan cabang kanan secara berturut-turut dimasukkan dalam antrean.

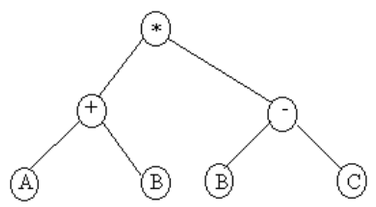
Dengan kata lain jika suatu elemen dikeluarkan dari antrean, maka cabang kiri dan kanan dari elemen yang baru saja dikeluarkan dimasukkan kedalam antrean.

7.2. Aplikasi Pohon Biner

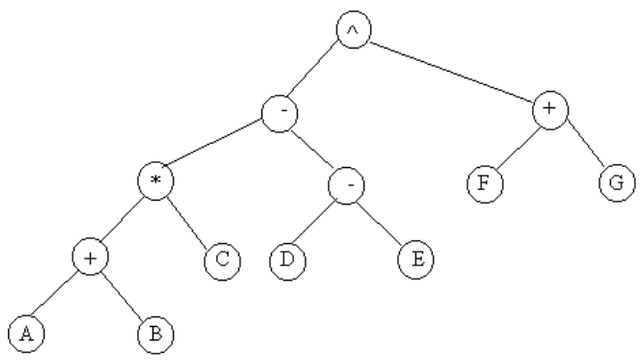
Pada bagian ini akan dibahas tentang bagaimana menyusun sebuah Pohon Binar yang apabila dikunjungi secara PreOrder akan menghasilkan Notasi Prefix, kunjungan secara InOrder menghasilkan Notasi Infix, dan kunjungan PostOrder menghasilkan Notasi Postfix.



(a)



(b)



(c)

Berdasarkan Gambar diatas, apabila dilakukan kunjungan secara PreOrder, maka akan diperoleh Notasi Prefix dari persamaan-persamaan yang digambarkan tersebut, yaitu :

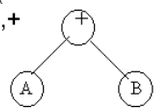
+A*BC	(Gambar.a)
*+AB-BC	(Gambar.b)
^-*+ABC-DE+FG	(Gambar.c)

Jika dilakukan kunjungan secara PostOrder, akan diperoleh Notasi Postfixnya, yaitu :

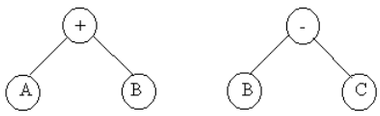
ABC*+	(Gambar.a)
AB+BC-*	(Gambar.b)
AB+C*DE-FG+^	(Gambar.c)

Contoh pembentukan pohon Binar dari persamaan : (A+B)*((B-C)+D)

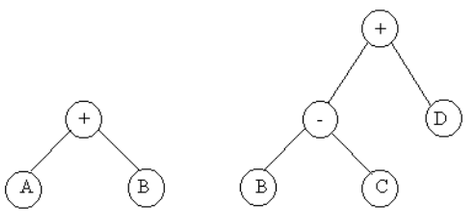
1. Karakter yang dibaca : (,A,+,B,) Pohon Binar yang terbentuk :
   Tumpukan Operator : (,(,+(,+(
   Tumpukan Operand : A,A,BA,+



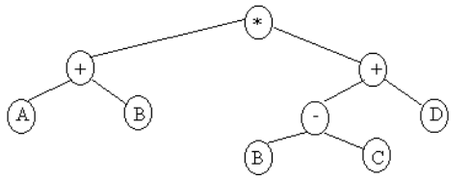
2. Karakter yang dibaca : *,(,(,B,-,C,)
   Tumpukan Operator : *,(*,((*,((*,-((*,-((*,(*
   Tumpukan Operand : +,+,+,B+,B+,CB+,-+
   Pohon Binar yang terbentuk :



3. Karakter yang dibaca : +,D,)
   Tumpukan Operator : +(*,+(*,*
   Tumpukan Operand : -+,D-+,++
   Pohon Binar yang terbentuk :



Berikut adalah Gambar pohon binar yang diperoleh dar persamaan : (A+B)*((B-C)+D).



BAB VIII
SORTING

Operasi Pengurutan (Sorting) adalah operasi yang sangat banyak dilakukan dalam 'Bussiness Data Processing'. Dalam hal ini pengurutan yang dilakukan adalah secara Ascending (menaik dari kecil ke besar) Macam-macam Sorting (Pengurutan), antara lain :

8.1.Buble Sort

Metode pengurutan bubble sort mempunyai algoritma atau prosedur sebagai berikut :

a. Pengecekan dimulai dari data ke – 1 sampai dengan data ke – n
b. Bandingkan data ke – n dengan data sebelumnya (n – 1), jika lebih kecil maka tukar bilangan tersebut dengan data yang didepannya (sebelumnya) satu persatu (n – 1, n – 2, n – 3, … , dan seterusnya)
c. Lakukan langkah ke – 2 sampai didapatkan urutan yang optimal

8.2.Quick Sort

Sangat baik untuk tabel data yang sangat besar. Algoritma atau prosedur quick sort adalah sebagai berikut :

a. Tentukan bilangan yang dinyatakan sebagai batas bawah (Lower Bound (I = 1)) dan bilangan yang dinyatakan sebagai batas atas (Upper Bound (I= N)).
b. Syarat pemindahan adalah LB > UB, dengan melihat perbandingan antara UB (awal bilangan) dan LB (akhir bilangan).
c. Jika LB > UB lakukan pertukaran antara kedua bilangan tersebut (I = I+ 1, I = I +2, …) ke bilangan selanjutnya dan bandingkan kembali dengan UB (I = N, I = N - 1, I = N - 2,…)
d. Lakukan langkah 2 dan 3 untuk bilangan selanjutnya sampai didapatkan urutan yang optimal.

8.3.Selection Sort

Metode pengurutan Selection Sort Prosedur atau Algoritmanya :

a. Pengecekan dimulai dari data ke –1 sampai dengan data ke – n
b. Tentukan bilangan dengan index terkecil dari data bilangan tersebut
c. Tukar bilangan dengan index terkecil tersebut dengan bilangan pertama (I = 1) dari data bilangan tersebut
d. Lakukan langkah 2 dan 3 untuk bilangan berikut (I = I + 1), sampai didapatkan urutan yang optimal.

8.4.Insertion Sort

Digunakan untuk melakukan insert suatu record dalam record-record yang telah diurutkan. Metode pengurutan ini mempunyai algoritma yang mirip dengan bubble sort, dengan prosedur sebagai berikut :

a. Pengecekan dimulai dari data ke – 1 sampai dengan data ke – n.
b. Pengurutan dilakukan dengan cara membandingkan data ke – I (dimana I dimulai dari data ke -2 sampai dengan data yang terakhir).
c. Bandingkan data ke – I tersebut dengan datav sebelumnya (I – 1), jika lebih

kecil maka data tersebut dapat disisipkan ke data awal 9depan) sesuai dengan posisi yang seharusnya.

d. Lakukan langkah ke – 2 dan 3 untuk bilangan selanjutnya (I = I +1) sampai didapatkan urutan yang optimal.

8.5.Merge Sort

Penggunakan metode iteratif merge sort mempunyai algoritma atau prosedur sebagai berikut :

e. Kelompokan deret bilangan kedalam 2 bagian, 4 bagian, 8 bagian, dan seterusnya.
f. Urutan secara langsung bilangan dalam tiap kelompok
g. Langka-langkah diatas untuk kondisi bilangan yang lain sampai didapatkan urutan yang optimal

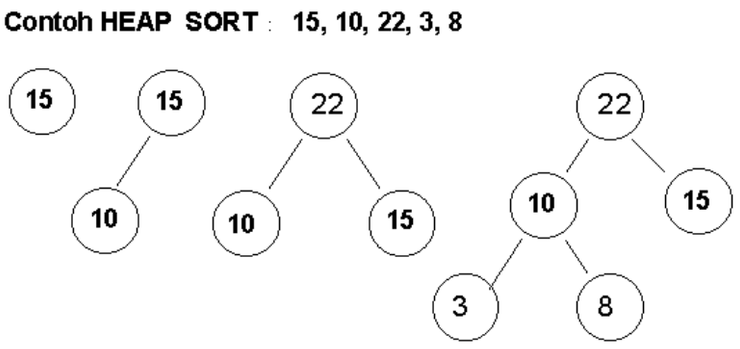
8.6.Heap Sort

Tujuan dari heap sort adalah mencari bilangan terbesar dari data dalam deret bilangan. Langkah-langkah atau algoritma dalam heap sort adalah sebegai berikut :

a. Buat (create) sebuah heap dengan data yang sudah ditentukan.
b. Bandingkan heap dengan element selanjutnya (next element), jika next element > heap, maka jadikan next element tersebut sebagai sebuah root.
c. Kerjakan cabang kiri terlebih dahulu, kemudian kerjakan cabang kanan.

Atau dapat pula dibuat algoritma sebagai ebrikut :

a. Bentuk pohon heap dengan prosedur yang ada
b. Langsanakan sort
   Ulangi sampai dengan langkah 10; Q = N, N – 1, …, 2
c. Tukar nilai record : K[I] < --- > K[Q]
d. Berikan harga awal
   I < --- 1; Key < --- K[I], J < ---2,
e. Dapatkan indeks dari harga terbesar anak dari record baru
   If J+1<Q then
      If K[J+1] > K[J] then
         J:=J+1
f. Buat kembali heap baru
   Ulangi sampai langkah 10 apabila J < Q - 1 dan K[J] > Key
g. Tukarkan harga record : K[I] < --- > K[J]
h. Dapatkan anak kiri berikutnya
   I < --- J: J < --- Q + 1
i. Dapatkan indeks dari anak dengan harga terbesar berikutnya
   If J + 1 < Q then
      If K[J+ 1] > K[J] then
         J = J + 1
      Else
         If J > N then J - N
j. Copykan record kedalam tempatnya yang cocok
   K[I] < --- Key
k. Return



CONTOH PROGRAM 1 :

```
/**
 *
 * @author Agung Setiawan
 */
public class quicksort {
     Comparable items[];

   /** Creates a new instance of QuickSortCToy */
   public quicksort(Comparable[] items) {
      this.items = items;
      sort(items,0,items.length-1);
   }

   private void sort(Comparable[] a, int left, int right){

      System.out.println(toString());

      if(left<right){
         int p=partition(a,left,right);
         sort(a,left,p-1);
         sort(a,p+1,right);
      }
   }

   private static int partition(Comparable[] a, int left, int right){
      Comparable pivot=a[left];
      int p=left;
      for(int r=left+1;r<=right;r++){
         int comp=a[r].compareTo(pivot);
         if(comp<0) {
            a[p] = a[r];
            a[r] = a[p+1];
            a[p+1] = pivot;
            p++;
```

```
            }
        }
        return p;
    }

    public String toString(){
        String str ="";
        for(int i=0;i<items.length;i++){
            str += items[i] + "\t";
        }
        return str;
    }

    static void test(){
        Double[] items = {
            new Double(4.0),
            new Double(12.0),
            new Double(3.0),
            new Double(9.0),
            new Double(1.0),
            new Double(21.0),
            new Double(5.0),
            new Double(2.0)
        };
        quicksort t = new quicksort(items);
        System.out.println(t);
    }

    public static void main(String[] args){
        test();
    }
}
```

CONTOH PROGRAM 2 :

```
public class selekse {
Comparable items[];

    /** Creates a new instance of SelectionSortCToy */
    public selekse(Comparable[] items) {
        this.items = items;
        sort(items,0,items.length-1);}

    private void sort(Comparable[] a, int left, int right){
        for (int l=left;l<right;l++){
            int p = l;
            for(int k=l+1;k<=right;k++){
                int comp = a[k].compareTo(a[p]);
```

```
            if(comp<0)p=k;  }
         if (p!=1) {
            Comparable least = a[p];
            a[p] = a[l];
            a[l] = least;}
         // for testing only
         System.out.println(toString()); }}

   public String toString(){
      String str ="";
      for(int i=0;i<items.length;i++){
         str += items[i] + "\t";  }
      return str;
   }

   static void test(){
      Double[] items = {
         new Double(4.0),
         new Double(12.0),
         new Double(3.0),
         new Double(9.0),
         new Double(1.0),
         new Double(21.0),
         new Double(5.0),
         new Double(2.0)};
      selekse t = new selekse(items);
      System.out.println(t);}

   public static void main(String[] args){
      test(); }}
```

CONTOH PROGRAM 3 :

```
/**
 *
 * @author Agung Setiawan
 */
public class setengah {
   Comparable items[];

   /** Creates a new instance of insertionSortCToy */
   public setengah(Comparable[] items) {
      this.items = items;
      sort(items,0,items.length-1);
   }

   private void sort(Comparable[] a, int left, int right){
      for (int r=left;r<=right;r++){
```

```
            Comparable val=a[r];
            int p=r;
            while (p>left && val.compareTo(a[p-1])<0) {
               a[p] = a[p-1];
               p--;
            }
            a[p] = val;

            // for testing only
            System.out.println(toString());
         }
      }

      public String toString(){
         String str ="";
         for(int i=0;i<items.length;i++){
            str += items[i] + "\t";
         }
         return str;
      }

      static void test(){
         Double[] items = {
            new Double(4.0),
            new Double(12.0),
            new Double(3.0),
            new Double(9.0),
            new Double(1.0),
            new Double(21.0),
            new Double(5.0),
            new Double(2.0)
         };
         setengah t = new setengah(items);
         System.out.println(t);
      }

      public static void main(String[] args){
         test();
      }
   }
```

BAB IX
SEARCHING

Searching adalah salah satu pekerjaan yang paling mendasar dalam bidang perkomputeran. Struktur Searching digunakan dalam setiap tindakan yang perlu, untuk mengetahui sebuah elemen yang tercantum di dalam sebuah daftar, serta percarian ulang dari file informasi yang berhubungan dengan unsur tersebut.

9.1. Linier Searching (Sequential Searching)

Pencarian dimulai dari record ke-1, diteruskan ke record berikutnya, yaitu record ke-2,ke-3 …… dst, sampai diperoleh isi record sama dengan bilangan yang dicari.

```
Procedure Linier_Search(K,N,X);
Var I:=Integer;
Begin I := 1; K[N +1] := X;
  While K[I] <> X Do
  I : = I+1;
  If I = N + 1 Write("Search Gagal")
  Else Write("Serach Sukses")
End;
```

Dalam kasus yang paling buruk, prosedur tersebut memerlukan waktu 0(n). Hal ini dengan jelas merupakan titik optimal karena setiap elemen K harus diuji (Bila X tidak berada dalam K) sebelum dinyatakan gagal.

9.2. Binary Search

Merupakan metode terbaik dalam search (pencarian), karena memulai pencarian dari lokasi tengah (Middle). Kemudian berdasarkan posisi tengah tersebut terdapat 3 kemungkinan :

a. Jika X < K[Middle], maka informasi yang dicari berada dibagian bawah dari lokasi tengah (Middle)
b. Jika X = K[Middle], maka record tengah tersebut adalah informasi yang dicari
c. Jika X > K[Middle], maka informasi yang dicari ada dibagian atas dari lokasi tengah (Middle)

```
Procedure Binary_Search(K,N,X);
Var     Low,High,Middle : Integer;
Begin
    Low :=1;High:=N;
     While Low <= High Do
    Begin
        Middle:=((Low+High) Div 2);
        If X < K[Middle] Then High := Middle – 1
                        Else If X > K[Middle] Then  Low := Middle+1
                        Else Write('Search Sukses')
        End;
        Write('Search Gagal');
End;
```

9.3. Fibonancy Search

Pencarian yang menggunakan deret Fibonancy sebagai dasar pencarian. Deret Fibonancy : 0,1,1,2,3,5,8,13,21,……..

Langkah-langkah yang harus dilalui :

a. Tentukan Angka Pertambahan (Increament).
   Rumus : Fk+M = N+1
   Dimana : Fk = Angka Fibonancy terdekat
   M = Angka Pertambahan
   N = Banyak deret bilangan
b. Tent. Fk-1,Fk-2 dan Fk-3
c. Misalkan I=Fk-1,P=Fk-2,Q=Fk-3
d. Lakukan Algorithma Jika X>K[i], maka i = i + M

Lakukan Perulangan bila i <> 0 yang berisi :
Case Of : 1. X<K[I] : jika Q=0, maka I=0
jika Q<>0,
I=I-Q,t=P,P=Q,Q=t-Q
2. X=K[I] : "Search Sukses"
3. X>K[I] : jika P=1, maka I=0
jika P<>1,
I=I+Q,P=P-Q,Q=Q-P

CONTOH PROGRAM :

```
/**
 *
 * @author Agung Setiawan
 */
public class setengah {
   Comparable items[];

   /** Creates a new instance of insertionSortCToy */
   public setengah(Comparable[] items) {
      this.items = items;
      sort(items,0,items.length-1);
   }

   private void sort(Comparable[] a, int left, int right){
      for (int r=left;r<=right;r++){
         Comparable val=a[r];
         int p=r;
         while (p>left && val.compareTo(a[p-1])<0) {
            a[p] = a[p-1];
            p--;
         }
         a[p] = val;
```

```
            // for testing only
            System.out.println(toString());
        }
    }

    public String toString(){
        String str ="";
        for(int i=0;i<items.length;i++){
            str += items[i] + "\t";
        }
        return str;
    }

    static void test(){
        Double[] items = {
            new Double(4.0),
            new Double(12.0),
            new Double(3.0),
            new Double(9.0),
            new Double(1.0),
            new Double(21.0),
            new Double(5.0),
            new Double(2.0)
        };
        setengah t = new setengah(items);
        System.out.println(t);
    }

    public static void main(String[] args){
        test();
    }
}
```

BAB X
GRAPH

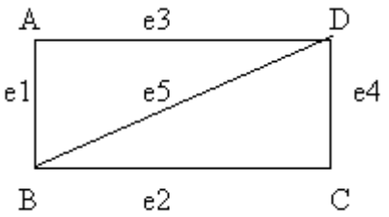
10.1. Graph

Kata Graph di dalam Matematika mempunyai bermacam-macam arti. Biasanya di kenal kata Graph atau Grafik Fungsi, ataupun relasi. Untuk itu kali ini akan digunakan kata Graph dalam arti lain.Suatu Graph mengandung 2 himpunan, yaitu :

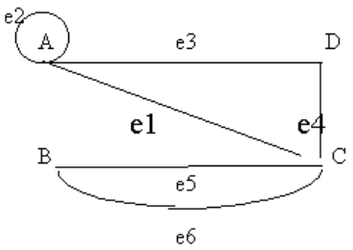
1. Himpunan V yang elemennya disebut simpul (Vertex atu Point atau Node atau Titik)
2. Himpunan E yang merupakan pasangan tak urut dari simpul. Anggotanya disebut Ruas (Edge atau rusuk atau sisi)

Graph seperti dimaksud diatas, ditulis sebagai G(E,V).Sebagai contoh, Gambar berikut menanyakan Graph G(E,V) dengan :

1. V mengandung 4 simpul, yaitu simpul A,B,C,D
2. E mengandung 5 ruas, yaitu :
   e1 = (A,B) e4 = (C,D)
   e2 = (B,C) e5 = (B,D)
   e3 = (A,D)



Banyaknya simpul (vertex) disebut ***Order***, sedangkan banyaknya ruas (edge) disebut ***Size*** dari Graph.

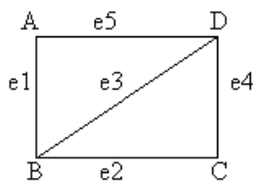


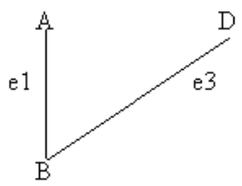
Gambar diatas menyatakan suatu Graph yang lebih umum, disebut ***Multigraph***. Disini, ruas e2 pada kedua titik ujungnya adalah suatu simpul yang sama, yaitu simpul A. Ruas semacam ini disebut ***Gelung atau Self-Loop***.

Sedangkan ruas e5 dan e6 mempunyai titik ujung yang sama, yaitu simpul-simpul B dan C. Kedua ruas ini disebut ruas ***berganda*** atau ruas ***sejajar.*** Suatu Graph yang tak mengandung ruas sejajar ataupun self-loop, sering disebut juga sebagai Graph sederhana atau simple Graph.

Suatu Graph G'(E',V') disebut SubGraph dari G(E,V), bila E' himpunan bagian dari E dan V' himpunan bagian dari V. Jika E' mengandung semua ruas dari E yang titik ujungnya di V', maka G' disebut Subgraph yang direntang oleh V' (Spanning Subgraph).

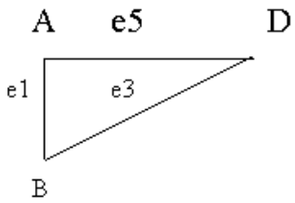
Contoh :





G' Subgraph dari G (namun bukan dibentuk olehV'={A,B,D}).

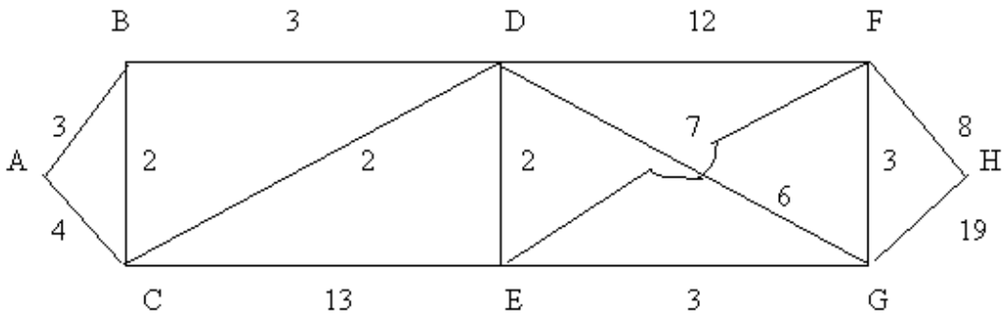
G' Subgraph yang dibentuk oleh V ' = {A,B,D}



Suatu Multigraph disebut juga hingga apabila mempunyai hingga simpul dan sejumlah hingga ruas.

10.2. Graph Berlabel

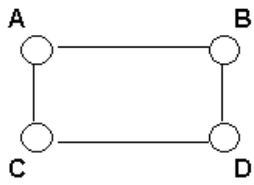
Graph G disebut berlabel jika ruas dan atau simpulnya dikaitkan dengan suatu besaran tertentu. Khususnya jika stiap Ruas e dari G dikaitkan dengan suatu bilangan non negatif d(e), maka d(e) disebut bobot atau panjang dari ruas e. Sebagai contoh, Gambar berikut ini menyajikan hubungan antar kota. Disini simpul menyatakan kota dan label d(e) menyatakan jarak antara dua kota.



10.3. Derajat Graph

Derajat simpul V, ditulis d(v) adalah banyaknya ruas yang menghubungi v. Karena setiap ruas dihitung dua kali ketika menentukan derajat suatu Graph, maka:

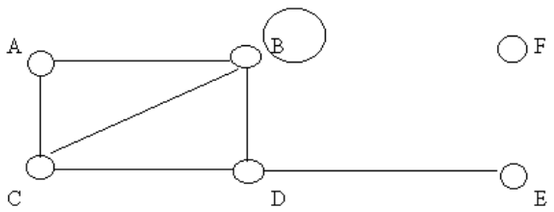
Jumlah derajat semua simpul suatu Graph (disebut derajat) =
dua kali banyaknya ruas Graph (Size atau ukuran Graph).



Dari gambar diatas Jumlah Semua Simpul = 4, maka Jumlah Derajat Semua

Simpul = 8. Bila Jumlah Derajat Semua Simpul sama dengan Genap, maka disebut **EULER Graph**. Suatu simpul disebut genap/ganjil tergantung apakah derajat simpul tersebut genap/ganjil. Kalau terdapat self-loop, maka self-loop dihitung 2 kali pada derajat simpul.

Suatu simpul disebut genap/ganjil tergantung apakah derajat simpul tersebut genap/ganjil. Jika terdapat self-loop, maka self loop dihitung 2 kali pada derajat simpul.



Disini banyak ruas = 7, sedangkan derajat masing-masing simpul adalah :

d(A) = 2 maka, total jumlah derajat simpul adalah : 14
d(B) = 5
d(C) = 3
d(D) = 3
d(E) = 1
d(F) = 0

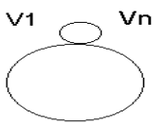
Catatan :
Disini E disebut simpul bergantung/akhir, yaitu simpul yang berderajat satu. Sedangkan F disebut simpul terpencil, yaitu simpul yang berderajat Nol.

10.4. Keterhubungan

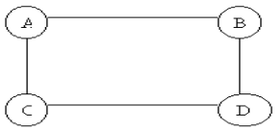
Walk atau perjalanan dalam Graph G adalah barisan simpul dan ruas berganti-ganti : V1,e1,V2,e2,....., e n-1, Vn. Disini ruas ei menghubungkan simpul Vi dan Vi+1.

Banyaknya ruas disebut Panjang Walk. Walk dapat ditulis lebih singkat dengan hanya menulis deretan ruas : e1,e2, ..., en-1atau deretan simpul : V1, V2, ..., Vn-1, Vn dimana : V1 = simpul awal Vn = simpul akhir. Walk disebut tertutup bila V1 = Vn.

Contoh : Cycle



1. Walk disebut terbuka, yang menghubungkan V1 dan Vn, yaitu Setiap Ruas menghubungkan Simpul Awal dan Akhir

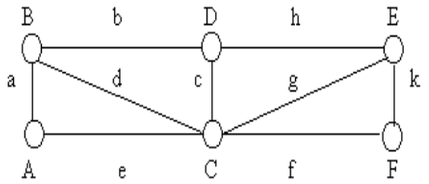


2. TraiL adalah Walk dengan semua ruas dalam barisan adalah berbeda.
3. Path atau Jalur adalah Walk yang semua simpul dalam barisan adalah berbeda. Jadi suatu Path pastilah sebuah Trail.

Graph diatas Bukan WALK, karena tidak ada ruas yang menghubungkan Simpul U dan T, tetapi merupakan suatu Path atau Trail terbuka dengan derajat setiap simpulnya = 2, kecuali simpul awal V1 dan akhir Vn berderajat = 1.

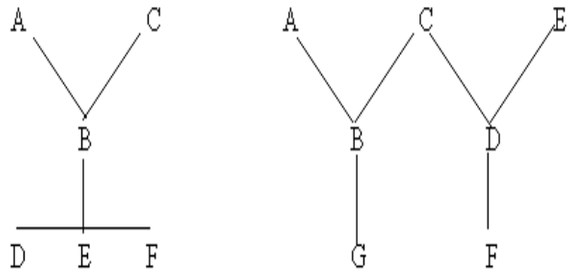
1. Cycle atau sirkuit adalah suatu Trail tertutup dengan derajat setiap simpul = 2. Cycle dengan panjang k disebut dengan k-cycle. Demikian pula Jalur dengan panjang k disebut k-jalur.

Contoh :



- Barisan ruas a,b,c,d,b,f,g,h adalah Walk bukan Trail (ruas b dua kali muncul).
- Barisan simpul A, B, E, F bukan Walk (tak ada ruas menghubungkan simpul B ke F).
- Barisan simpul A, B, C, D, E, C, F adalah Trail bukan jalur karena c dua kali muncul.
- Barisan ruas a, d, g, k adalah jalur menghubungkan A dengan F
- Dan ruas a, b, h, g, e adalah Cycle. Graph yang tidak mengandung Cycle disebut Acyclic.

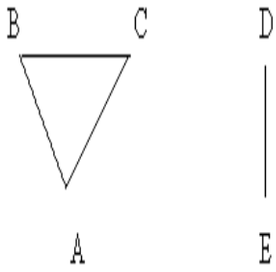
Contoh dari Graph Acyclic adalah pohon atau Tree. contoh dari pohon :



Suatu Graph G disebut terhubung jika untuk setiap 2 simpul dari Graph terdapat jalur yang menghubungkan 2 simpul tersebut.

Subgraph yang terhubung pada suatu Graph disebut komponen dari G bila Subgraph tersebut tidak terkandung dalam Subgraph terhubung lain yang lebih besar.

Contoh :



Terlihat misalnya antara D dan A
Tak ada jalur.

CONTOH PROGRAM :

```
/**
 *
 * @author Agung Setiawan
 */
import java.awt.Point;
public class segiempat {
int x1=0;
int y1=0;
int x2=0;
int y2=0;
public void buatsegiempat(int x1,int x2,int y1,int y2){
this.x1=x1;
this.y1=y1;
this.x2=x2;
```

```
this.y2=y2;
}
public void buatsegiempat(Point topLeft,Point bottomRight){
x1=topLeft.x;
y1=topLeft.y;
x2=bottomRight.x;
y2=bottomRight.y;
}
public void buatsegiempat(Point topLeft,int w,int h){
x1=topLeft.x;
y1=topLeft.y;
x2=(x1+w);
y2=(y1+h);
}
void cetaksegiempat(){
System.out.print("Segi empat <"+x1+","+y1);
System.out.print(","+x2+","+y2+">");
}
public static void main (String[]args){
segiempat rect=new segiempat();
System.out.print("Buat segi empat dengan koordinat (25,25) dan (50,50)");
rect.buatsegiempat (25,25,50,50);
rect.cetaksegiempat();
System.out.println();
System.out.println("Buat segi empat dengan titik (10,10) dan (20,20)");
rect.buatsegiempat(new Point(10,10), new Point(20,20));
rect.cetaksegiempat();
System.out.println();
System.out.print("Buat segi empat dengan 1 point (10,10), koordinat (50,50)");
rect.buatsegiempat(new Point (10,10),50,50);
rect.cetaksegiempat();

}}
```

BAB XI
MATRIKS PENYAJIAN GRAPH

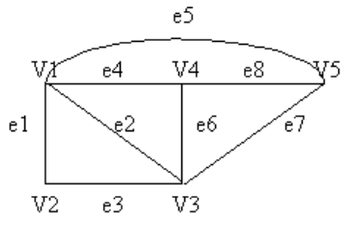
11.1. Matriks Penyajian Graph

Matriks Adjacency dari Graph G, yaitu Matriks yang menghubungkan Vertex dengan Vertex, tanpa ruas sejajar adalah Matriks A berukuran (N x N) yang bersifat : 1 , bila ada ruas (Vi, Vj) aij = 0.

Matriks Adjacency merupakan matriks simetri. Untuk Graph dengan ruas sejajar, Matriks Adjacency didefinisikan sebagai berikut : P bila ada p buah ruas menghubungkan (Vi, Vj)(p>0) aij = 0, bila dalam hal lain.

Matriks Incidence dari Graph G, yaitu Matriks yang menghubungkan Vertex dengan Edge, tanpa self-loop didefinisikan sebagai Matriks M berukuran (NXM) sebagai berikut : 1, bila ada ruas ej berujung di simpul Vi mij = 0.

Contoh :



Matriks Ruas

1	2
1	3
1	4
1	5
2	3
3	4
3	5
4	5

atau :

1	1	1	1	2	3	3	4
2	3	4	5	3	4	5	5

atau secara pasangan
{(1,2),(1,3),(1,4),(1,5),(2,3),(3,4),(3,5),(4,5)}

Matriks Adjacency

	V1	V2	V3	V4	V5
V1	0	1	1	1	1
V2	1	0	1	0	0
V3	1	1	0	1	1
V4	1	0	1	0	1
V5	1	0	1	1	0

Matriks Incidence

	e1	e2	e3	e4	e5	e6	e7	e8
V11	1	0	1	1	0	0	0	
V21	0	1	0	0	0	0	0	
V30	1	1	0	0	1	1	0	
V40	0	0	1	0	1	0	1	
V50	0	0	0	0	1	0	1	

11.2. Graph Terarah (Directed Graph / Digraph)

Graph terarah adalah Graph yang dapat menghubungkan V1 ke V2 saja (1 arah). Maksimum jumlah busur dari n simpul adalah : n (n - 1). Suatu Graph Berarah (Directed Graph) D terdiri atas 2 himpunan :

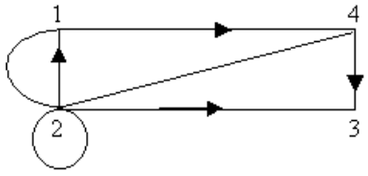
1. Himpunan V, anggotanya disebut simpul.
2. Himpunan A, merupakan himpunan pasangan terurut, yang disebut ruas berarah atau arkus.

Graph Berarah di atas ini, ditulis sebagai D(V,A).

Sebagai contoh, Gambar dibawah ini adalah sebuah Graph Berarah D(V,A), dengan:

1. V mengandung 4 simpul, yaitu 1, 2, 3 dan 4
2. A mengandung 7 arkus, yaitu (1,4) ,(2,1), (2,1), (4,2), (2,3), (4,3) dan (2)

Arkus (2,2) disebut gelung (self-loop), sedangkan arkus (2,1) muncul lebih dari satu kali, disebut arkus sejajar atau arkus berganda.

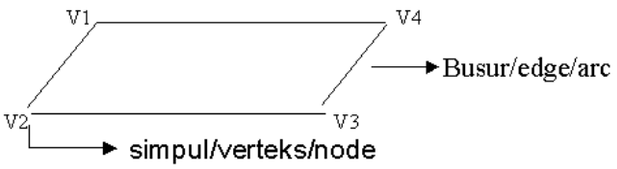


Apabila arkus/atau simpul suatu Graph Berarah menyatakan suatu bobot, maka Graph Berarah tersebut dinamakan suatu jaringan atau Network. Graph semacam itu biasanya digunakan menggambarkan situasi dinamis.

Bila V' himpunan bagian dari V serta A' himpunan bagian dari A, dengan titik ujung anggota A' terletak di dalam V', maka dikatakan bahwa D'(V',A') adalah Graph bagian (Subgraph) dari D(V,A). kalau A' mengandung semua arkus anggota A yang titik ujungnya anggota V', maka dikatakan bahwa D'(V',A') adalah Graph Bagian yang dibentuk atau direntang oleh V'.

GRAPH TAK TERARAH (UNDIRECTED GRAPH)

Graph Tak Terarah adalah Graph yang menghubungkan 2 verteks V1 ke V2 dan V2 ke V1 (2 arah). Bila Verteks = n, maka Graph tak terarah komplit akan mempunyai busur edge sama dengan : n (n - 1) / 2

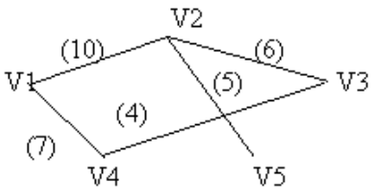


Yang dapat dilakukan adalah : V1 - V2 = V2 - V1
V1 - V4 = V4 - V1
dan seterusnya.

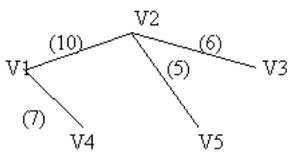
11.3. Minimum Spanning Tree

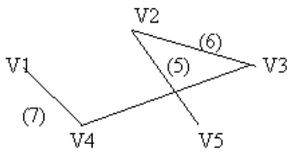
Adalah Spanning Tree yang mempunyai Bobot dan tidak mempunayi arah dengan hasil penjumlahan bobotnya adalah minimum. Lihat dari gambar Graph G diatas.

G ' = Subgraph dari G



G" = Subgraph dari G'
Membentuk Struktur pohon Spanning dari G





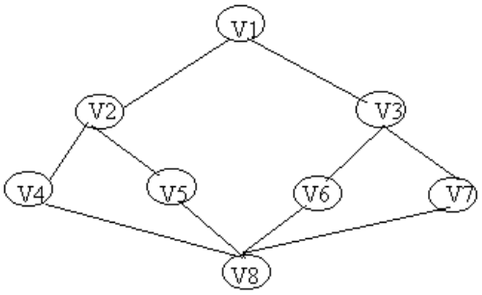
MST = 21

11.4. Penelusuran Graph

Dapat dilakukan dengna dua cara, yaitu :

1. Depth First Search (DFS)

   Penelusuran dengan DFS pada Graph Tak Berarah dengan melakukan pengecekan pada node dengan kedalaman pertama dri node yang ditinjau



VERTEKS

V1		---->	2		---->	3	0			
V2		---->	1		---->	4		---->	5	0
V3		---->	1		---->	6			7	0
V4		---->	2		---->	8	0			
V5		---->	2		---->	8	0			
V6		---->	3		---->	8	0			
V7		---->	3		---->	8	0			
V8		---->	4		---->	5		---->	7	0

Dari gambar di atas akan diperoleh urutan :
V1 ---> V2 ---> V4 ----> V8 ----> V5, V6 ---> V3 ---> V7

2. Breadth First Search (BFS)

   Berbeda dengan cara BFS, dengan BFS penelusuran akan diawasi dari Node-1, kemudian melebar pada Adjacent Node dari Node-1 dan diteruskan pada Node-2, Node- 3 dan seterusnya.

   Dari gambar di atas akan diperoleh urutan :

   V1 , V2 ---> V3 , V4 ---> V5 ---> V6 ---> V7, V8

DAFTAR PUSTAKA

Bambang Haryanto, Ir. MT (2003). “Struktur Data Memuat Dasar Pengembangan Berorientasi Objek”. Informatika. Bandung.

Bambang Haryanto, Dr (2011). “Esensi-esensi Bahasa Pemrograman Java”. Informatika. Bandung.

G. Sri Hartati, B. Herry Suharto, M. Soesilo Wijono (2007). “Pemrograman GUI Swing Java”. Andi Offset. Yogyakarta.