



## MODUL PERKULIAHAN

# Teori Bahasa Otomata

## Contoh Penerapan Otomata Pada Algoritma Tree.

Fakultas  
Teknik

Program Studi  
Program  
Studi Informatika

Tatap Muka  
**12-13**

Kode MK  
**190641006 - [e]**  
Teori Bahasa &  
Otomata

Disusun Oleh

- Sukenda, S.Kom., M.T.
- Ari Purno Wahyu Wibowo, S.Kom., M.Kom.

### Abstract

Pada teknik Automata mahasiswa diajarkan untuk memahi dan mengerti pengguna variable dan komponen pemograman lainnya serta menurunkannya pada bahasa pemograman atau dengan kata lain mampu membuat sebuah rekayasa program berbasis matematika. Kemampuan lain yang diperlukan adalah mahasiswa bisa mendiskripsikan kebutuhan sistem dalam membuat sebuah program atau produk serta mampu membuat sebuah dokumentasi hasil rekayasa sistem.

### Kompetensi

Mahasiswa memahami pengertian dan kedudukan Teori Bahasa dan Otomata (TBO) pada ilmu komputer, mampu menjelaskan sebuah kebutuhan dengan mendeskripsikan cara kerja sistem tersebut dalam bentuk simbol dan contoh produk tepat guna serta dapat mendefinisikan langkah-langkah algoritmik dalam pemrosesan tata bahasa.

## Analisa cara kerja alat dengan teknik automata

Suatu komputer dapat melaksanakan sesuatu bila diberikan sederetan perintah / instruksi yang dimengerti yang diurut secara logika. Deretan perintah ini disebut “program”. Sekumpulan aturan-aturan dalam membuat program disebut sebagai bahasa pemrograman (*Programming Language*). Karena komputer itu adalah suatu mesin, maka insruksi dan bahasa yang dimengerti adalah juga instruksi dan bahasa mesin. Bahasa mesin pada dasarnya hanya mengandung dua simbol yaitu simbol biner 0 dan 1 sehingga sangat sulit bagi manusia membuat program untuk komputer dalam bahasa mesin, terlebih lagi karena setiap jenis komputer mempunyai bahasa mesin sendiri yang berbeda dari satu komputer ke komputer lain.

Untuk memudahkan manusia membuat program komputer, telah diciptakan bermacam-macam bahasa pemrograman. Secara garis besar, bahasa pemrograman dapat dibagi menjadi dua bagian besar yaitu,

1. Bahasa Pemrograman Tingkat Rendah (*Low Level Language*), contoh bahasa pemrograman Assembly.
2. Bahasa Pemrograman Tingkat Tinggi (*High Level Language*), contoh bahasa pemrograman FORTRAN, COBOL, BASIC, Pascal, RPG, C dan sebagainya.

Kelemahan semua bahasa pemrograman ini adalah membutuhkan proses penerjemah dan sarana penerjemah berupa *Assembler*, *Compiler* atau *Interpreter*. Selain itu, program yang dibuat dengan menggunakan bahasa pemrograman tingkat

tinggi pada umumnya tidak dapat menjangkau keseluruhan bagian komputer dimana program tersebut dilaksanakan. Program yang dapat menjangkau dan memanfaatkan seluruh kemampuan komputer hanyalah yang dibuat dalam bahasa mesin atau bahasa Assembly.

Tujuan dari pembuatan bahasa pemrograman tingkat tinggi pada awalnya hanya ditujukan untuk menyelesaikan ekspresi aritmatika. Sebuah bahasa pemrograman dikatakan baik apabila mampu untuk memberikan keleluasaan kepada para *programmer* untuk menuliskan ekspresi aritmatika dengan ketentuan yang hampir sama seperti penulisan matematika secara manual. Sebuah *compiler* dikatakan berkompeten apabila mampu untuk membaca ekspresi – ekspresi berikut ini,

$$(x + y) * \exp(x - z) - 4.0$$

$$a * b + c / d - c * (x + y)$$

$$\text{not}(p \text{ and } q) \text{ or } (x \leq 7.0)$$

Pada tahun 1959 seorang ahli bernama *Noam Chomsky* melakukan penggolongan tingkatan tata bahasa menjadi empat, yang disebut dengan *Hirarki Chomsky*. Penggolongan tersebut bisa dilihat pada tabel berikut,

*Tabel 2.1 Hirarki Chomsky*

Bahasa	Mesin Automata	Batasan Aturan Produksi
<i>Regular / Tipe 3</i>	<i>Finite State Automata (FSA)</i> meliputi <i>Deterministic Finite Automata (DFA)</i> dan <i>Non deterministic Finite Automata (NFA)</i>	$\alpha$ (ruas kiri) adalah sebuah simbol non terminal $\beta$ (ruas kanan) maksimal hanya memiliki sebuah simbol non terminal yang bila ada terletak di posisi paling kanan
Bebas Konteks / <i>Context Free / Tipe 2</i>	<i>Push Down Automata (PDA)</i>	$\alpha$ berupa sebuah simbol non terminal

<i>Context Sensitive / Tipe 1</i>	<i>Linier Bounded Automata</i>	$ \alpha  \leq  \beta $
<i>Unrestricted / Phase Structure / Natural Language / Tipe 0</i>	Mesin Turing	Tidak ada Batasan

Semua aturan produksi dinyatakan dalam bentuk  $\alpha \rightarrow \beta$  yang bisa dibaca  $\alpha$  menghasilkan  $\beta$ , atau  $\alpha$  menurunkan  $\beta$ .  $\alpha$  menyatakan simbol – simbol pada ruas kiri aturan produksi (sebelah kiri tanda ' $\rightarrow$ ') dan  $\beta$  menyatakan simbol – simbol pada ruas kanan aturan produksi atau bisa disebut juga hasil produksi. Simbol – simbol tersebut bisa berupa simbol terminal atau simbol nonterminal. Simbol nonterminal adalah simbol yang masih bisa diturunkan, sedang simbol terminal sudah tidak bisa diturunkan lagi. Untuk contoh berikut simbol terminal akan dinyatakan dengan huruf kecil, seperti 'a', 'b', 'c'. Simbol nonterminal dinyatakan dengan huruf besar, seperti 'A', 'B', dan 'C'.

Dengan menerapkan aturan produksi, suatu tata bahasa bisa menghasilkan sejumlah *string*. Himpunan semua *string* tersebut adalah bahasa yang didefinisikan oleh tata bahasa tersebut. Contoh aturan produksi,

$$T \rightarrow a$$

Bisa dibaca T menghasilkan a, dimana T merupakan simbol nonterminal, dan a merupakan simbol terminal.

$$E \rightarrow T \mid T + E$$

Bisa dibaca E menghasilkan T atau E menghasilkan T + E, dimana E dan T merupakan simbol nonterminal, dan + merupakan simbol terminal.

Simbol ‘|’ bertujuan untuk menyatakan *atau*, yang biasa digunakan untuk mempersingkat penulisan aturan produksi yang mempunyai ruas kiri yang sama. Pada contoh sebelumnya,

$$E \rightarrow T \mid T + E$$

merupakan penyederhanaan dari aturan produksi berikut,

$$E \rightarrow T$$

$$E \rightarrow T + E$$

Bahasa manusia / bahasa alami termasuk ke dalam tata bahasa (*grammar*) tipe 0 / *unrestricted*, dimana tidak ada batasan pada aturan produksinya. Misalkan saja,

$$Abc \rightarrow De$$

Pada bahasa *context sensitive*, panjang *string* pada ruas kiri  $\leq$  panjang ruas kanan (  $|\alpha| \leq |\beta|$  ). Contoh aturan produksi yang *context sensitive*,

$$Ab \rightarrow DeF$$

$$CD \rightarrow eF$$

Perhatikan aturan produksi seperti,

$$S \rightarrow \emptyset$$

Kita ketahui  $|S| = 1$ , sedang  $|\emptyset| = 0$ , menurut aturan *context sensitive* aturan produksi diatas tidak diperkenankan, tetapi di sini kita buat suatu pengecualian, sehingga  $S \rightarrow \emptyset$  dianggap memenuhi *context sensitive grammar*.

Batasan *context sensitive* biasanya turut digunakan dalam proses analisis semantik pada tahapan translasi.

Pada bahasa bebas konteks, batasannya bertambah lagi dengan ruas kiri haruslah tepat satu simbol nonterminal. Misalnya,

$$B \rightarrow CDeFg$$

$$D \rightarrow BcDe$$

Bahasa bebas konteks menjadi dasar dalam pembentukan suatu *parser* / analisis sintaks dalam translasi. Bagian sintaks dari suatu *translator* kebanyakan didefinisikan dalam tata bahasa bebas konteks (*context free grammar*), yang dideskripsikan secara formal dengan notasi *BNF* (*Backus Naur Form*).

Pada bahasa regular, batasannya bertambah dengan ruas kanan maksimal memiliki sebuah simbol nonterminal yang terletak di paling kanan. Artinya bisa memiliki simbol terminal saja dalam jumlah tidak dibatasi, tetapi bila terdapat simbol nonterminal, maka simbol nonterminal tersebut hanya berjumlah satu dan terletak di posisi paling kanan. Misalnya,

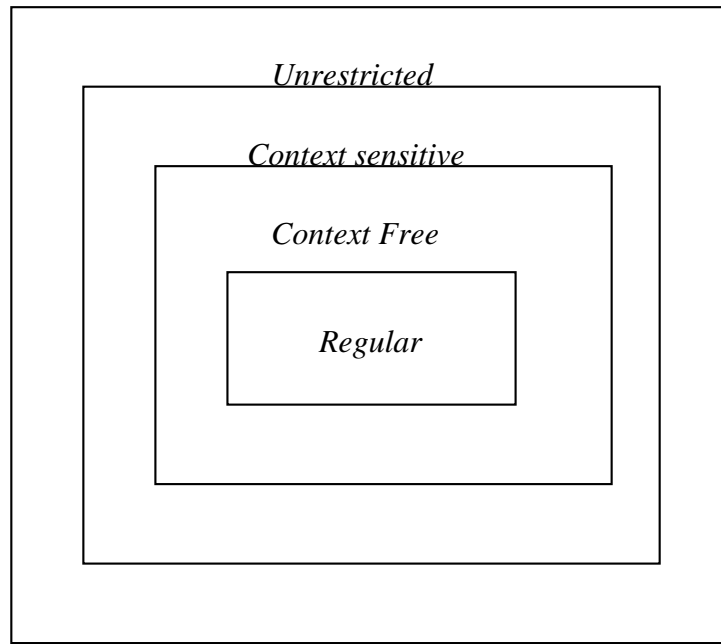
$$A \rightarrow e$$

$$A \rightarrow efg$$

$$A \rightarrow efgH$$

$$C \rightarrow D$$

Bisa kita lihat, batasan bahasa makin bertambah dari tipe 0 sampai dengan tipe 3. Berdasarkan keterkaitannya antar tipe bahasa bisa dilihat secara sederhana pada gambar berikut,



**Gambar 2.1 Keterkaitan bahasa pada hirarki chomsky**

Ada beberapa hal yang harus diperhatikan dalam aturan produksi, seperti contoh berikut,

$$\emptyset \rightarrow Abd$$

Bukan aturan produksi yang legal, karena simbol  $\emptyset$  tidak boleh berada pada ruas kiri. Sedangkan aturan produksi yang ruas kirinya hanya memuat simbol terminal saja, seperti,

$$a \rightarrow bd$$

$$ab \rightarrow bd$$

Bukan aturan produksi yang legal (untuk *unrestricted grammar* sekalipun), karena ruas kiri harus juga memuat simbol yang bisa diturunkan, sementara pada contoh ruas kiri hanya terdiri dari simbol terminal saja padahal sesuai dengan definisinya simbol terminal sudah tidak bisa diturunkan lagi, berbeda dengan aturan produksi seperti,  $aA \rightarrow bd$ .

Sebuah ekspresi aritmatika terdiri dari operand dan operator. Operator dalam ekspresi aritmatika dapat dibagi menjadi 2 jenis, yaitu:

- *Binary operator* (operator pasangan)
- *Unary operator* (operator tunggal)

*Binary operator* adalah operator yang memiliki 2 buah operand (diapit oleh 2 buah operand), sedangkan *unary operator* adalah operator yang hanya memiliki 1 buah operand (diikuti oleh sebuah operand). Operator – operator yang termasuk dalam *binary operator* adalah operator penjumlahan (+), pengurangan (-), perkalian (\*), pembagian (/), modulo (mod), divisor (div), pemangkatan (^), operator logika *AND*, operator logika *OR*, dan operator perbandingan (seperti operator lebih besar, lebih kecil, sama dengan, lebih besar sama dengan, lebih kecil sama dengan, dan tidak sama dengan). Sedangkan operator yang termasuk dalam *unary operator* adalah operator minus (~), operator faktorial (!), operator trigonometri (seperti operator *sinus*, *cosinus*, *tangen*, *cotangen*, *secan*, dan *cosecan*), operator logika *NOT*, operator *exponential* (exp) dan fungsi logaritma (log).

Prioritas / kedudukan dari masing – masing operator (baik *unary operator* maupun *binary operator*) dari tinggi ke rendah adalah sebagai berikut,

3. Operator pemangkatan (^) dan semua *unary operator*.
4. Operator perkalian (\*), pembagian (/), modulo (mod) dan divisor (div).
5. Operator penjumlahan (+) dan pengurangan (-).
6. Operator perbandingan, yaitu operator lebih besar, lebih kecil, sama dengan, lebih besar sama dengan, lebih kecil sama dengan, dan tidak sama dengan.
7. Operator logika *NOT*.



8. Operator logika *AND* dan *OR*.

9. *Assignment Operator* (=).

Ekspresi aritmatika akan diselesaikan berdasarkan urutan prioritas dari operator di atas dengan ketentuan operator yang memiliki prioritas yang lebih tinggi akan diselesaikan terlebih dahulu. Tahapan – tahapan penyelesaian suatu ekspresi aritmatika dapat direpresentasikan dalam bentuk *graph* yang dinamakan pohon ekspresi (*expression tree*).

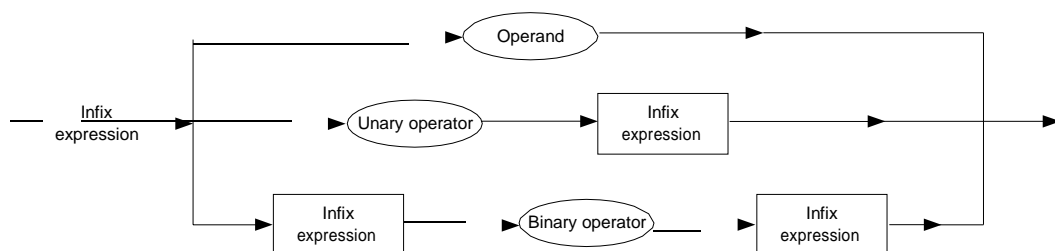
### Notasi / Penulisan Ekspresi Aritmatika

*Polish Notation* diperkenalkan oleh seorang ahli matematika Polandia bernama Jan Lukasiewicz. *Polish Notation* merupakan notasi penulisan ekspresi aritmatika. *Polish Notation* terdiri dari 3 bentuk, yaitu:

1. *Prefix*
2. *Suffix (Postfix)*
3. *Infix*

### Infix

Bentuk *infix* merupakan bentuk penulisan normal dari ekspresi aritmatika. Suatu *infix* dapat berupa operand tunggal, atau gabungan dari *unary operator* dengan *infix*, ataupun berupa gabungan dari *binary operator* dengan dua buah *infix*. Diagram bentuk *infix* dapat dilihat pada gambar di bawah ini.

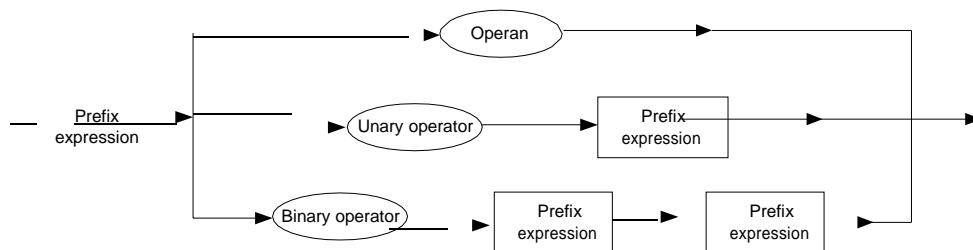


## Gambar 2.2 Diagram bentuk infix

Bentuk *infix* dari ekspresi aritmatika tersebut tidak mengubah bentuk penulisan ekspresi dan hanya melakukan pembuangan spasi-spasi yang berlebihan saja. Sebagai contoh, misalkan diketahui suatu ekspresi aritmatika  $x * y + 2 * (z - 3)$ , maka bentuk *infix*-nya berupa  $x*y+2*(z-3)$ .

### 2.2.1.1 Prefix

Bentuk *prefix* merupakan cara / bentuk penulisan ekspresi aritmatika dimana operator ditulis di depan dari operandnya. Suatu *prefix* dapat berupa operand tunggal, atau gabungan dari *unary operator* dengan *prefix*, ataupun berupa gabungan dari *binary operator* dengan dua buah *prefix*. Diagram bentuk *prefix* dapat dilihat pada gambar di bawah ini.



## Gambar 2.3 Diagram bentuk prefix

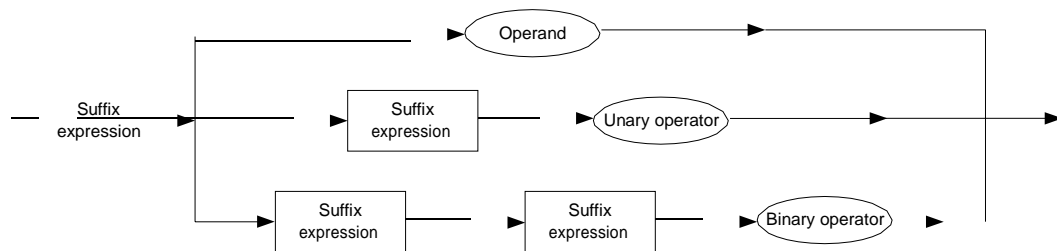
Sebagai contoh, misalkan diketahui suatu ekspresi aritmatika  $x * y + 2 * (z - 3)$ , maka bentuk *prefix*-nya berupa:  $+ * x y * 2 - z 3$ .

Proses pengubahan ekspresi aritmatika  $x * y + 2 * (z - 3)$  ke dalam bentuk *prefix* dilakukan berdasarkan urutan prioritas dari operator – operator yang terdapat di dalam ekspresi aritmatika tersebut. Proses dimulai dengan mengubah sub ekspresi  $(z - 3)$  menjadi  $- z 3$  sehingga ekspresi aritmatika tersebut menjadi  $x * y + 2 * - z 3$ . Proses dilanjutkan dengan mengubah sub ekspresi  $x * y$  menjadi  $* x y$  sehingga ekspresi aritmatika menjadi  $* x y + 2 * - z 3$  dan dilanjutkan dengan

mengubah bentuk  $2 * - z 3$  menjadi  $* 2 - z 3$  sehingga ekspresi aritmatika menjadi  $* x y + * 2 - z 3$ . Proses diakhiri dengan mengubah ekspresi aritmatika menjadi bentuk prefix  $+ * x y * 2 - z 3$ .

### Suffix (Postfix)

Bentuk *suffix (postfix)* merupakan cara / bentuk penulisan ekspresi aritmatika dimana operator ditulis di belakang dari operandnya. Suatu *suffix* dapat berupa operand tunggal, atau gabungan dari *suffix* dengan *unary operator*, ataupun berupa gabungan dari dua buah *suffix* dengan *binary operator*. Diagram bentuk *suffix* dapat dilihat pada gambar di bawah ini.



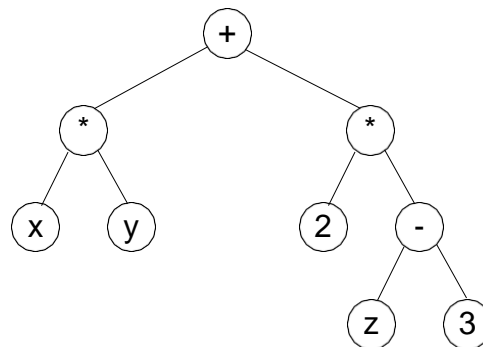
**Gambar 2.4 Diagram bentuk suffix**

Sebagai contoh, misalkan diketahui suatu ekspresi aritmatika  $x * y + 2 * (z - 3)$ , maka bentuk *suffix*-nya berupa:  $x y * 2 z 3 - * +$ . Proses pengubahan ekspresi aritmatika  $x * y + 2 * (z - 3)$  ke dalam bentuk *suffix (postfix)* juga dilakukan berdasarkan urutan prioritas dari operator – operator yang terdapat di dalam ekspresi aritmatika tersebut. Proses dimulai dengan mengubah sub ekspresi  $(z - 3)$  menjadi  $z 3 -$  sehingga ekspresi aritmatika tersebut menjadi  $x * y + 2 * z 3 -$ . Proses dilanjutkan dengan mengubah sub ekspresi  $x * y$  menjadi  $x y *$  sehingga ekspresi aritmatika menjadi  $x y * + 2 * z 3 -$  dan dilanjutkan dengan mengubah bentuk  $2 * z 3 -$  menjadi  $2 z 3 - *$  sehingga ekspresi aritmatika

menjadi  $x y * + 2 z 3 - *$ . Proses diakhiri dengan mengubah ekspresi aritmatika menjadi bentuk *suffix*  $x y * 2 z 3 - * +$ .

### Pohon Ekspresi (Expression Tree)

Pohon ekspresi (*expression tree*) adalah sebuah pohon biner (*binary tree*) dimana daun berisi operand yang terdapat dalam ekspresi aritmatika dan akar berisi operator yang terdapat dalam ekspresi aritmatika tersebut. Proses pembacaan dari pohon ekspresi dimulai dari daun paling kiri hingga akar utama. Operand dan operator yang berada pada level bawah akan dibaca terlebih dahulu. Sebagai contoh, misalkan diketahui sebuah ekspresi matematika  $x * y + 2 * (z - 3)$ , maka pohon ekspresinya adalah sebagai berikut,



**Gambar 2.5 Pohon ekspresi untuk ekspresi aritmatika  $x * y + 2 * (z - 3)$**

Sesuai dengan struktur pohon ekspresi di atas, maka proses penyelesaian ekspresi aritmatika  $x * y + 2 * (z - 3)$  dibagi menjadi 2 bagian, yaitu sub ekspresi di sebelah kiri operator penjumlahan (+) dan sub ekspresi di sebelah kanan operator penjumlahan (+). *Left subtree* dari pohon ekspresi menggambarkan proses penyelesaian sub ekspresi di sebelah kiri operator penjumlahan (+), dan *right subtree* menggambarkan proses penyelesaian sub ekspresi di sebelah kanan operator penjumlahan (+). Operator dan operand yang berada pada kedudukan

(*level*) yang lebih bawah pada pohon akan dikerjakan terlebih dahulu. Proses pembacaan pada *left subtree* akan membaca sub ekspresi  $(x * y)$ . Sedangkan proses pembacaan pada *right subtree* akan dimulai dengan membaca sub ekspresi  $(z - 3)$  terlebih dahulu dan dilanjutkan dengan pembacaan sub ekspresi  $2 * (z - 3)$ . Proses pembacaan akan diakhiri dengan menggabungkan hasil proses pembacaan pada *left subtree* dan *right subtree*, sehingga didapat ekspresi aritmatika  $x * y + 2 * (z - 3)$ .

### Token, Pattern dan Lexemes

Bentuk “*token*”, “*pattern*” dan “*lexemes*” sering digunakan dalam analisis leksikal. Secara umum, terdapat sebuah kumpulan (set) dari *string* dalam *input* di mana token yang sama diproduksi sebagai *output*. Set dari *string* ini dideskripsikan oleh sebuah aturan yang disebut *pattern* diasosiasikan dengan token. *Pattern* digunakan untuk mencocokkan setiap *string* dalam set. *Lexemes* adalah sebuah sekuens (urutan) dari karakter dalam *source program* yang dicocokkan oleh *pattern* untuk sebuah token. Bentuk *token*, *pattern* dan *lexemes* yang akan digunakan dideskripsikan sebagai berikut:

Tabel 1.1 Token, pattern dan lexemes dari ekspresi aritmatika

Token	Pattern	Lexemes
Digit	Angka = $[0..9]$	8
	Angka <sup>3</sup>	251
Variabel	Huruf = $[a..z]$	a
	Huruf <sup>3</sup>	ad
	$(Huruf).(Huruf \mid Angka)^{0..2}$	a20

+	+	+
-	-	-
~	~	~
*	*	*
/	/	/
\	\	\
%	%	%
(	(	(
)	)	)

## Struktur Data

Struktur berarti susunan / jenjang, dan data berarti sesuatu simbol / huruf / lambang angka yang menyatakan sesuatu. Struktur data berarti susunan dari simbol / huruf / lambang angka untuk menyatakan sesuatu hal. Sebagai contoh, struktur program Pascal dapat didefinisikan seperti berikut,

- Judul Program
- Bagian Deklarasi / Blok Program, yang terdiri dari,
  - Deklarasi Tipe Data
  - Deklarasi Konstanta
  - Deklarasi Variabel
  - Deklarasi Nama Fungsi / Prosedur
  - Deklarasi Label

Gabungan dari algoritma dan struktur data akan membentuk suatu program.

Adapun manfaat dari struktur data adalah sebagai berikut,

- Mengefisiensikan program.

Program yang dibuat dengan menerapkan konsep – konsep yang berlaku pada struktur data akan lebih efisien dibandingkan dengan program yang dibuat dengan mengabaikan konsep struktur data.

- Modifikasi

Sesuatu program harus dapat dimodifikasi apabila diperlukan, hal ini dapat dilakukan jika fasilitas yang diperlukan dibuat (disertakan) walaupun pada tahap awal belum dipakai.

- Memilih metode yang tepat

Misalkan suatu *plaza* pada hari – hari tertentu mengalami antrian yang panjang pada kasir, hal ini dapat diatasi dengan metode,

- Pemasukan data tidak melalui *keyboard* lagi, melainkan melalui *barcode*.
- Membuat pemberitahuan pada kasir – kasir.

## Tree

Pohon (*tree*) merupakan struktur data nonlinier yang banyak digunakan dalam aplikasi sehari-hari. Contoh aplikasi pohon yang dapat kita lihat sehari-hari adalah pengelolaan file dalam direktori penyimpanan. Pohon merupakan struktur data yang memiliki suatu struktur hirarki pada sekumpulan elemen, dan memiliki hubungan satu ke banyak (*one to many relationship*) seperti yang kita lihat dalam struktur organisasi sebuah perusahaan atau daftar isi sebuah buku. Dalam struktur organisasi kita dapat melihat bahwa ada level atas biasanya hanya ada satu pimpinan tertinggi. Pada level berikutnya diisi oleh beberapa orang dengan jabatan yang berbeda tetapi dalam tingkatan yang sama. Selanjutnya dapat dipecah lagi ke level berikutnya sampai struktur dapat memenuhi fungsi dan tujuan organisasi.

Biasanya satu atasan memiliki beberapa bawahan yang berada dalam ruang lingkup wewenang dan tugas atasan. Begitu juga dalam daftar isi buku, dimana satu buku terdiri dari beberapa bab dan setiap terdiri dari beberapa sub bab, satu sub bab terdiri dari beberapa sub sub bab dan seterusnya. Dengan demikian hirarki dapat kita anggap sebagai “terdiri dari” atau “bawahan” atau “diawasi” dari atas ke bawah. Salah satu keuntungan pohon dibandingkan dengan struktur data linier adalah waktu cari sebuah node maksimum (dapat) lebih kecil dari  $n$  jika jumlah data =  $n$ .

Sebuah *tree* dapat mempunyai hanya sebuah simpul tanpa sebuah sisi pun. Dengan kata lain, jika  $G = (V, E)$  adalah *tree*, maka  $V$  tidak boleh berupa himpunan kosong, namun  $E$  boleh kosong. *Tree* juga seringkali didefinisikan sebagai graf tak-berarah dengan sifat bahwa hanya terdapat sebuah lintasan unik antara setiap pasang simpul. Selain itu, di dalam *tree* jumlah sisinya adalah jumlah simpul dikurangi satu.

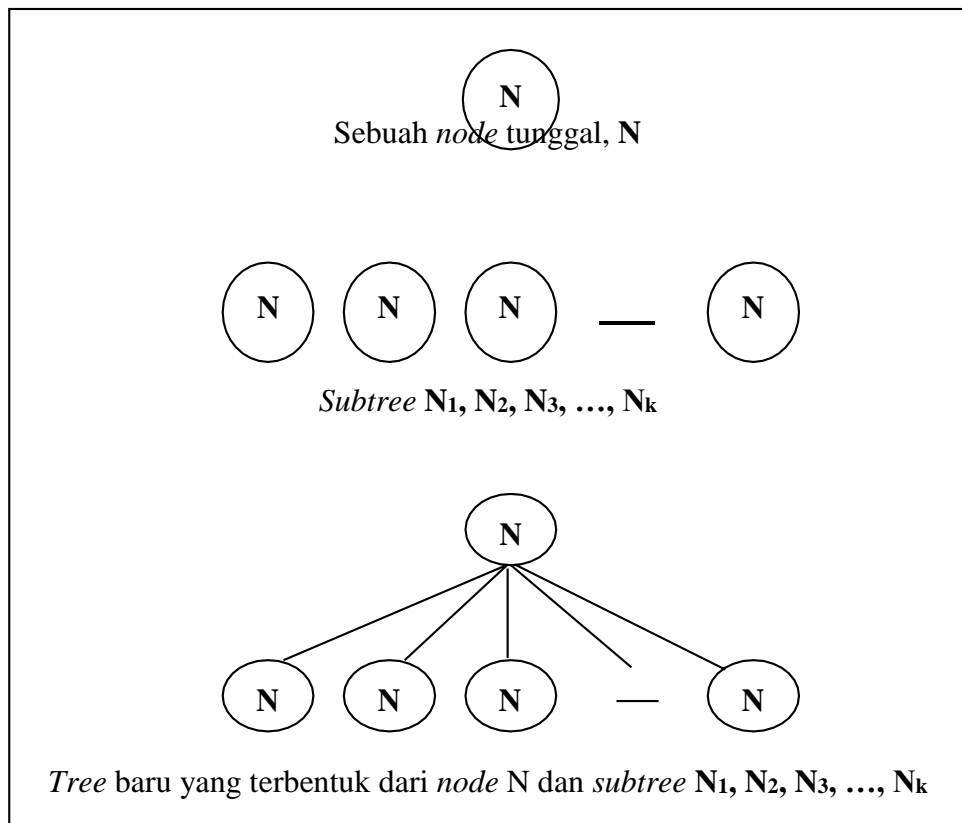
Secara sederhana, sebuah *tree* bisa didefinisikan sebagai kumpulan dari elemen – elemen yang disebut dengan *node* / *vertex* (simpul) dimana salah satu *node* disebut dengan *root* (akar), dan sisa *node* lain terpecah menjadi himpunan yang saling tidak berhubungan satu sama lain dan disebut dengan *subtree* (pohon bagian). Jika dilihat pada setiap *subtree* maka *subtree* juga mempunyai *root* dari *subtree*-nya masing – masing.

Dengan melihat istilah dasar di atas, maka sebuah *tree* secara rekursif dapat didefinisikan sebagai berikut:

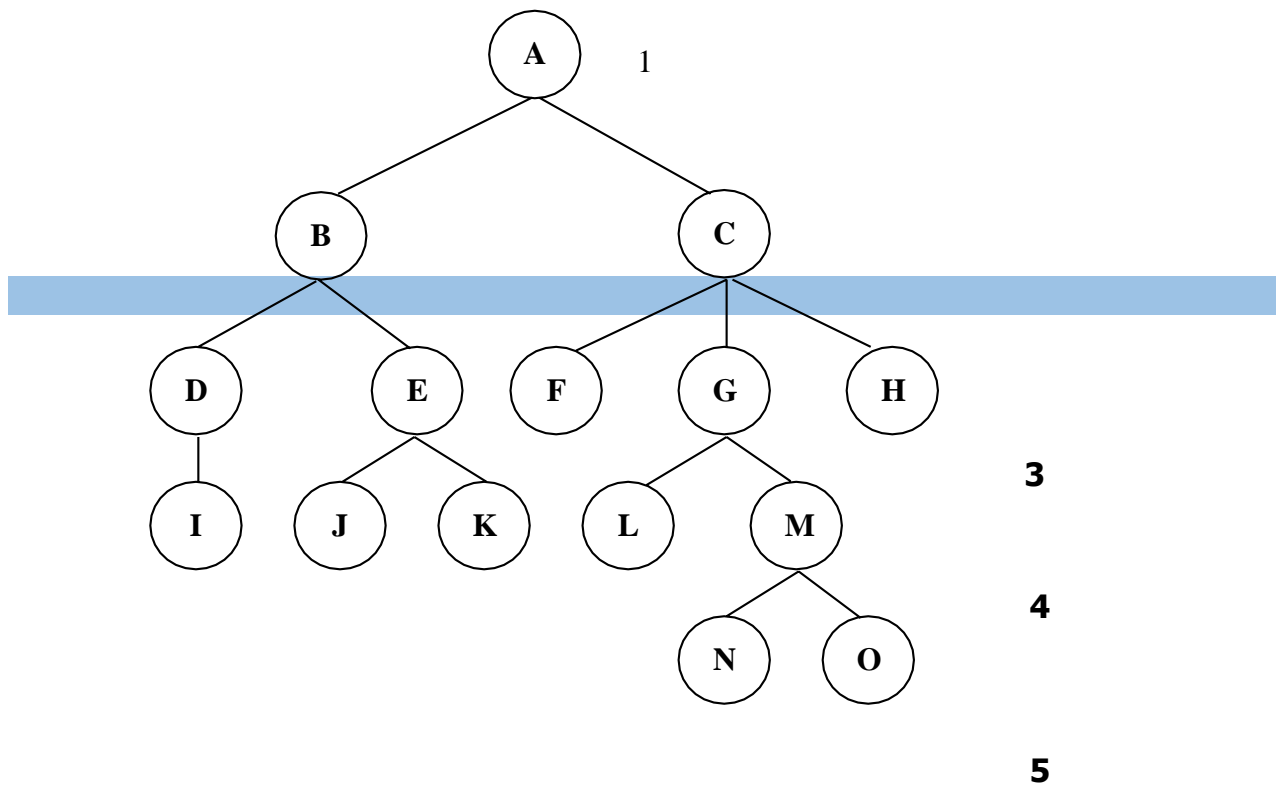
1. Sebuah *node* tunggal adalah sebuah *tree*.



2. Jika terdapat sebuah *node*  $N$  dan beberapa *subtree*  $N_1, N_2, N_3, \dots, N_k$  maka dari *node*  $N$  dan *subtree* yang ada dapat dibentuk sebuah *tree* yang mempunyai *root* pada *node*  $N$ .



**Gambar 2.6 Contoh pembentukan Tree**



**Gambar 2.7 Contoh tree dengan 15 node**

Seperti yang terlihat pada gambar 2.7 di atas, sebenarnya yang disebut dengan *node* itu adalah bagian dari *tree* yang berisikan data / informasi dan penunjuk percabangan. *Tree* pada gambar 2.7 berisi 15 *node* yang berisikan informasi berupa huruf A, B, C, D hingga huruf O lengkap dengan percabangannya masing – masing, dimana *tree* ini mempunyai *root* pada *node* A.

Hubungan antara satu *node* dengan *node* lain bisa dianalogikan seperti halnya dalam sebuah keluarga, yaitu ada anak, bapak, saudara, dan lain – lain. Dalam gambar 2.7 *node* A adalah bapak dari *node* B dan C, dengan demikian *node* B dan C ini bersaudara. *Node* D dan E adalah anak dari *node* B. *Node* C adalah paman dari *node* D dan E.

Level (tingkatan) suatu *node* ditentukan dengan pertama kali menentukan *root* sebagai tingkat pertama. Jika suatu *node* dinyatakan sebagai tingkat N, maka *node* yang merupakan anaknya dikatakan berada dalam tingkat N + 1. Gambar 2.7 menunjukkan contoh *tree* lengkap dengan *level* pada setiap *node*. Di samping definisi di atas, ada juga beberapa buku yang menyebutkan bahwa *root* dinyatakan

sebagai *level* 0 dan *node* lain dinyatakan mempunyai *level* 1 tingkat lebih tinggi dari *root*.

Selain *level*, juga dikenal istilah *degree* (derajat) dari suatu *node*. *Degree* suatu *node* dinyatakan sebagai banyaknya anak atau turunan dari *node* tersebut. Sebagai contoh dalam gambar 2.7 *node* A mempunyai *degree* 2, *node* B mempunyai *degree* 2, dan *node* C mempunyai *degree* 3. *Node* lain F, H, I, J, K, L, N, dan O yang semuanya mempunyai *degree* 0 disebut juga dengan *leaves* (daun).

*Leaf* dan *root* tergolong *external node* (simpul luar), sedangkan selain *node* tersebut di atas disebut dengan *internal node* (simpul dalam).

*Height / depth* (ketinggian / kedalaman) dari suatu *tree* adalah tingkat maksimum dari suatu *node* dalam *tree* tersebut dikurangi dengan 1. Dengan demikian *tree* pada gambar 2.7 yang mempunyai *root* pada *node* A mempunyai *height* 4.

*Ancestor* (leluhur) dari suatu *node* adalah semua *node* yang terletak dalam suatu jalur dengan *node* tersebut mulai dari *root* sampai *node* yang ditinjau. Sebagai contoh *ancestor* dari *node* L gambar 2.15 adalah *node* A, C, dan G.

*Descendant* (keturunan) dari suatu *node* adalah semua *node* yang terletak tepat di bawah *node* tersebut. Sebagai contoh *descendant* dari *node* G pada gambar 2.7 adalah *node* L, M, N, dan O.

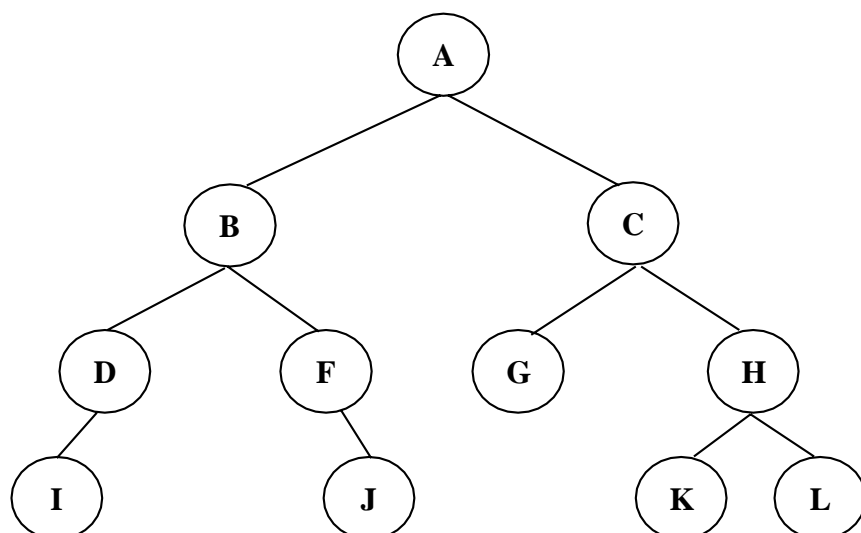
*Predecessor* dari suatu *node* adalah semua *node* yang berada pada *level* di atas dari *level node* tersebut. Sebagai contoh *predecessor* dari *node* G pada gambar 2.7 adalah *node* A, B, dan C.

*Successor* dari suatu *node* adalah semua *node* yang berada pada *level* di bawah dari *level node* tersebut. Sebagai contoh *successor* dari *node* G pada gambar 2.7 adalah *node* I, J, K, L, M, N, dan O. *Sibling* dari suatu *node* adalah semua *node* yang berada pada *level* yang sama yang berasal dari *node* asal (satu *level* di atas *node* yang ditinjau) yang sama dengan *node* tersebut. Sebagai contoh *sibling* dari *node* G pada gambar 2.7 adalah *node* F dan H.

## Binary Tree

*Binary Tree* (pohon biner) didefinisikan sebagai suatu kumpulan *node* yang mungkin kosong atau mempunyai *root* dan paling banyak dua *subtree* (anak) yang saling terpisah yang disebut dengan *left subtree* (pohon bagian kiri / anak kiri / cabang kiri) dan *right subtree* (pohon bagian kanan / anak kanan / cabang kanan). *Subtree* bisa disebut juga dengan istilah *branch* (cabang).

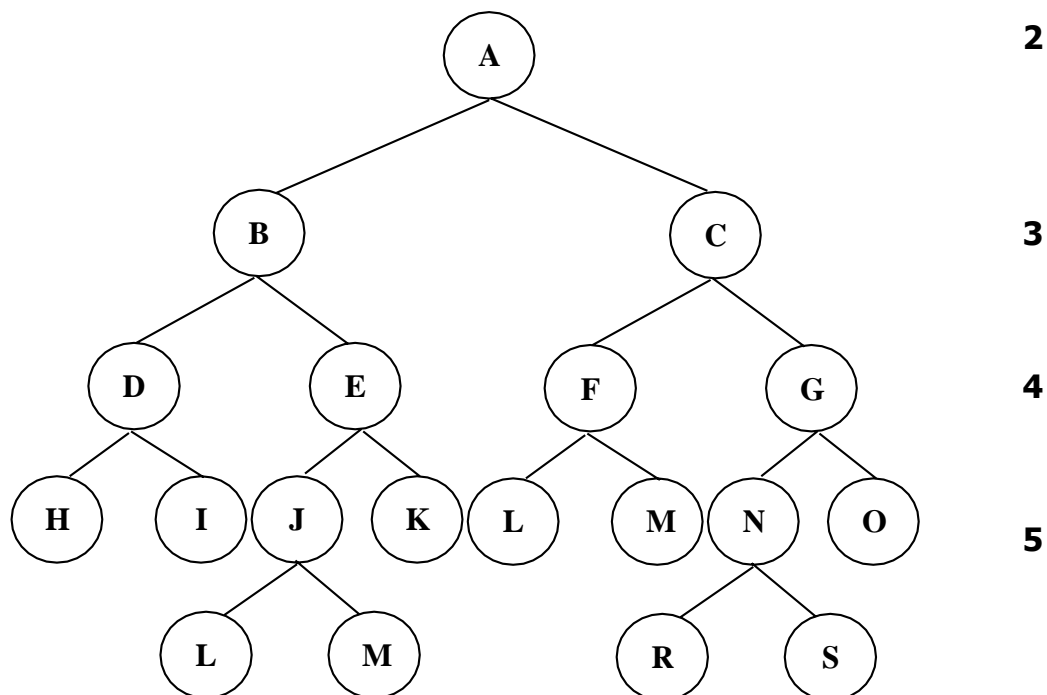
*Binary tree* merupakan tipe yang sangat penting dari struktur data *tree*, dan banyak dijumpai dalam berbagai terapan. Lebih lanjut, dalam *binary tree* akan dibedakan antara *left subtree* dengan *right subtree*, sementara dalam struktur *tree* secara umum urutan ini tidak penting. Jadi *binary tree* merupakan bentuk *tree* yang beraturan. Karakteristik lain adalah bahwa dalam *binary tree* dimungkinkan tidak mempunyai *node*. Gambar 2.8 berikut ini menunjukkan contoh suatu *binary tree*.



## Gambar 2.8 Contoh binary tree

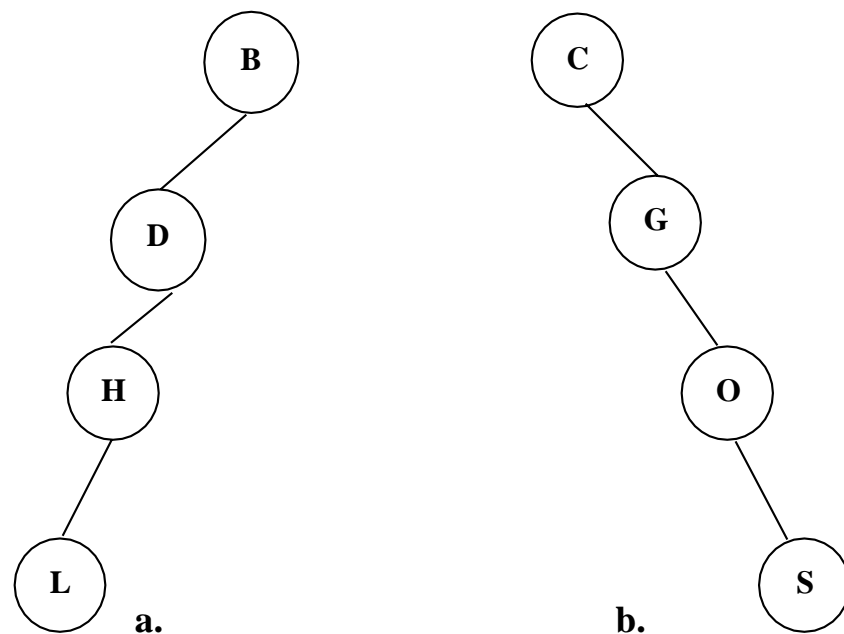
Pengertian *leaf*, *parent*, *child*, *level*, dan *degree* yang berlaku dalam *tree* juga berlaku dalam *binary tree*. Selain definisi yang telah ada, dalam *binary tree* juga dikenal istilah *complete binary tree* (pohon biner lengkap) *level N*, yang didefinisikan sebagai sembarang *binary tree* dimana semua *leaf*–nya terdapat pada *level N* dan semua *node* yang mempunyai *level* lebih kecil dari *N* selalu mempunyai *left subtree* dan *right subtree*. Gambar 2.9 berikut ini merupakan contoh *complete binary tree level 4*, tetapi bukan *complete binary tree level 5*.

### Level 1



Gambar 2.9 Complete binary tree level 4

Selain istilah *complete binary tree*, juga ada istilah *skewed binary tree* (pohon biner miring), yaitu suatu *binary tree* yang banyaknya *node* dalam *left subtree* tidak seimbang dengan banyaknya *node* dalam *right subtree*. Gambar 2.9 menunjukkan *right* dan *left skewed binary tree*.



**Gambar 2.10 Contoh skewed binary tree (a) Skewed left (b) Skewed right**

Dengan memperhatikan gambar sembarang *binary tree*, kita akan memperoleh tambahan informasi, yaitu banyaknya *node* maksimum pada *level* N adalah  $2^{(N-1)}$ . Sehingga banyaknya *node* maksimum sampai *level* N adalah:

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

Dengan demikian untuk *complete binary tree level 5*, banyaknya *leaf* adalah 16 buah dan banyaknya *node* yang bukan *leaf*, termasuk *root*, adalah 15 buah.

## Representasi Binary Tree

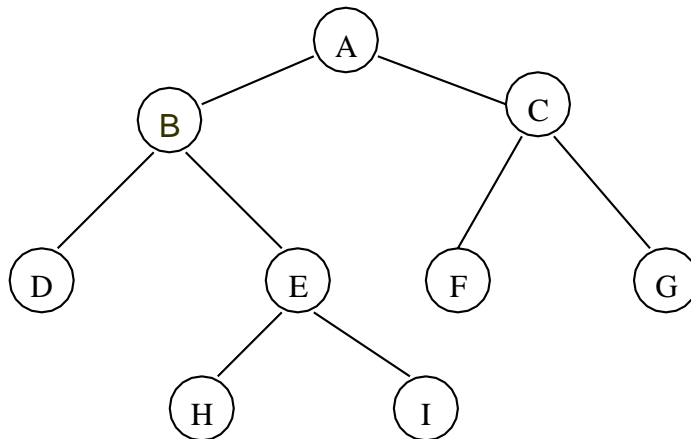
Pohon biner (*Binary Tree*) dapat disimpan / direpresentasikan dengan *array* ataupun *pointer* dimana masing-masing representasi memiliki kelebihan dan kekurangan.

### a. Representasi Pohon Biner dengan Array

Pohon biner dapat direpresentasikan dengan *array* dengan ketentuan sebagai berikut:

- Akar (*root*) ditempatkan pada posisi 1 dalam *array*
- Left subtree* data yang ada pada posisi ke-*i* diletakkan pada posisi  $2i$
- Right subtree* - nya diletakkan pada posisi  $2i + 1$

Sebagai contoh, misalkan terdapat suatu pohon biner seperti gambar 2.19 di bawah ini.



### Gambar 2.11 Pohon Biner dengan Jumlah Level 4

Sesuai dengan aturan pada gambar 2.11 di atas, maka jumlah data maksimum jika jumlah *level* = 4 adalah 15, sehingga kita menyediakan *array* dengan dimensi 15. Dengan demikian pohon di atas akan disimpan dalam *array* seperti Tabel 2.3 di bawah ini.

Tabel 2.3 Representasi Array untuk Pohon Biner pada Gambar 2.14

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	B	C	D	E	F	G			H	I				

Kelemahan dengan representasi *array* adalah sulitnya memperhitungkan jumlah sel *array* yang harus disediakan untuk bentuk pohon biner yang belum diketahui, karena permintaan tempat dilakukan sekaligus di awal program serta tidak dimungkinkan menambah selama eksekusi program. Kemudian jika bentuk pohon tidak seimbang (misalkan miring kiri atau kanan) maka jumlah tempat yang terisi sangat sedikit dibandingkan dengan jumlah tempat yang sudah disediakan. Hal ini tentunya merupakan pemborosan. Di sisi lain representasi *array* memiliki keuntungan dalam waktu *akses*, karena *array* merupakan satu blok memori yang *contiguous*.

### b. Representasi Pohon Biner dengan Linked List (pointer)

Dengan mempertimbangkan kelemahan representasi *array*, maka pilihan dengan *linked list (pointer)* menjadi lebih sesuai dengan kondisi riil. Pohon biner dengan representasi *pointer* hampir sama dengan *double linked list* dengan menggunakan dua buah pointer untuk setiap simpul yakni *pointer* kiri dan *pointer* kanan. Deklarasi pohon biner dengan definisi ini dapat dituliskan dengan,

Type

```
Pointer = ^Simpul;
```

```
Simpul = Record
```

```
Kiri: Pointer;
```

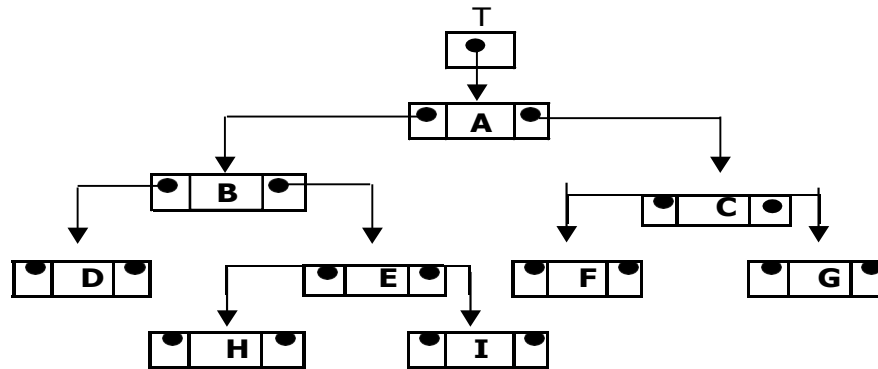
```
Data: TipeData: (*sesuai dengan kebutuhan; misalnya char, integer*) Kanan:  
Pointer;
```



End; PohonBiner =

Pointer;

Dengan representasi *linked list* maka pohon pada Gambar 2.11 di atas dapat digambarkan sebagai berikut,



**Gambar 2.12 Representasi Linked List untuk Pohon Biner pada Gambar**

Dari Gambar 2.12 di atas pohon biner dengan nama T yaitu sebuah *pointer* yang menunjuk ke simpul yang berisi data A dengan *pointer* kiri menunjuk ke simpul yang berisi data B (sebagai cabang kiri dari A) dan *pointer* kanan menunjuk ke simpul yang berisi data C (sebagai cabang kanan dari A) dan seterusnya. Keuntungan representasi *linked list* dibandingkan dengan representasi *array* adalah jumlah tempat yang dibutuhkan bersifat fleksibel. Prosedur penyelesaian *Postorder traversal* dapat dituliskan dalam bahasa *Pascal* seperti berikut,

*Procedure PostOrder (Aku : Pohon);*

*Begin*

*If (Aku = NIL) then*

*Write('Pohon Biner belum ada.')*

*Else*

*Begin*

*PostOrder(Aku^.Kiri);*

*PostOrder(Aku^.Kanan);*

*Writeln(Aku^.Isi);*

*End;*

*End;*

#### **a. Levelorder Traversal (Penelusuran Level demi Level)**

Penelusuran dengan cara ini dilakukan mulai dari *root* kemudian pendaftaran *level* demi *level* dari kiri ke kanan. Sehingga kalau terhadap pohon pada Gambar 2.11 di atas dilakukan penelusuran *by level* diperoleh urutan sebagai berikut:

A, B, C, D, E, F, G, H, I

Jika dilakukan *levelorder traversal* terhadap *binary tree* gambar 2.13, akan menghasilkan data “H, A, K, C, J, L, B, D”.

#### **Perangkat Lunak Pembelajaran**

Seiring dengan perkembangan peradaban manusia dan kemajuan pesat di bidang teknologi, tanpa disadari komputer telah ikut berperan dalam dunia pendidikan terutama penggunaannya sebagai alat bantu pengajaran. Percobaan penggunaan komputer untuk proses belajar dimulai di Amerika Serikat pada akhir

tahun 1950-an dan awal tahun 1960-an. Kemudian penelitian selanjutnya dilakukan oleh Harvard University bekerja sama dengan IBM pada tahun 1965. Setelah munculnya komputer mikro, sistem pengajaran dengan komputer menjadi semakin meluas pada pengembangan aplikasi perangkat lunak ajar yang dikenal dengan istilah perangkat lunak pembelajaran. Perangkat lunak pembelajaran dengan komputer muncul dari sejumlah disiplin ilmu, terutama ilmu komputer dan psikologi. Dari ilmu komputer dan matematika muncul program – program yang membuat semua perhitungan dan fungsi lebih mudah dan bermanfaat. Sedangkan dari ilmu psikologi muncul pengetahuan mengenai teori belajar, teknik belajar, serta motivasi yang baik.

### **Tujuan Perangkat Lunak Pembelajaran**

Tujuan dari perangkat lunak pembelajaran antara lain :

1. Peningkatan pengawasan
2. Penggunaan sumber daya
3. Individualisasi
4. Ketepatan waktu dan tingkat ketersediaan
5. Pengurangan waktu latihan
6. Perbaikan hasil kerja
7. Alat yang nyaman dipakai
8. Pengganti cara belajar
9. Peningkatan kepuasan belajar
10. Pengurangan waktu pengembangan

### **Jenis-Jenis Perangkat Lunak Pembelajaran**

Jenis pemakaian komputer untuk perangkat lunak pembelajaran digolongkan menjadi tiga bagian, yaitu :

1. Pengujian

Dalam jenis CBT, komputer digunakan untuk memberikan penilaian dan analisis tes, membuat soal tes, membuat nilai acak, tes interaksi, dan tes adaptasi. Jenis ini sering disebut dengan *Computer Assisted Testing* (CAT).

2. Manajemen

Jenis pemakaian ini disebut dengan *Computer Managed Instruction* (CMI), dimana komputer digunakan untuk mengatur kemajuan peserta pelatihan dan alat-alat yang dipakai. CMI biasanya digunakan untuk meningkatkan pengawasan dan efisiensi dalam sistem pelatihan.

3. Instruksi

Ada dua bentuk yang hampir sama mengenai pengguna komputer untuk instruksi. *Computer Assisted Instruction* (CAI), menganggap komputer sebagai media penyimpanan instruksi sama seperti *slide*, *tape*, video atau buku-buku. Menurut sudut pandang CAI, masalah utamanya adalah bagaimana menyusun bahan-bahan instruksi yang akan ditampilkan oleh komputer dengan cara yang paling efektif. Ada tiga jenis CAI yakni :

- a. *Drill and Practice*

Merupakan cara yang paling mudah, terdiri dari tahap-tahap penampilan permasalahan, penerimaan respon pengguna, pemberian hasil analisis, umpan balik, dan pemberian pertanyaan lain. Secara umum jenis ini tidak menampilkan informasi baru tapi memberikan latihan dari konsep yang sudah ada.

*b. Tutorial*

Jenis ini berisi konsep atau prosedur yang disertai dengan pertanyaan atau latihan pada akhir dari pelatihan. Selama pelatihan, komputer mengajarkan informasi-informasi yang baru kepada siswa seperti layaknya seorang guru pembimbing. Setelah itu, pemahaman siswa diukur melalui serangkaian tes dan komputer melanjutkan pengajaran berdasarkan hasil pengukuran tadi.

*c. Socratic*

Berisi komunikasi antara pengguna dan komputer dalam *natural language*. Jenis ini sebenarnya berasal dari penelitian dalam bidang inteligensia semu (*artificial intelligence*). *Socratic* mampu melakukan interaksi dalam *natural language* dan bisa memahami apa yang ditanyakan pengguna.

### **Langkah-Langkah Pengembangan Perangkat Lunak Pembelajaran**

Ada 5 tahap siklus pengembangan perangkat lunak pembelajaran yaitu :

1. Pengembangan spesifikasi perancangan detail

Tujuan dari pengembangan spesifikasi perancangan detail ini meliputi perluasan konsep perancangan untuk menciptakan suatu rencana yang efektif.

2. Pengembangan teknik

Pengembangan ini biasanya dilakukan dengan persetujuan dari tim perancang. Tahap ini biasanya akan diulang-ulang dan sering terjadi perbaikan.

3. Evaluasi

Untuk menghasilkan suatu perangkat lunak pembelajaran yang dapat memenuhi standar maka perlu dilakukan suatu pengujian. Pengujian biasanya dilakukan pada bagian pelajaran dan pelatihan. Hasil dari pengujian inilah yang dievaluasi oleh tim perancang.

#### 4. Produksi dan pengembangan

Produksi ini harus dilakukan secara teknis dan logis, baik dalam penyalinan produk CAI maupun dalam pembuatan dokumentasi. Sedangkan pengembangan yang dilakukan mengacu pada proses pengembangannya.

#### 5. Evaluasi akhir

Langkah pengujian yang dapat dilakukan misalnya dengan melakukan suatu kuisioner maupun konsultasi dengan mereka yang ingin belajar. Hasilnya dapat menjadi pedoman apakah perangkat lunak pembelajaran tersebut perlu dilakukan perbaikan lagi atau tidak.

### **Keuntungan Perangkat Lunak Pembelajaran**

Ada beberapa keuntungan yang bisa diraih dari suatu perangkat lunak pembelajaran dengan komputer yang interaktif yaitu :

1. Meningkatkan efektivitas pelatihan, seperti :
  - a. Meningkatkan daya minat pengguna.
  - b. Meningkatkan waktu pelatihan.
  - c. Meningkatkan pengetahuan.
2. Mengurangi waktu dan mengefisienkan waktu, seperti :
  - a. Mengurangi waktu belajar selama pelatihan.
  - b. Mengurangi instruktur pelatihan.
  - c. Biaya pelatihan yang lebih rendah.

## Visual Visual.Net

Microsoft Visual Basic 2008 adalah sebuah alat untuk mengembangkan dan membangun aplikasi yang bergerak di atas sistem. NET Framework, dengan menggunakan bahasa BASIC. Dengan menggunakan alat ini, para *programmer* dapat membangun aplikasi Windows Forms, Aplikasi web berbasis ASP.NET, dan juga aplikasi *command-line*. Alat ini dapat diperoleh secara terpisah dari beberapa produk lainnya (seperti Microsoft Visual C++, Visual C#, atau Visual J#), atau juga dapat diperoleh secara terpadu dalam Microsoft Visual Studio 2008.

Visual Basic juga salah satu development tools untuk membangun aplikasi dalam lingkungan Windows. Visual Basic menggunakan pendekatan Visual untuk merancang *user interface* dalam bentuk *form*, sedangkan untuk codingnya menggunakan dialek bahasa *Basic* yang cenderung mudah dipelajari. Pada pemrograman Visual, pengembangan aplikasi dimulai dengan pembentukan *user interface*, kemudian mengatur properti dari objek-objek yang digunakan dalam *user interface*, dan baru dilakukan penulisan kode program untuk menangani kejadian-kejadian. Tahap pengembangan aplikasi demikian dikenal dengan istilah pengembangan aplikasi dengan pendekatan *Bottom Up*.

Bahasa Visual Basic 2008 sendiri menganut paradigma bahasa pemrograman berorientasi objek yang dapat dilihat sebagai evolusi dari Microsoft Visual Basic versi sebelumnya yang diimplementasikan di atas. NET Framework. Peluncurannya mengundang kontroversi, mengingat banyak sekali perubahan yang dilakukan oleh Microsoft, dan versi baru ini tidak kompatibel dengan versi terdahulu.

## UML (unified Modeling Language)

Unified Modeling Language (UML) adalah himpunan [struktur](#) dan [teknik](#) untuk pemodelan [desain](#) program berorientasi objek ([OOP](#)) serta [aplikasinya](#). UML adalah [metodologi](#) untuk mengembangkan [sistem](#) OOP dan sekelompok perangkat [tool](#) untuk mendukung pengembangan sistem tersebut. UML mulai diperkenalkan oleh [Object Management Group](#), sebuah [organisasi](#) yang telah mengembangkan [model](#), [teknologi](#), dan standar OOP sejak tahun [1980](#)-an. Sekarang UML sudah mulai banyak digunakan oleh para praktisi OOP. UML merupakan dasar bagi perangkat (*tool*) desain berorientasi objek dari [IBM](#).

UML adalah suatu [bahasa](#) yang digunakan untuk menentukan, memvisualisasikan, membangun, dan mendokumentasikan suatu [sistem informasi](#). UML dikembangkan sebagai suatu alat untuk [analisis](#) dan desain berorientasi objek oleh Grady Booch, Jim Rumbaugh, dan Ivar Jacobson. Namun demikian UML dapat digunakan untuk memahami dan mendokumentasikan setiap sistem informasi. Penggunaan UML dalam [industri](#) terus meningkat. Ini merupakan standar terbuka yang menjadikannya sebagai bahasa pemodelan yang umum dalam industri [peranti lunak](#) dan pengembangan sistem.

## Tujuan UML

1. Memodelkan suatu sistem (bukan hanya perangkat lunak) yang menggunakan konsep berorientasi object.
2. Menciptakan suatu bahasa pemodelan yang dapat digunakan baik oleh manusia maupun mesin.


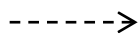
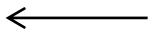
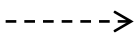







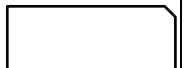
3. Memberikan model yang siap pakai, bahasa pemodelan visual yang ekspresif untuk mengembangkan dan saling menukar model dengan mudah dan dimengerti secara umum.
4. UML bisa juga berfungsi sebagai sebuah (blue print) cetak biru karena sangat lengkap dan detail. Dengan cetak biru ini maka akan bias diketahui informasi secara detail tentang coding program atau bahkan membaca program dan menginterpretasikan kembali ke dalam bentuk diagram (reverse engineering).

## Use Case

Use case adalah rangkaian/uraian sekelompok yang saling terkait dan membentuk sistem secara teratur yang dilakukan atau diawasi oleh sebuah aktor. Use case digunakan untuk membentuk tingkah-laku benda dalam sebuah model serta di Realisasikan oleh sebuah collaboration. Umumnya use case digambarkan dengan sebuah elips dengan garis yang solid, biasanya mengandung nama. Use case menggambarkan proses system (kebutuhan system dari sudut pandang user).

Tabel 2.4 Simbol-Simbol Use Case

NO	GAMBAR	NAMA	KETERANGAN
1		Actor	Menspesifikasikan himpunan peran yang pengguna mainkan ketika berinteraksi dengan use case.
2		Dependency	Hubungan dimana perubahan yang terjadi pada suatu elemen mandiri (independent) akan mempengaruhi elemen yang tidak mandiri
3		Generalization	Hubungan dimana objek anak (descendent) berbagai perilaku dan struktur data dari objek yang ada di atasnya objek induk (ancestor).
4		Include	Menspesifikasikan bahwa use case sumber secara eksplisit.

5		Extend	Menspesifikasikan bahwa use case target memperluas perilaku dari use case sumber pada suatu titik yang diberikan.
6		Association	Apa yang menghubungkan antara objek satu dengan objek lainnya.
7		System	Menspesifikasikan paket yang menampilkan sistem secara terbatas.
8		Use Case	Deskripsi dari urutan aksi-aksi yang ditampilkan sistem yang menghasilkan suatu hasil yang terukur bagi suatu actor.
9		Collaboration	Interaksi aturan-aturan dan elemen lain yang bekerja sama untuk menyediakan perilaku yang lebih besar dari jumlah dan elemen-elemennya.
10		Note	Elemen fisik yang eksis saat aplikasi dijalankan dan mencerminkan suatu sumber daya komputasi

Sumber: Yuni Sugiarti, S.T., M.Kom, 2013

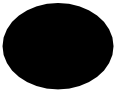


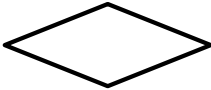


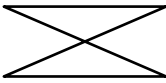
## Activity Diagram

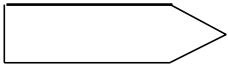
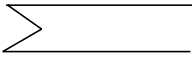

Diagram aktivitas atau dalam bahasa Inggris [activity diagram](#) adalah representasi grafis dari seluruh tahapan alur kerja. Diagram ini mengandung aktivitas, pilihan tindakan, perulangan dan hasil dari aktivitas tersebut. Pada pemodelan [UML](#), diagram ini dapat digunakan untuk menjelaskan proses bisnis dan alur kerja operasional secara langkah demi langkah dari komponen suatu sistem.

*Activity* diagram menggambarkan berbagai alir aktivitas dalam sistem yang sedang dirancang, bagaimana masing-masing alir berawal, decision yang mungkin terjadi, dan bagaimana mereka berakhir. *Activity* diagram juga dapat menggambarkan proses paralel yang mungkin terjadi pada beberapa eksekusi. *Activity* diagram merupakan state diagram khusus, di mana sebagian besar state

adalah *action* dan sebagian besar transisi di-trigger oleh selesainya state sebelumnya (internal processing). Tetapi lebih menggambarkan proses-proses dan jalur-jalur aktivitas dari level atas secara umum. Menggambarkan proses bisnis dan urutan aktivitas dalam sebuah proses. Dipakai pada business modeling untuk memperlihatkan urutan aktifitas proses bisnis. Struktur diagram ini mirip flowchart atau Data Flow Diagram pada perancangan terstruktur. Sangat bermanfaat apabila kita membuat diagram ini terlebih dahulu dalam memodelkan sebuah proses untuk membantu memahami proses secara keseluruhan. *Activity* diagram dibuat berdasarkan sebuah atau beberapa use case pada use case diagram.

Tabel 2.5 Activity Diagram

Simbol	Keterangan
	Titik awal
	Titik akhir
	Activity
	Pilihan untuk pengambilan keputusan
	<i>Fork</i> ; Digunakan untuk menunjukkan kegiatan yang dilakukan secara parallel atau untuk menggabungkan dua kegiatan paralel menjadi satu.
	<i>Rake</i> ; Menunjukkan adanya dekomposisi
	Tanda waktu

	Tanda pengiriman
	Tanda penerimaan
	Aliran akhir ( <i>Flow Final</i> )

## Daftar Pustaka

- Gautam, K. Das. (2011). Properties of Context-free Languages. Retrieved from <http://www.iitg.ernet.in/gkd/ma513/oct/oct18/note.pdf>
- Utdirartamo, F. (2005). Teori Bahasa dan Otomata. Yogyakarta: Penerbit Graha Ilmu.
- Utdirartatmo, F. (2001). Teori Bahasa Dan Otomata. J & J Learning. Yogyakarta.
- Hopcroft, John E.; Motwani, Rajeev.; Ullman, Jeffrey D. 2001.
- Introduction to Automata Theory, Languages, and Computation. Addison-Wesley. Martin, John C. 1997.
- Introduction To Languages And The Theory Of Computation. TheMcGraw-Hill Companies, Inc. Lewis, Harry R.; Papadimitriou, Christos H. 1981.
- Elements Of The Theory Of Computation. Prentice-Hall, Inc.