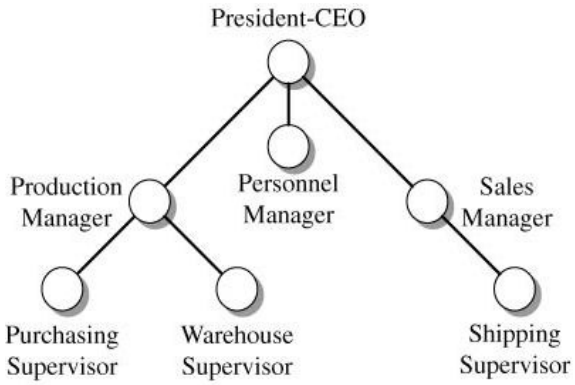
Binary Tree

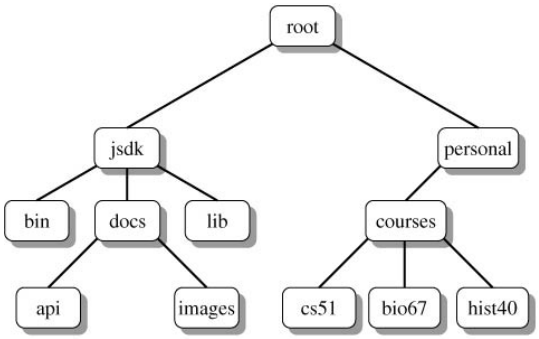
Tree

- Sebuah tree merepresentasikan sebuah hirarki
- Misal : Struktur organisasi sebuah perusahaan



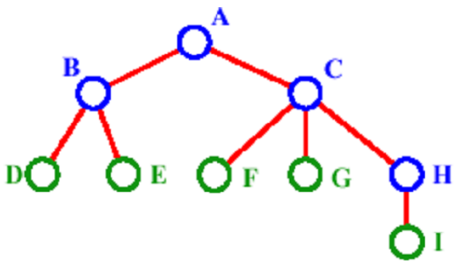
Contoh Tree

- Sistem operasi menggunakan tree untuk struktur file



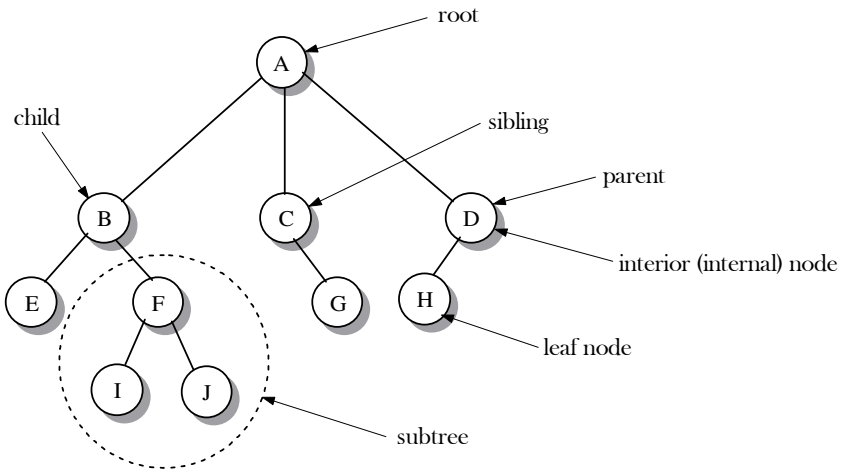
Path name: /jsdk/docs/api

Istilah Umum di Tree



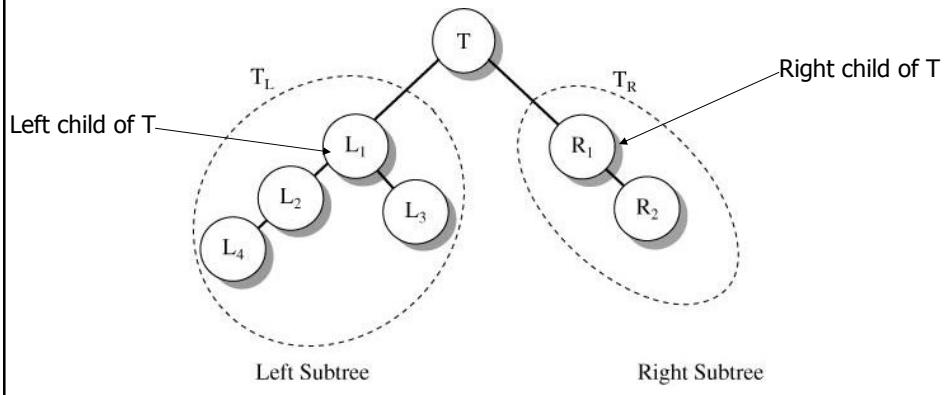
- A adalah **root** dari Tree
- B adalah **parent** dari D dan E
- C adalah **sibling** dari B
- D dan E adalah **children**/anak dari B
- D, E, F, G, I adalah **external nodes** atau **leaf**
- A, B, C, H adalah **internal nodes**
- Tinggi/**height** dari tree adalah 3
- B,D,E adalah **subtree**

Istilah Umum di Tree

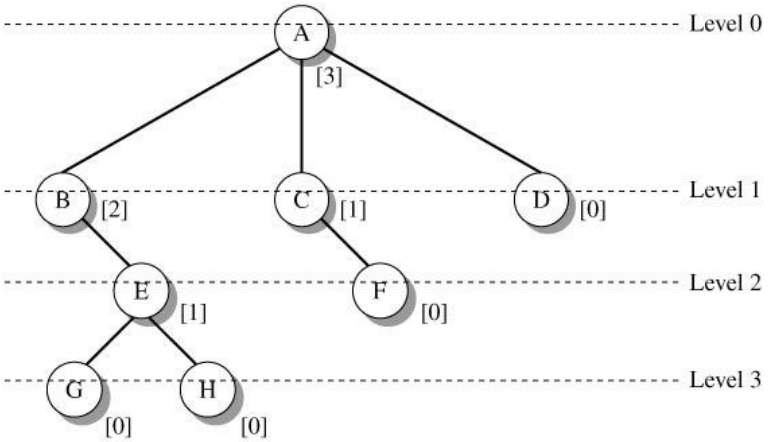


Binary Trees (continued)

- Tiap node pada binary tree adalah **subtree kiri** dan **subtree kanan**.
- Setiap **Subtree** adalah juga **tree**.



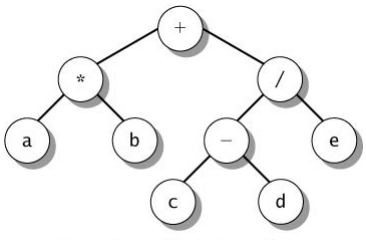
Level



Tree illustrates the level and height of a node.

Binary Tree

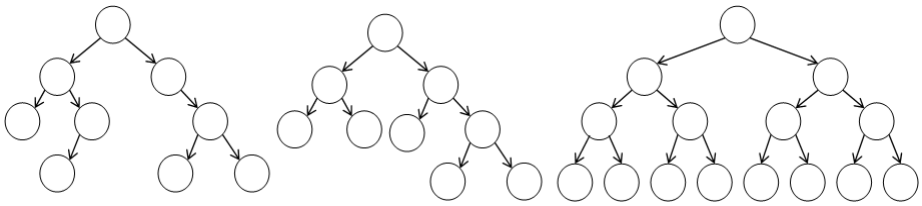
- Setiap node dalam Tree mempunyai maksimum dua anak



Expression: a* b + (c − d)/e

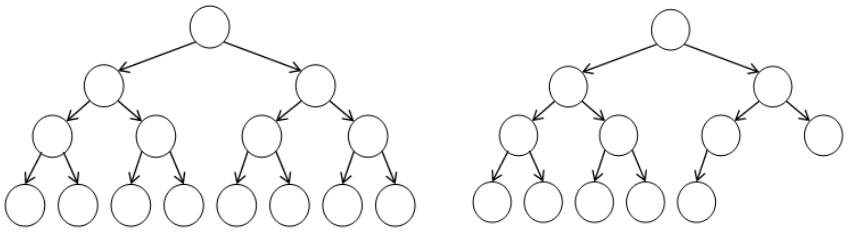
Binary Tree

- **Binary tree** adalah tree di mana setiap nodes memiliki maksimum 2 anak
- **Full Binary tree** adalah binary tree di mana setiap nodes memiliki anak 0 atau anak 2.
- **Perfect Binary tree** adalah binary tree di mana setiap nodes memiliki anak 0 atau 2 dan setiap leaf berada pada level terbawah



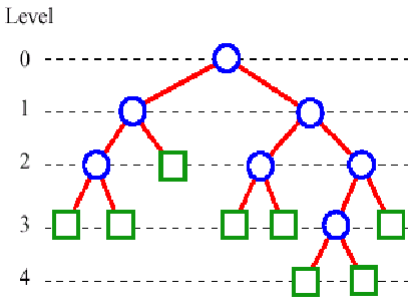
Binary Tree

- **Complete Binary tree**: binary tree dengan tinggi k adalah binary tree yang miliki jumlah maximum nodes di levels 0 sampai k – 1, dan pada level k seluruh node mampat ke kiri.
- Jadi suatu **perfect binary tree** adalah juga complete binary tree



Full Binary Tree

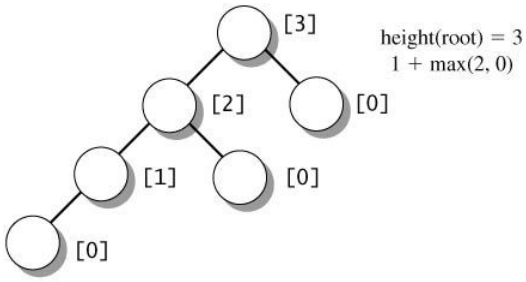
- (# external nodes) = (# internal nodes) + 1
- (# nodes at level i) ≤ 2^i
- (# external nodes) ≤ $2^{(height)}$
- (height) ≥ $\log_2$(# external nodes)
- (height) ≥ $\log_2$(# nodes) - 1
- (height) ≤ (# internal nodes) = ((# nodes) - 1)/2
- Jika tinggi = k, maka #node = 2^{k+1}-1



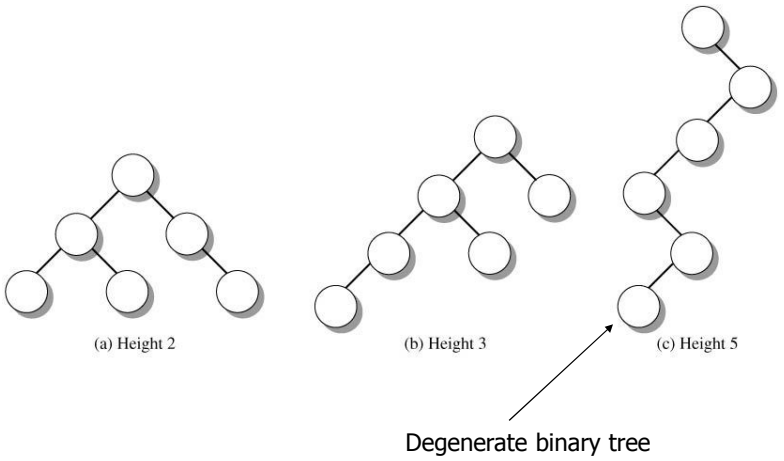
Height dari Binary Tree

- **Height**/kedalaman tree adalah maksimum level dari tree.
- Misal T_N adalah subtree dengan root N dan T_L adalah root subtree kiri dan T_R adalah root dan subtree kanan.

$$height(N) = height(T_N) = \begin{cases} -1 & \text{if } T_N \text{ is empty} \\ 1+\max(height(T_L), height(T_R)) & \text{if } T_N \text{ not empty} \end{cases}$$



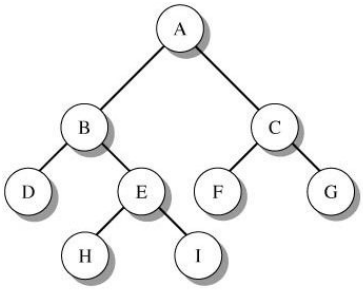
Height dari Binary Tree



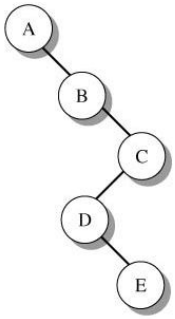
Density Binary Tree

- Jumlah node di tiap level berada pada range tertentu.
  - Level 0, terdapat 1 node yaitu root.
  - Level 1, mempunyai 1 atau 2 node
  - Level k, jumlah node antara 1 to 2^k
- **Densitas** adalah besar/size tree berdasarkan jumlah node relatif terhadap tinggi/height tree.

Density Binary Tree (continued)



Tree A
Size 9 Height 3

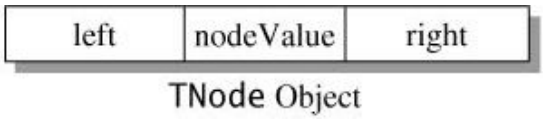


Tree B
Size 5 Height 4

Implementasi Binary Tree dalam Bahasa Java

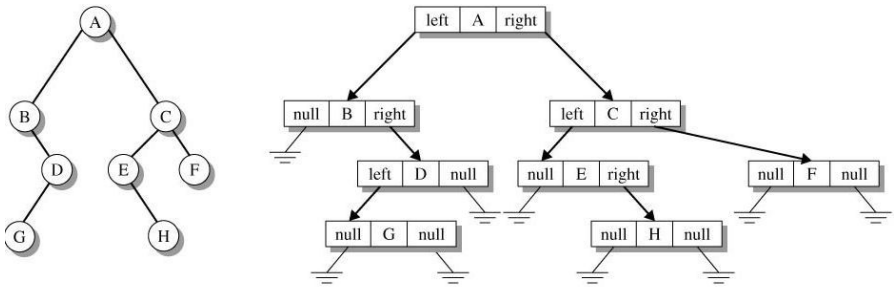
Node Binary Tree

- Class Tnode merepresentasikan sebuah Node pada Binary Tree
- Sebuah Node terdiri dari 3 field yaitu
  - Data disebut nodeValue.
  - Variabel reference, left untuk menyimpan alamat dari anak kiri dan right untuk menyimpan alamat dari anak kanan



TNode Object

Node Binary Tree



Abstract Tree Model

Tree Node Model

Abstract and tree node model of a binary tree.

Node Binary Tree

- Dengan menggunakan class TNode dapat membangun Binary Tree sebagai kumpulan dari obyek TNode.

Class TNode

```
public class TNode<T>
{
   // data dari sebuah node
   public T nodeValue;
   // menyimpan alamat dari anak kiri dan anak kanan
   public TNode<T> left, right;

   // membuat obyek dengan sebuah data item
   // dan anak kiri dan anak kanan diset null
   public TNode(T item)
   {
      nodeValue = item;
      left = right = null;
   }
```

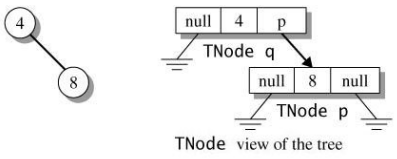
Class TNode

```
    // membuat obyek dengan sebuah data item
    // dan menentukan alamat dari anak kiri dan anak kanan
    public TNode (T item, TNode<T> left, TNode<T> right)
    {
       nodeValue = item;
       this.left = left;
       this.right = right;
    }
}
```


Membangun Binary Tree

- Binary Tree merupakan kumpulan dari obyek TNode.
- TNode mempunyai variabel reference left untuk menghubungkan node tersebut dengan anak kiri dan variabel reference right untuk menghubungkan node tersebut dengan anak kanan. Sehingga akan terbentuk Binary Tree

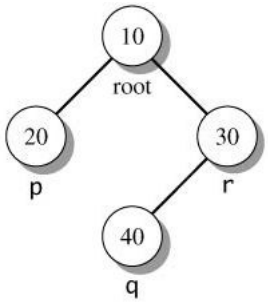
```
TNode<Integer> p, q;  // references to TNode objects with
                      // Integer data
// p is a leaf node with value 8;
p = new TNode<Integer>(8);

// q is a node with value 4 and p as a right child
q = new TNode<Integer>(4, null, p);
```



Membangun Binary Tree

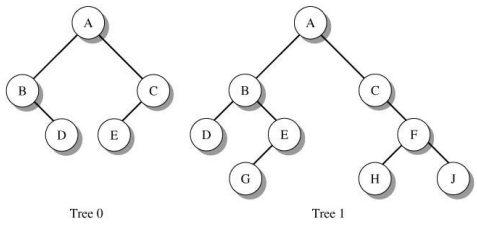
- Menggunakan Class Tnode untuk membangun Binary Tree dengan bottom up.

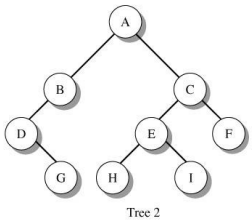
```
TNode<Integer> root, p, q, r;
// membuat lef node p dengan nilai 20
// dan leaf node q dengan nilai 40
p = new TNode<Integer>(20);
q = new TNode<Integer>(40);
// membuat internal node r dengan nilai 30
// left child q, dan right child diset null
r = new TNode<Integer>(30, q, null);
// membuat root node dengan nilai 10
// left child p, and right child r
root = new TNode<Integer>(10, p, r);
```



Membangun Binary Tree

```
// n is in the range 0 to 2
public static TNode<Character> buildTree(int n)
{ ... }
```





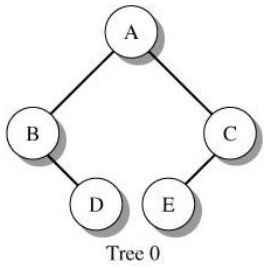
Trees created by buildTree().

Algoritma Binary Tree-Scan Secara Rekursif

- Langkah-langkah untuk membaca/scan tree secara rekursif :
  - mengunjungi node(N)
  - Membaca subtree kiri (L)
  - Membaca subtree kana (R)
- Urutan dari N, L, R akan menentukan algoritma pembacaan tree

Inorder Scan

- Langkah-langkah pembacaan tree menggunakan InOrder:
  - Mengunjungi subtree kiri L,
  - Mengunjungi node N,
  - Mengunjungi subtree kanan R.

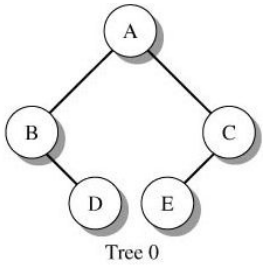


Tree 0

Scan order: B D A E C

Preorder Scan

- Langkah-langkah pembacaan tree menggunakan PreOrder:
  - Mengunjungi node N,
  - Mengunjungi subtree kiri L,
  - Mengunjungi subtree kanan R.

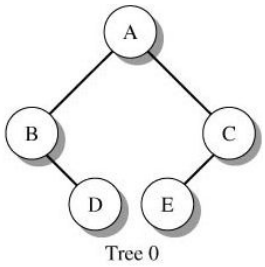


Tree 0

Scan order: A B D C E

Postorder Scan

- Langkah-langkah pembacaan tree menggunakan PostOrder:
  - Mengunjungi subtree kiri L,
  - Mengunjungi subtree kanan R.
  - Mengunjungi node N,

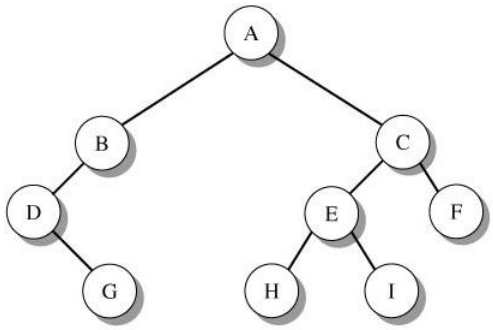


Tree 0

Scan order: D B E C A

Contoh Pembacaan Tree Secara Rekursif

```
Preorder (NLR):     A  B  D  G  C  E  H  I  F
Inorder (LNR):      D  G  B  A  H  E  I  C  F
Postorder (LRN):    G  D  B  H  I  E  F  C  A
```



Tree 2

Algoritma Inorder

```
public static <T> String inorderDisplay(TNode<T> t)
{
   String s = "";

   if (t != null)
   {
      s += inorderDisplay(t.left);
      s += t.nodeValue + "  ";
      s += inorderDisplay(t.right);
   }

   return s;
}
```

Algoritma Preorder

```
public static <T> String preorderDisplay(TNode<T> t)
{
   String s = "";

   if (t != null)
   {
      s += t.nodeValue + "  ";
      s += preorderDisplay(t.left);
      s += preorderDisplay(t.right);
   }

   return s;
}
```


Algoritma Postorder

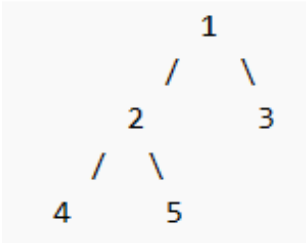
```
public static <T> String postorderDisplay(TNode<T> t)
{
   String s = "";

   if (t != null)
   {
      s += postorderDisplay(t.left);
      s += postorderDisplay(t.right);
      s += t.nodeValue + "  ";
   }

   return s;
}
```

Algoritma Pembacaan Tree Inorder Non Rekursif

1) Buatlah Stack S.
2) Lakukan inisialisasi current node sebagai root
3) Push current node ke stack S dan set current = current.left sampai current =null
4) Jika current = null dan stack tidak kosong maka lakukan
   - N = Pop Stack
   - Cetak data dari node N tersebut dan set current = N.right
   - Kembali ke langkah 3
5) Jika current = null dan stack kosong maka proses berhenti.

Algoritma Pembacaan Tree Inorder Non Rekursif

- Step 1 Buat Stack S.
- Step 2 set root=current
- Step 3
  - current → 1
  - push 1 pada Stack S [1]
  - current → 2
  - push 2 pada Stack S [1, 2]
  - current → 4
  - push 4 pada Stack S [1, 2, 4]
  - current = NULL
- Step 4 pop dari Stack S
  - N = Pop 4 dari Stack S [1, 2]
  - Cetak "4". Current = N.right
  - current = null /*kanan dari node 4 */ menuju ke step 4



Algoritma Pembacaan Tree Inorder Non Rekursif

- Step 4.
  - N = Pop 2 dari Stack S [1]
  - cetak "2“. Set current = N.right
  - current → 5/*kanan dari node 2 */ dan menuju ke step 3
- Step 3
  - Push 5 ke stack S dan current = current.left (current = null). Stack S [1, 5] menuju ke step 4
- Step 4
  - N=Pop 5 dari Stack S [1]
  - cetak "5“Set current = N.right
  - current = NULL /*kanan dari node 5 */ menuju ke step 4

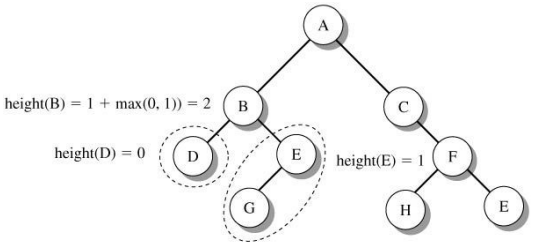
Algoritma Pembacaan Tree Inorder Non Rekursif

- Step 4.
  - N = Pop 1 dari Stack S [ ]
  - cetak "1" Set current = N.right (Node 3)
  - current adalah Node 3 /*kanan dari Node 5 */
- Step 3
  - Push 3 ke stack S [3] dan current = current.left (current = null).
  - menuju ke step 4
- Step 4
  - N = Pop 3 dari Stack S []
  - cetak "3“. Set current = N.right (current = null)
- Karena Stack sudah kosong dan current = null maka proses pembacaan tree selesai

Menghitung Tinggi Tree (Tree Height)

- Menghitung tinggi dari tree dengan rekursif

$$\text{height}(T) = \begin{cases} -1 & \text{if T is empty} \\ 1 + \max(\text{height}(T_L), \text{height}(T_R)) & \text{if T is nonempty} \end{cases}$$



Binary tree of height 3.

Menghitung Tinggi Tree (Tree Height)

```
public static <T> int height(TNode<T> t)
{
   int heightLeft, heightRight, heightval;

   if (t == null)
      // tinggi dari tree kosong adalah -1
      heightval = -1;
   else
   {
      // mendapatkan tinggi dari subtree kiri dari tree t
       heightLeft = height(t.left);
      // mendapatkan tinggi dari subtree kanan dari tree t
      heightRight = height(t.right);
```

Menghitung Tinggi Tree (Tree Height)

```
        // tinggi tree adalah 1 + max dari dua subtree
            heightval = 1 +
           (heightLeft > heightRight ? heightLeft :
           heightRight);
    }

    return heightval;
}
```


Mengkopi Binary Tree

- Mengkopi Binary Tree menggunakan postorder scan, menduplikat tree dari bawah ke atas (bottom up)

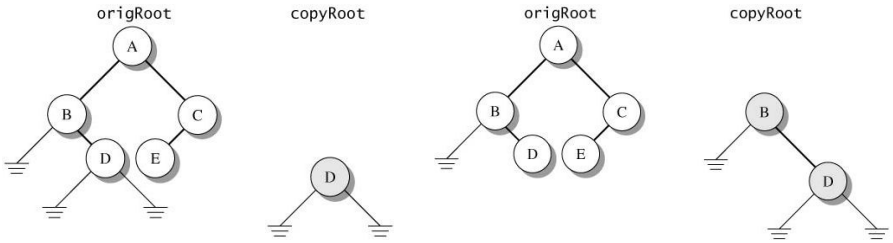
public static <T> TNode<T> copyTree(TNode<T> t)

Original Tree 0
`origRoot = TNode.buildTree (0)`

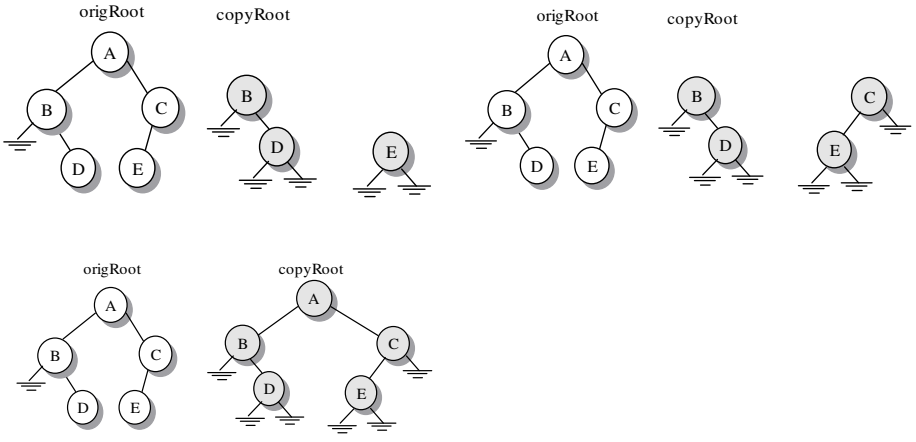
Copy of Tree 0
`copyRoot = TNode.copyTree (origRoot)`

Tree 0 and the copy of Tree 0 with roots origRoot and copyRoot respectively.

Mengkopi Binary Tree



Mengkopi Binary Tree



Mengkopi Binary Tree

```
public static <T> TNode<T> copyTree(TNode<T> t)
{
   TNode<T> newLeft, newRight, newNode;

   if (t == null)
      return null;
   newLeft = copyTree(t.left);
   newRight = copyTree(t.right);

   newNode = new TNode<T> (t.nodeValue, newLeft,
             newRight);
   return newNode;
}
```


Membersihkan a Binary Tree

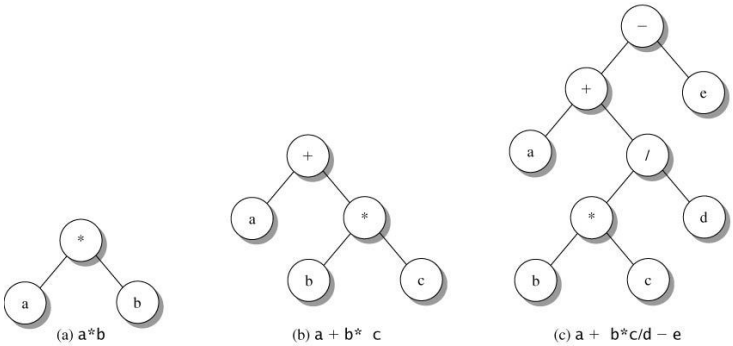
- Membersihkan(clear) tree dengan postorder scan. Proses ini akan menghapus subtree kiri dan subtree kanan sebelum menghapus node.

```
public static <T> void clearTree(TNode<T> t)
{
   // postorder scan; delete left and right
   // subtrees of t and then node t
   if (t != null)
   {
      clearTree(t.left);
      clearTree(t.right);
      t = null;
   }
}
```

Aplikasi Binary Tree

Ekspresi Aritmatika direpresentasikan Binary Tree

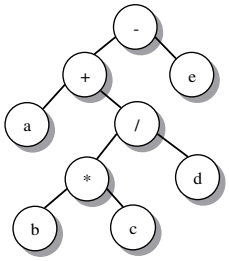
- Ekspresi aritmatika dapat direpresentasikan dengan binary tree
- Pada ekspresi aritmatika terdapat operator dan operand. Operator sebagai internal node dan Operand sebagai leaf.



Ekspresi Aritmatika direpresentasikan Binary Tree

- Pembacaan Tree
  - Inorder → Infix
  - Preorder → Prefix
  - Postorder → Postfix

Mengubah Notasi Postfix menjadi Binary Tree



a + b*c/d - e

```
Preorder(Prefix):   - + a / * b c d e   // preorder scan
Inorder(Infix):     a + b * c / d - e   // inorder scan
Postorder(Postfix): a b c * d / + e -   // postorder scan
```

Mengubah Notasi Postfix menjadi Binary Tree

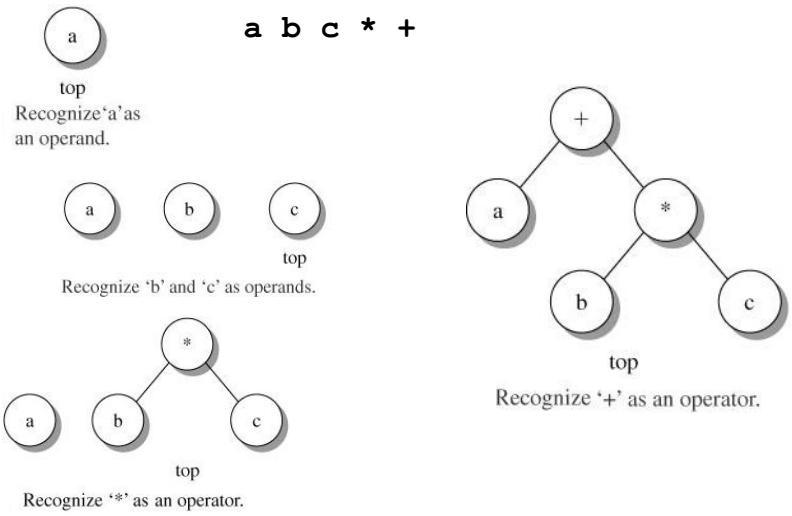
- Siapkan stack S, data yang dimasukkan pada stack S adalah objek TNode
- Jika karakter adalah operand
  - membuat node baru dengan nilai operand, dengan anak kiri dan kanan bernilai null. Push node tersebut ke stack dengan tipe TNode

Mengubah Notasi Postfix menjadi Binary Tree

- Jika karakter adalah operator
  - Jika stack kosong maka tampilkan exception “Terlalu banyak operator”
  - Pop node N1 dari stack S, tentukan sebagai operand kanan.
  - Pop node N2 dari stack S, tentukan sebagai operand kiri.
  - Buat node baru M dengan nilai operator dan tentukan anak kiri dengan node N1 dan anak kanan dengan node N2.
  - Masukkan Node M ke Stack S.
- Jika akhir String pop Stack(), dan jika ternyata stack S masih belum kosong, maka tampilkan exception “Terlalu banyak operand”

Mengubah Notasi Postfix menjadi Binary Tree

a b c * +



buildExpTree()

```
public static TNode<Character> buildExpTree(
String postfixExp)
{
   // newNode is a reference to the root of subtrees
   // we build, and newLeft/newRight are its children
   TNode<Character> newNode, newLeft, newRight;
   char token;
   // subtrees go into and off the stack
   ALStack<TNode<Character>> s = new ALStack<
         TNode<Character>>();
   int i = 0, n = postfixExp.length();

   // loop until i reaches the end of the string
   while(i != n)
   {
      // skip blanks and tabs in the expression
      while (postfixExp.charAt(i) == ' ' ||
             postfixExp.charAt(i) == '\t')
         i++;
```

buildExpTree() (continued)

```
      // if the expression has trailing whitespace,
      // we could be at the end of the string
      if (i == n)
         break;

      // extract the current token and increment i
      token = postfixExp.charAt(i);
      i++;

      // see if the token is an operator or an operand
      if (token == '+' || token == '-' ||
          token == '*' || token == '/')
      {
         // current token is an operator; pop two
         // subtrees off the stack
         newRight = s.pop();
         newLeft = s.pop();
```


buildExpTree() (continued)

```
         // create a new subtree with token as root and
         // subtrees newLeft and newRight and push it
         // onto the stack
         newNode = new TNode<Character>(token,
               newLeft,newRight);
         s.push(newNode);
      }
      else // must be an operand
      {
         // create a leaf node and push it onto the stack
         newNode = new TNode<Character>(token);
         s.push(newNode);
      }
   }
```

buildExpTree() (concluded)

```
   // if the expression was not empty, the root of
   // the expression tree is on the top of the stack
   if (!s.isEmpty())
      return s.pop();
   else
      return null;
}
```