STRUKTUR DATA

N-ER TREE / POHON N-ER

Rosa Ariani Sukamto

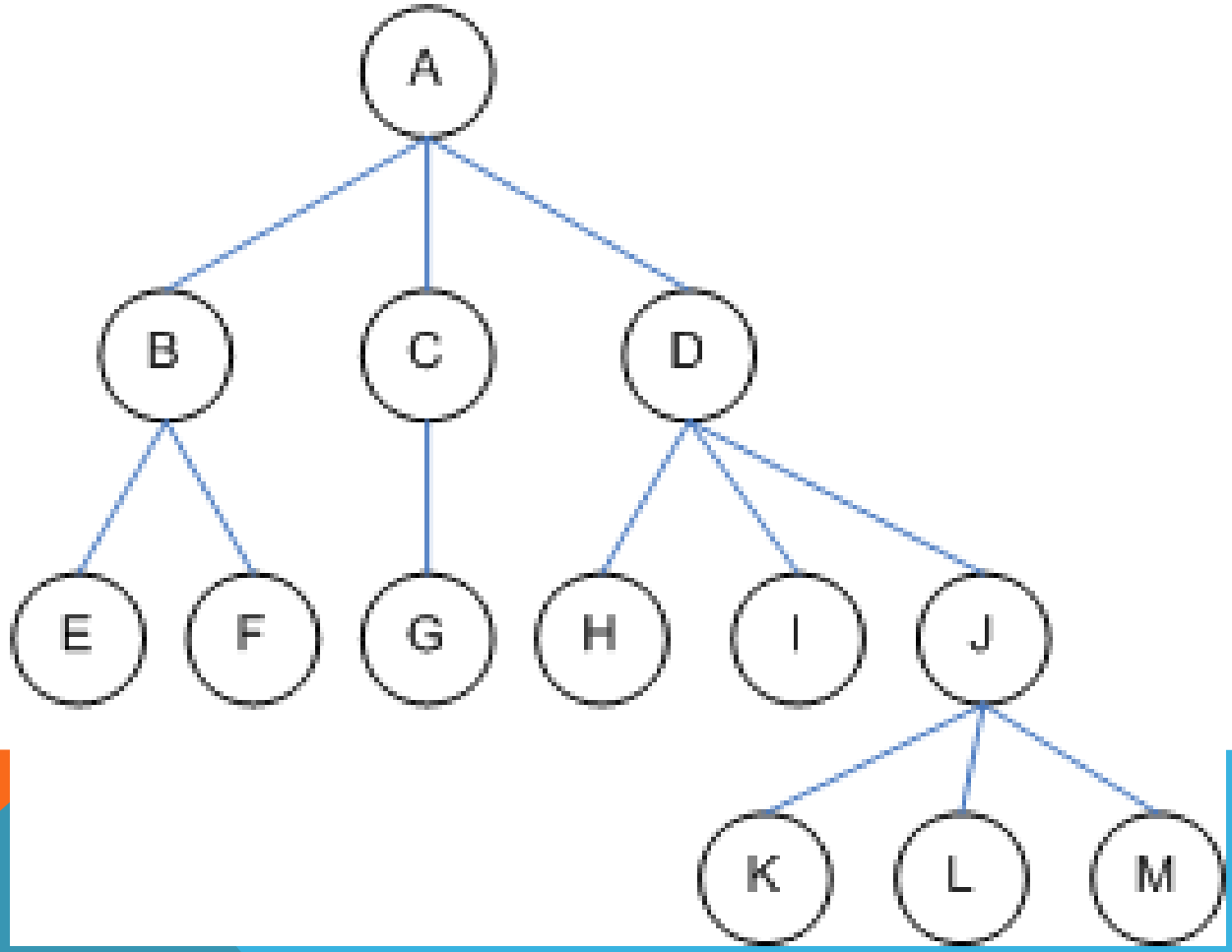
ROSA ARIANI SUKAMTO

Blog: http://hariiniadalahhadiah.wordpress.com

Facebook: https://www.facebook.com/rosa.ariani.sukamto

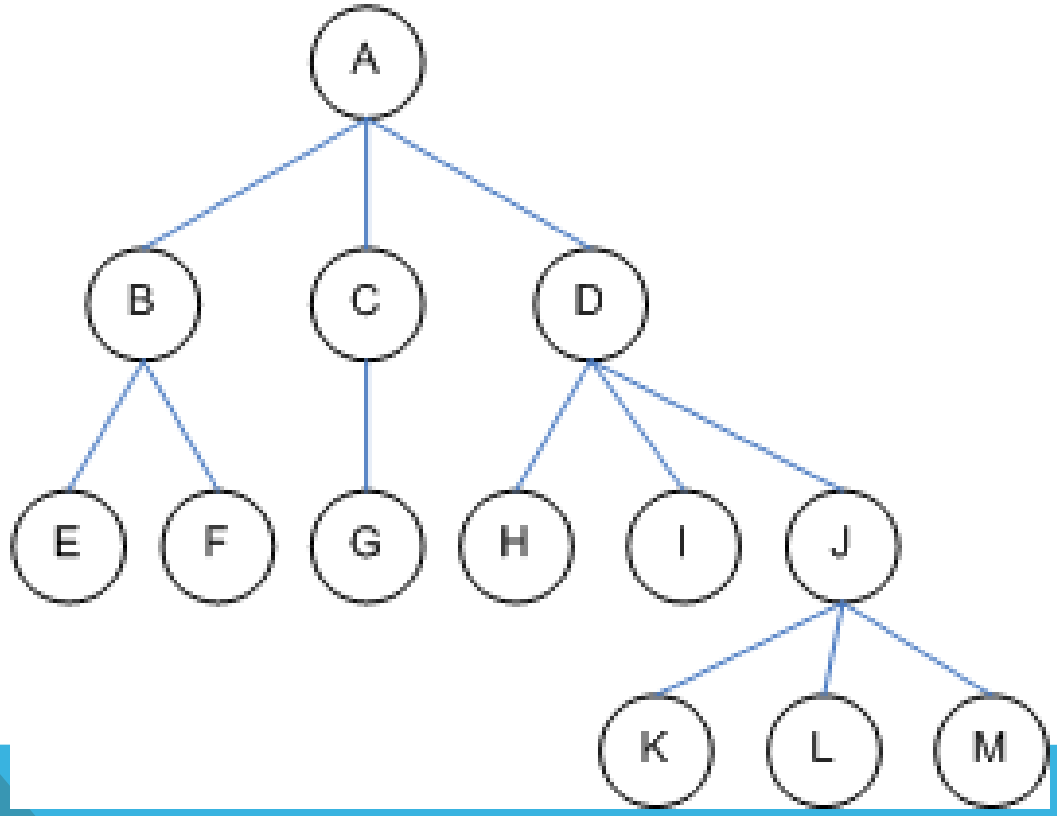
Email: rosa_if_itb_01@yahoo.com

PENGGAMBARAN POHON N-ER



OPERASI KUNJUNGAN POHON N-ER (1)

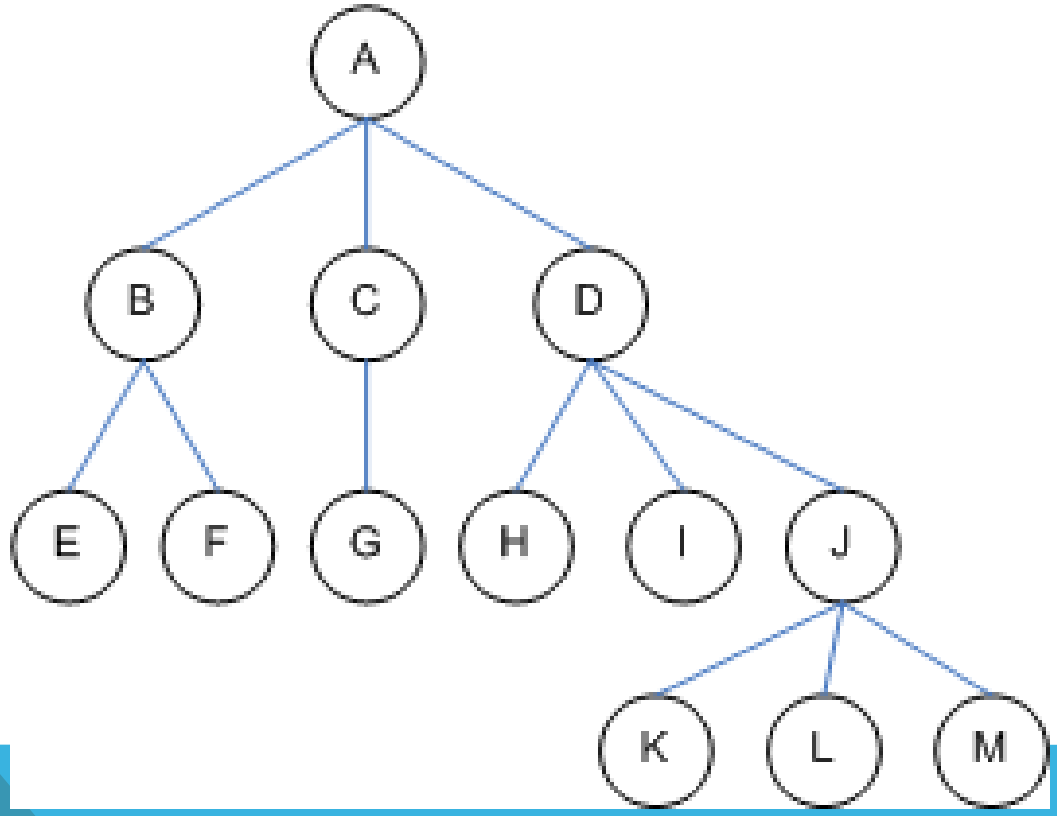
PreOrder : Kunjungan dari akar, kemudian anak



A – B – E- F – C – G – D – H – I – J – K – L – M

OPERASI KUNJUNGAN POHON N-ER (2)

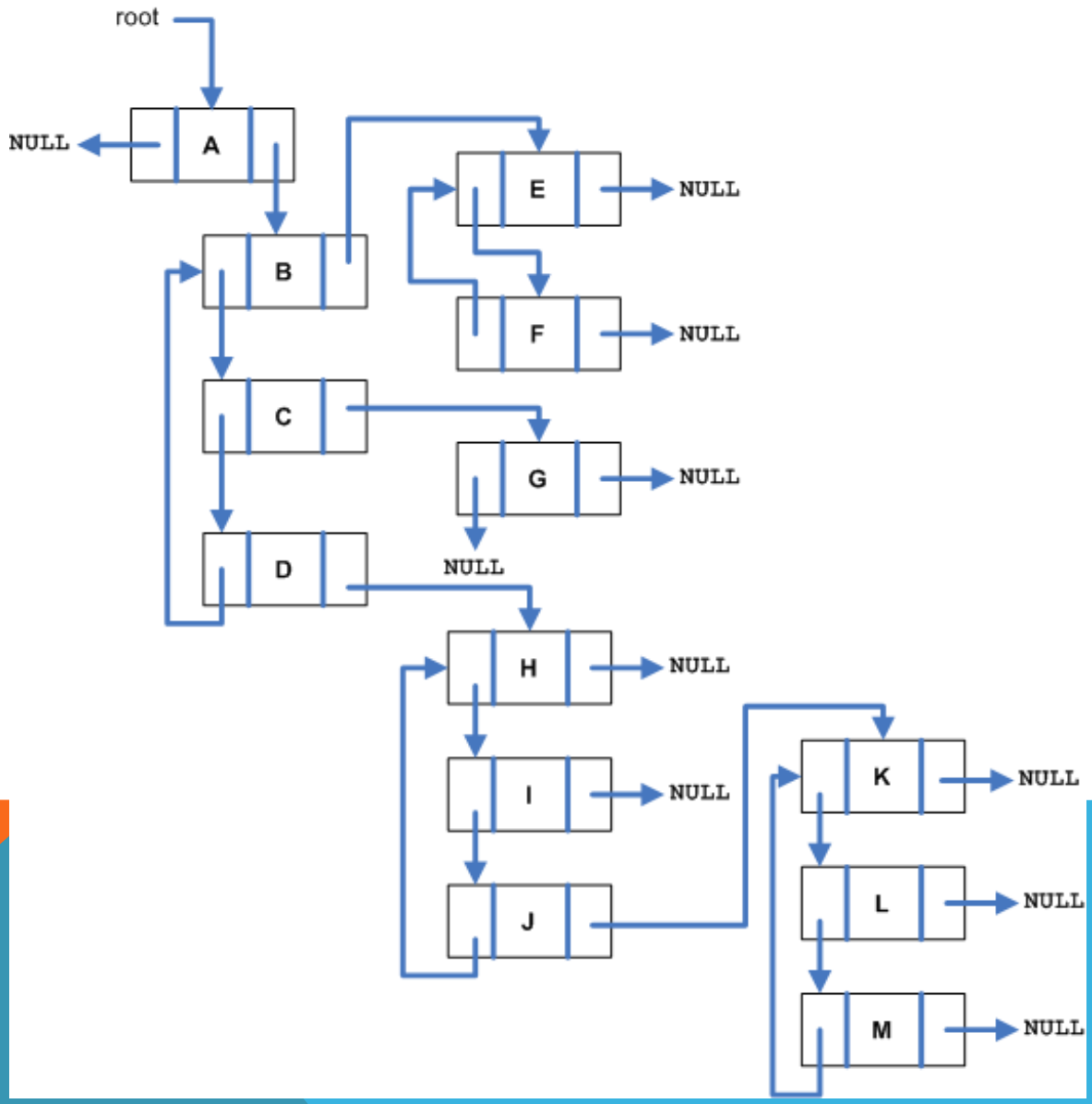
PostOrder : Kunjungan dari anak, lalu akar



E – F – B – G – C – H – I – K – L – M – J – D – A

IMPLEMENTASI POHON N-ER (1) - ELEMEN

IMPLEMENTASI POHON N-ER (2) - POHON



DEKLARASI ELEMEN DAN INISIALISASI

```
#include <stdio.h>
#include <malloc.h>

typedef struct smp *alamatsimpul;
typedef struct smp{
  char info;
  alamatsimpul sibling;
  alamatsimpul child;
}simpul;

typedef struct{
  simpul *root;
}tree;
```

```
void makeTree(char c, tree *T){


  simpul *node;
  node = (simpul *) malloc
   (sizeof (simpul));
  node->info = c;
  node->sibling = NULL;
  node->child = NULL;
  (*T).root = node;


}
```

ADDCHILD

```
void addChild(char c, simpul *root){
  if(root != NULL){
    /*jika root tidak kosong*/
    simpul *node;
    node = (simpul *) malloc (sizeof
   (simpul));
    node->info = c;
    node->child = NULL;
    if(root->child == NULL){
      /*simpul baru menjadi anak
   pertama*/
      node->sibling = NULL;
      root->child = node;
    }else{
     if(root->child->sibling
        == NULL){
      /*jika simpul baru menjadi
   anak kedua*/
      node->sibling = root->child;
      root->child->sibling = node;
     }
```

```
else{
        simpul *last = root-
   >child;

        /*mencari simpul anak
   terakhir*/
        while(last->sibling !=
          root->child){
           last = last->sibling;
        }

        node->sibling =
                  root->child;
        last->sibling = node;

      }
    }
  }
}
```

DELCHILD (1)

```
void delChild(char c, simpul
   *root){
  simpul *node = root->child;

  if(node != NULL){
    if(node->sibling == NULL){
      /*jika hanya mempunyai satu
   anak*/
      if(root->child->info == c){
        root->child = NULL;
        free(node);
      }
      else{
        printf("tidak ada simpul
   anak dengan info karakter
   masukan\n");
      }
    }
```

```
    else{
      /*jika memiliki banyak anak*/

      simpul *prec = NULL;

      /*mencari simpul yang akan
   dihapus*/

      int ketemu = 0;
      while((node->sibling !=
        root->child)&&
        (ketemu == 0)){
        if(node->info == c){
          ketemu = 1;
        }
        else{
          prec = node;
          node = node->sibling;
        }
      }
```

DELCHILD (2)

```
    /*memproses simpul anak
terakhir karena belum
terproses dalam pengulangan*/
    if((ketemu == 0)
       &&(node->info == c)){
      ketemu = 1;
    }

    if(ketemu == 1){

     simpul *last = root-
>child;

     /*mencari simpul anak
terakhir*/
     while(last->sibling !=
      root->child){
       last = last->sibling;
     }
```

```
      if(prec == NULL){
         /*jika simpul yang
dihapus anak pertama*/
          if((node->sibling ==
last)&&
(last->sibling == root-
>child)){
         /*jika hanya ada 2
anak*/
         root->child = last;
         last->sibling = NULL;
       }
       else{
    root->child = node-
>sibling;
    last->sibling = root-
>child;
       }
    }
```

DELCHILD (3)

```
      else{
        if((prec == root->child)
&&(last->sibling == root-
   >child)){
          /*jika hanya ada 2
   simpul anak, yang dihapus anak
   kedua*/
      root->child->sibling =
   NULL;
       }
       else{
      prec->sibling = node-
   >sibling;
           node->sibling = NULL;
       }
      }

     free(node);
    }
```

```
      else{
        printf("tidak ada simpul
   anak dengan info karakter
   masukan\n");
      }
     }
   }

}
```

FINDSIMPUL (1)

```
simpul* findSimpul(char c, simpul
   *root){
  simpul *hasil = NULL;
  if(root != NULL){
    if(root->info == c){
      hasil = root;
    }
    else{
      simpul *node = root->child;
      if(node != NULL){
        if(node->sibling == NULL){
          /*jika memiliki satu anak*/
          if(node->info == c){
      hasil = node;
          }
   else{
        hasil = findSimpul(c, node);
          }
        }
```

```
else{
          /*jika memiliki banyak
   anak*/
          int ketemu = 0;
          while((node->sibling !=
   root->child)
          &&(ketemu == 0)){

            if(node->info == c){
              hasil = node;
      ketemu = 1;
            }
            else{
              hasil =
               findSimpul(c, node);
              node = node->sibling;
            }
          }
```

FINDSIMPUL (2)

```
        /*memproses simpul anak
   terakhir karena belum terproses
   dalam pengulangan*/
        if(ketemu == 0){
           if(node->info == c){
             hasil = node;
           }
           else{
             hasil =
               findSimpul(c, node);
           }
         }
       }
     }
   }

  return hasil;
}
```

PREORDER

```
void printTreePreOrder(simpul *root){
  if(root != NULL){
    printf(" %c ", root->info);
    simpul *node = root->child;

    if(node != NULL){
       if(node->sibling == NULL){
        /*jika memiliki satu anak*/
        printTreePreOrder(node);
       }
       else{
        /*jika memiliki banyak anak*/

        /*mencetak simpul anak*/
        while(node->sibling !=
          root->child){
          printTreePreOrder(node);
          node = node->sibling;
        }
```

```
        /*memproses simpul anak
    terakhir karena belum
    terproses dalam pengulangan*/
          printTreePreOrder(node);
       }
     }
   }

}
```

POSTORDER

```
void printTreePostOrder(simpul
   *root){
  if(root != NULL){
    simpul *node = root->child;
    if(node != NULL){
      if(node->sibling == NULL){
        /*jika memiliki satu anak*/
        printTreePostOrder(node);
      }
      else{
        /*jika memiliki banyak anak*/

        /*mencetak simpul anak*/
        while(node->sibling !=
         root->child){
          printTreePostOrder(node);
          node = node->sibling;
        }
```

```
        /*memproses simpul anak
    terakhir karena belum terproses
    dalam pengulangan*/
          printTreePostOrder(node);
      }
    }

    printf(" %c ", root->info);
  }

}
```

COPYTREE

```
void copyTree(simpul *root1, simpul
   *root2){
  if(root1 != NULL){
    root2 = (simpul *) malloc (sizeof
   (simpul));
    root2->info = root1->info;
    root2->sibling = NULL;
    root2->child = NULL;

    if(root1->child != NULL){
       if(root1->child->sibling ==
   NULL){
         /*jika memiliki satu anak*/
         copyTree(root1->child, root2-
   >child);
       }
      else{
         /*jika memiliki banyak
   anak*/
         simpul *node1 = root1->child;

         simpul *node2 =
            root2->child;
          while(node1->sibling !=
           root1->child){
           copyTree(node1, node2);
           node1 = node1->sibling;
           node2 = node2->sibling;
         }
         /*memproses simpul anak
   terakhir karena belum terproses
   dalam pengulangan*/
         copyTree(node1, node2);
       }
     }
  }
}
```

ISEQUAL (1)

```
int isEqual(simpul *root1, simpul *root2){
  int hasil = 1;
  if((root1 != NULL)&&(root2 != NULL)){
    if(root1->info != root2->info){
      hasil = 0;
    }
    else{
      if((root1->child != NULL)&&(root2->child != NULL)){
        if(root1->child->sibling == NULL){
          /*jika memiliki satu anak*/
          hasil =
            isEqual(root1->child,
            root2->child);
        }
```

```
    else{
          /*jika memiliki banyak anak*/
        simpul *node1 = root1->child;
        simpul *node2 = root2->child;

      while(node1->sibling != root1->child){
        if((node1 != NULL) &&(node2 != NULL)){
          hasil =
            isEqual(node1, node2);
          node1 = node1->sibling;
          node2 = node2->sibling;
    }
    else{
      hasil = 0;
          break;
    }
  }
```

ISEQUAL (2)

```
        /*memproses simpul anak
    terakhir karena belum terproses
    dalam pengulangan*/
          hasil =
              isEqual(node1, node2);
       }
      }
     }
   }
   else{
     if((root1 != NULL)||
        (root2 != NULL)){
       hasil = 0;
     }
   }

   return hasil;

}
```

MAIN (1)

```
int main(){
  tree T;
  makeTree('A', &T);
  addChild('B', T.root);
  addChild('C', T.root);
  addChild('D', T.root);

  simpul *node =
      findSimpul('B', T.root);
  if(node != NULL){
    addChild('E', node);
    addChild('F', node);
  }

  node = findSimpul('C', T.root);
  if(node != NULL){
    addChild('G', node);
  }
```

```
  node = findSimpul('D', T.root);
  if(node != NULL){
    addChild('H', node);
    addChild('I', node);
    addChild('J', node);
  }

  node = findSimpul('J', T.root);
  if(node != NULL){
    addChild('K', node);
    addChild('L', node);
    addChild('M', node);
  }
```

MAIN (2)

```
printf("================\n");
printf("preOrder\n");
printTreePreOrder(T.root);
printf("\n================\n");
printf("postOrder\n");
printTreePostOrder(T.root);
printf("\n================\n");

tree T2;

copyTree(T.root, T2.root);
if(isEqual(T.root, T2.root) == 1){

  printf("pohon sama\n");
}
else{
  printf("pohon tidak sama\n");
}
```

```
  node = findSimpul('J', T.root);
  if(node != NULL){
    delChild('K', node);
    delChild('L', node);
    delChild('M', node);
  }

  printf("================\n");
  printf("preOrder setelah
   dihapus\n");
  printTreePreOrder(T.root);
  printf("\n================\n");

  return 0;

}
```

DAFTAR PUSTAKA

S, Rosa A. dan M. Shalahuddin. 2010. Modul Pembelajaran: Struktur Data. Modula: Bandung.