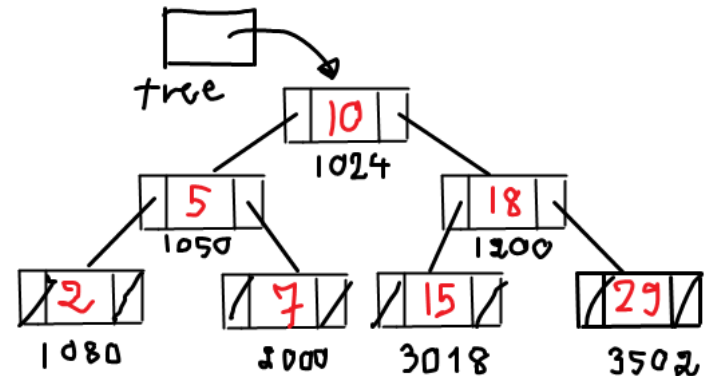




ทบทวน

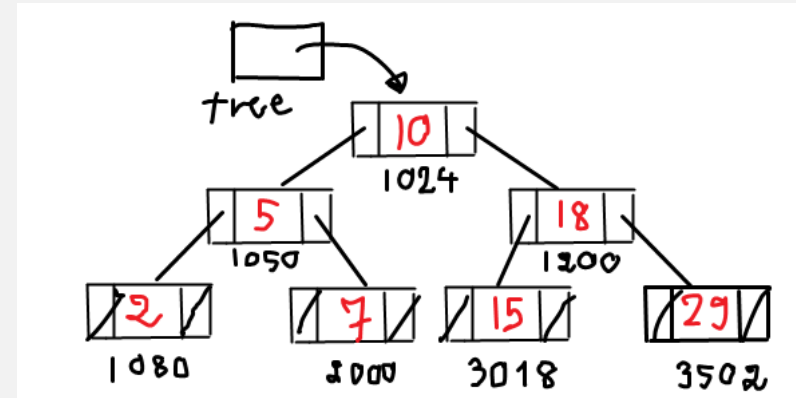
1. นิยามโครงสร้างต้นไม้
2. วิธีการ insert ของ Binary Search Tree :
 - BigO
3. การ print
 - Preorder
 - Inorder
 - Postorder





1. นิยามโครงสร้างต้นไม้

โครงสร้างข้อมูลแบบทรี (Tree) คือโครงสร้างข้อมูลแบบไม่เชิงเส้น (non-linear) ที่ประกอบด้วยโหนด (nodes) และเส้นเชื่อมโยง (edges) ระหว่างโหนดเหล่านั้น ลักษณะเด่นของทรีคือการจัดเรียงข้อมูลแบบลำดับชั้น โดยมีโหนดราก (root node) อยู่ด้านบนสุด และโหนดอื่นๆ แยกย่อยเป็นกิ่งก้านสาขา (subtrees) ลงไป





2. วิธีการ insert ของ Binary Search Tree :

- BigO

3. การ print

- Preorder
- Inorder
- Postorder



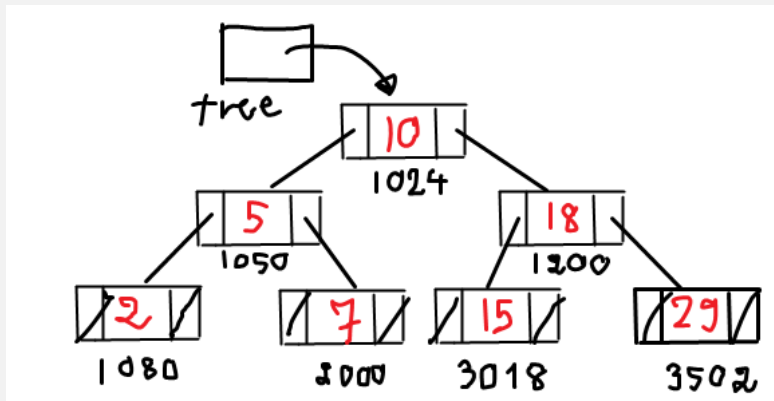
เนื้อหาที่จะเรียนในวันนี้ การลบข้อมูลในทรี

1. การ findmin
2. Delete Tree :
 - ไม่มีลูก
 - ลูก 1 ด้าน
 - ลูก 2 ด้าน
3. Expression Trees



การ findmin :

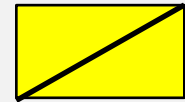
- ค่าที่น้อยที่สุดในทรีควรอยู่ด้านใด?
- การหาค่าที่น้อยที่สุดในทรี ควรจะมีเงื่อนไขอย่างไร?





```
struct node *find_min(struct node *tree)
1 {   if(tree==NULL)
2     return NULL;
3   else
4     if( tree->left == NULL )
5       return tree;
6   else
7     return (find_min(tree->left));
}
```

tree



2000

tree



2000

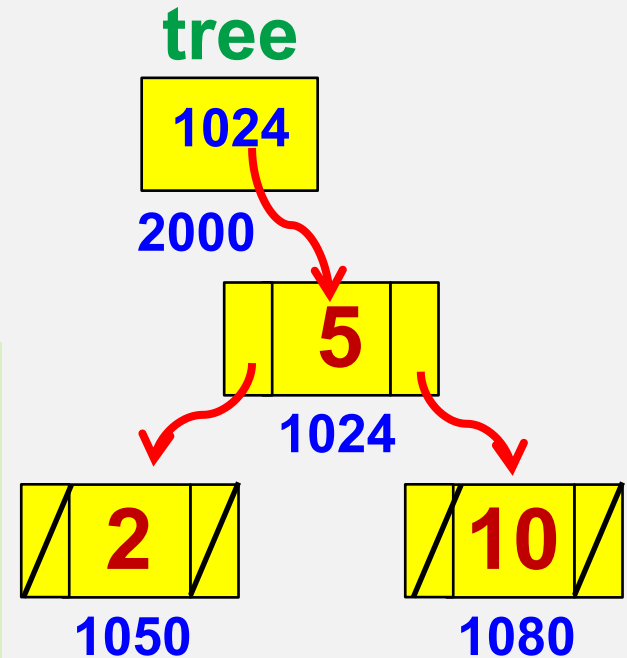


1024



```
1. struct node *find_min(struct node *tree) 1024
2. {   if(tree==NULL)
3.       return NULL;
4.   else
5.       if( tree->left == NULL )
6.           return tree;
7.       else
8.           return (find_min(tree->left));
9. }
```

```
1. struct node *find_min(struct node *tree) 1050
2. {   if(tree==NULL)
3.       return NULL;
4.   else
5.       if( tree->left == NULL )
6.           return tree;
7.       else
8.           return (find_min(tree->left));
9. }
```





tree
1024

2000



1024



1050



1080



3000

```
struct node *find_min(struct node *tree)
```

```
1 {   if(tree==NULL)
2     return NULL;
3   else
4     if( tree->left == NULL )
5         return tree;
6   else
7     return (find_min(tree->left));
}
```




Big O ของการ find min



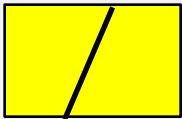
Delete

1. No Child :
2. One Child :
 - นำลูกที่เหลือมาแทนโหนดที่ถูกลบ
3. Two Childs :
 - นำลูกด้านขวาที่มีค่าน้อยที่สุดมาแทนโหนดที่ถูกลบ

Delete

1. No Child

tree

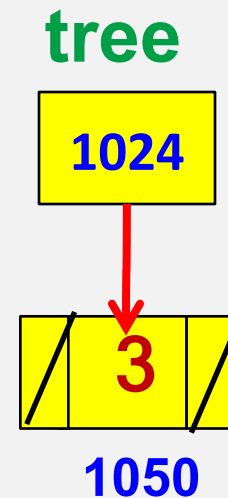
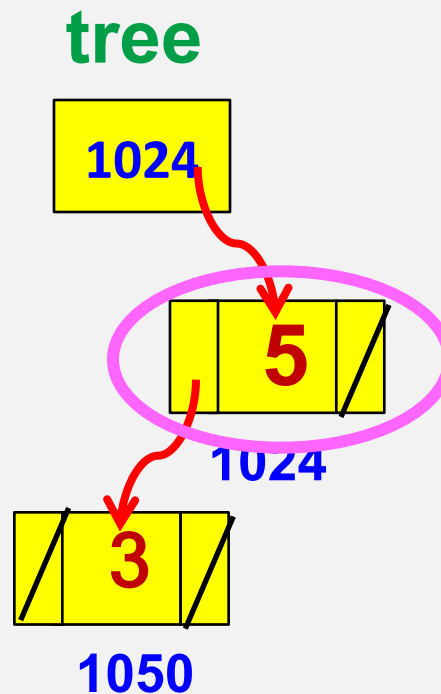
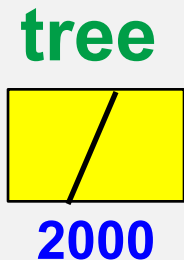
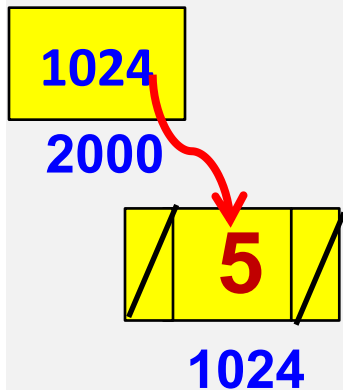


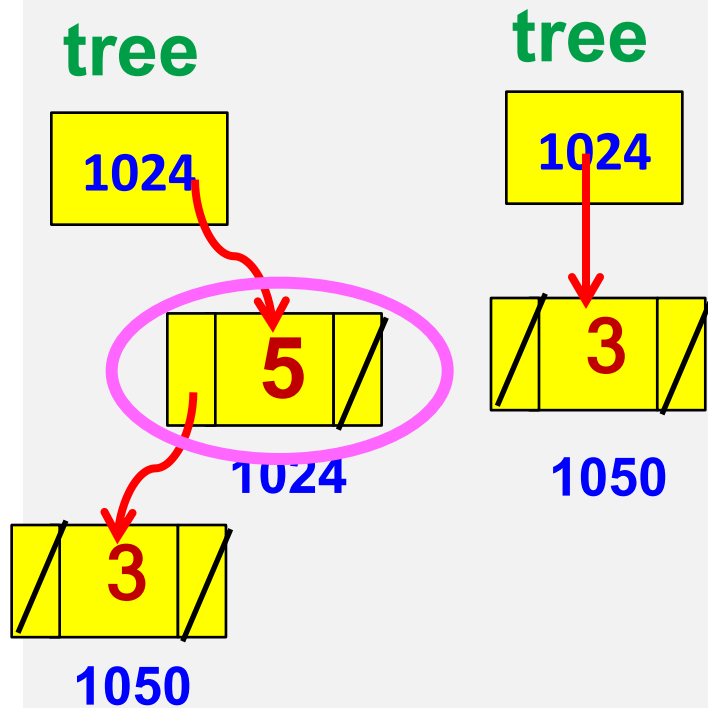
2000

```
if (tree==NULL)
    cout << "No node";
return tree;
```

Delete

2. One Child tree





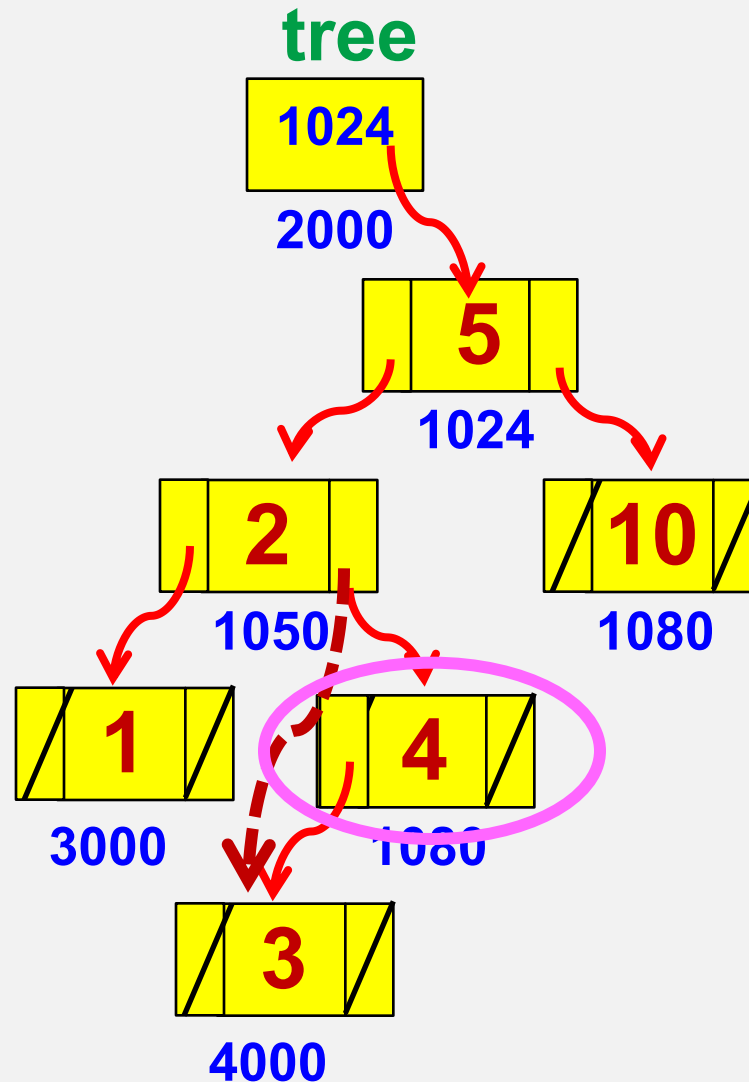
```
tmpcell=tree;
if( tree->left == NULL )
    child = tree->right;
if(tree->right ==NULL)
    child = tree->left;
delete(tmpcell);

return child;
```



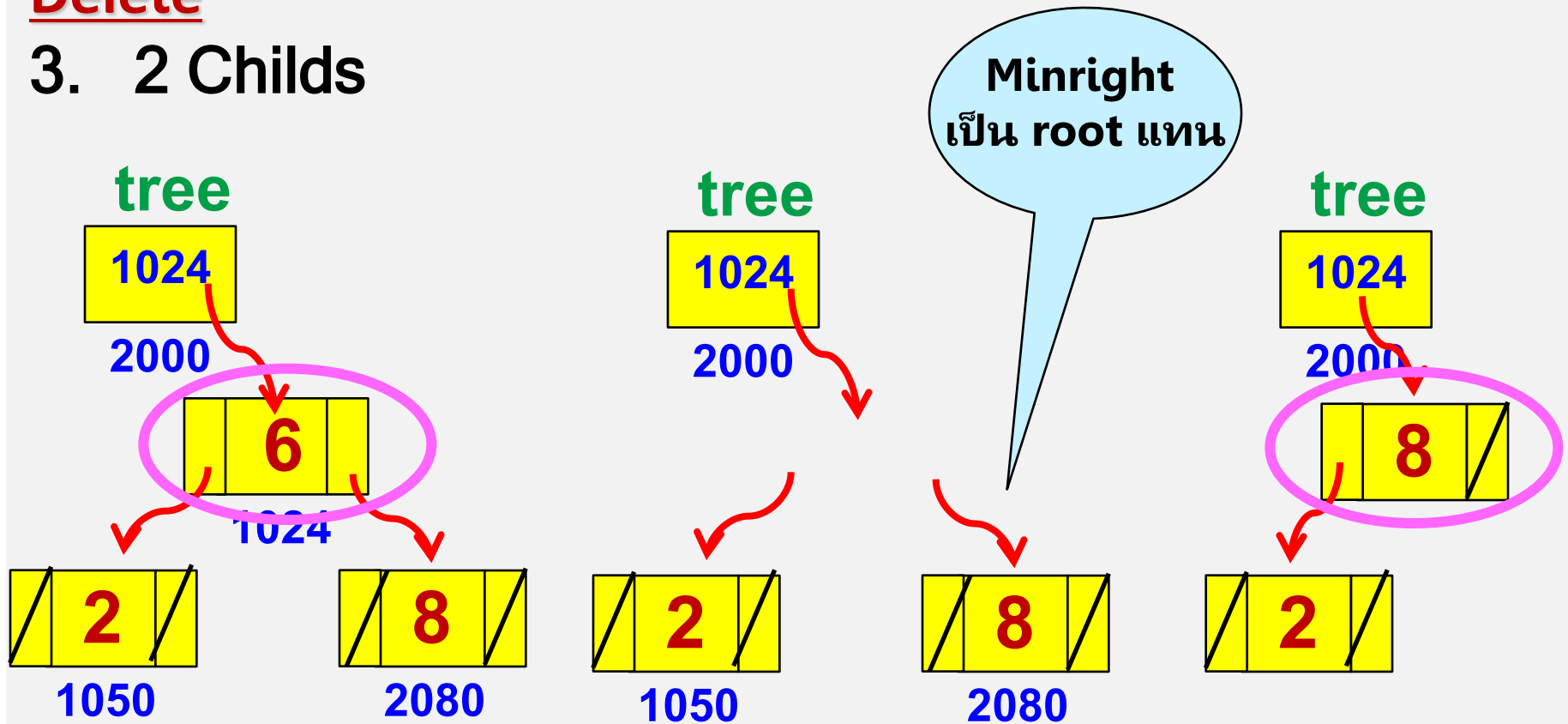
Delete

2. One Child



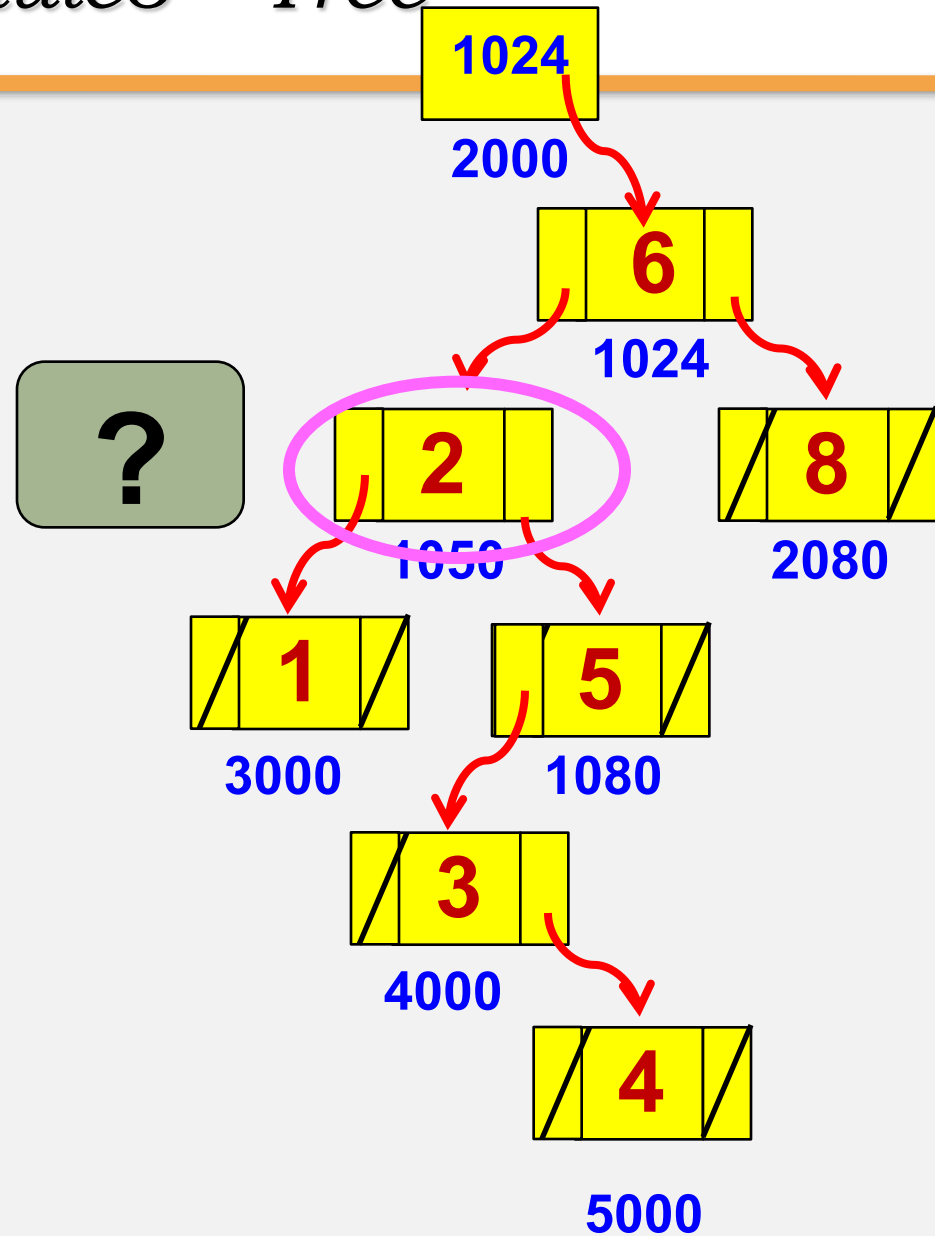
Delete

3. 2 Childs



Delete

3. 2 Childs

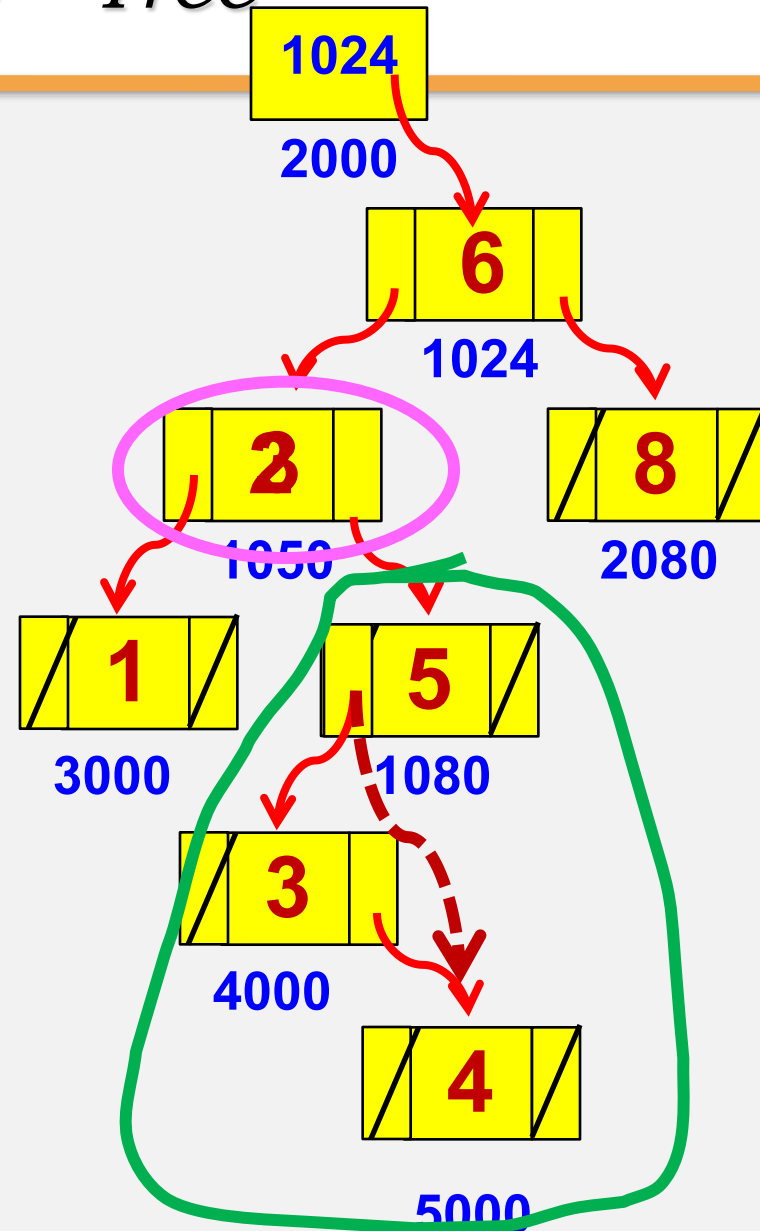




Delete

3. 2 Childs

- นำลูกที่เป็น Min right subtree มาแทนที่
- recursive delete



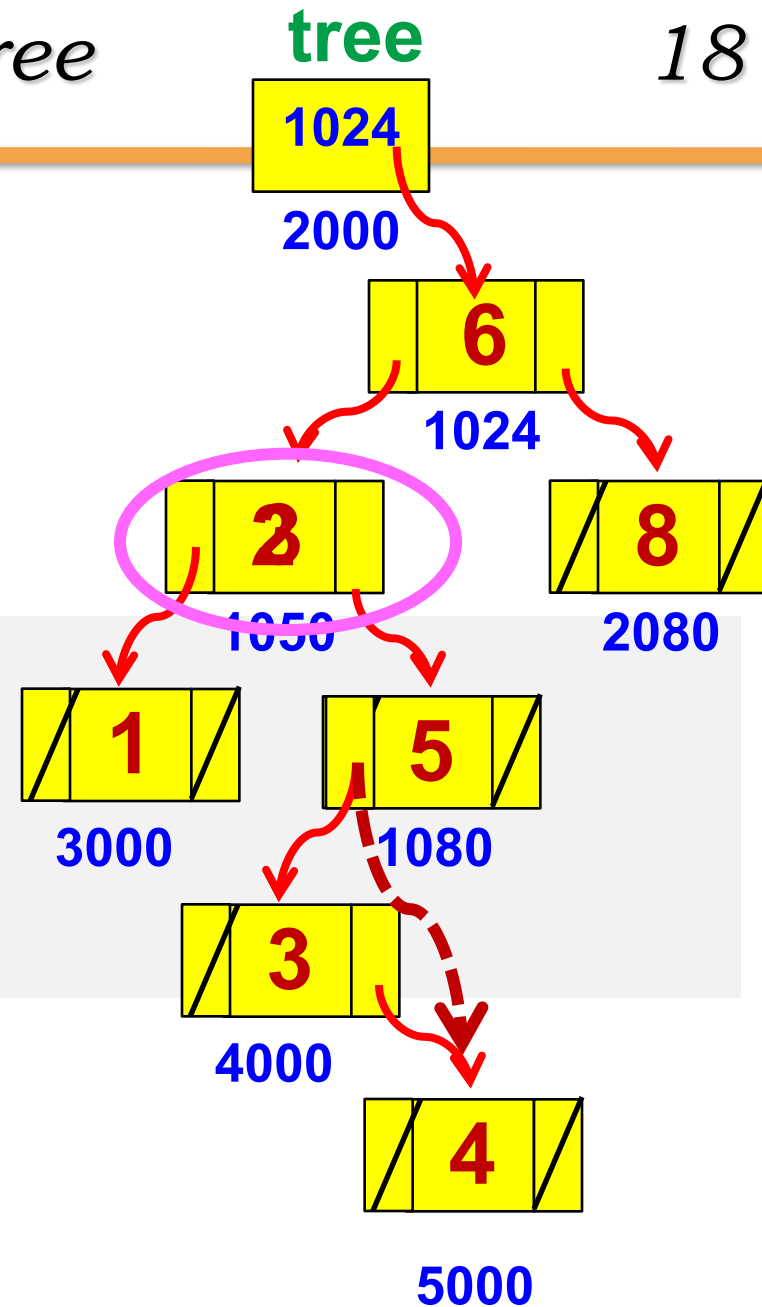


Delete

3. 2 Childs

- นำลูกที่เป็น Min right subtree มาแทนที่

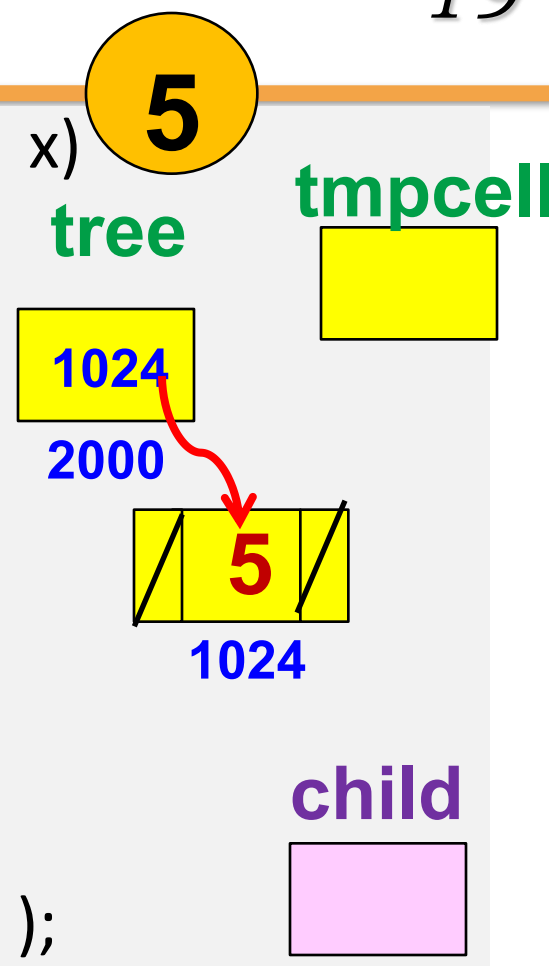
```
tmpcell=find_min(tree->right);  
tree->value = tmpcell->value;  
tree->right =  
    dTree(tree->right,tree->value );
```



```

1  struct node *dTree(struct node *tree,int x)
2  {  struct node *tmpcell, *child;
3      if ( tree==NULL)
4          printf("No Node\n");
5      else
6      {  if( x < tree->value)
7          tree->left = dTree(tree->left, x);
8          else
9              if( x > tree->value)
10                 tree->right=dTree(tree->right,x );
11                 else
12                     if( tree->left !=NULL&& tree->right!=NULL)

```



```

12 if( tree->left !=NULL&& tree->right!=NULL)
13 { tmpcell=find_min(tree->right);
14   tree->value = tmpcell->value;
15   tree->right = dTree(tree->right,tree->value );
16 }
17 else
18 { tmpcell=tree;
19   if( tree->left == NULL )
20       child = tree->right;
21   if(tree->right ==NULL)
22       child = tree->left;
23   delete(tmpcell);
24   return child;
25 }
26 } /* end else tree is not NULL */
27 return tree;
28 } /* end function */

```

5

tmpcell

1024

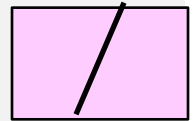
1024

2000



1024

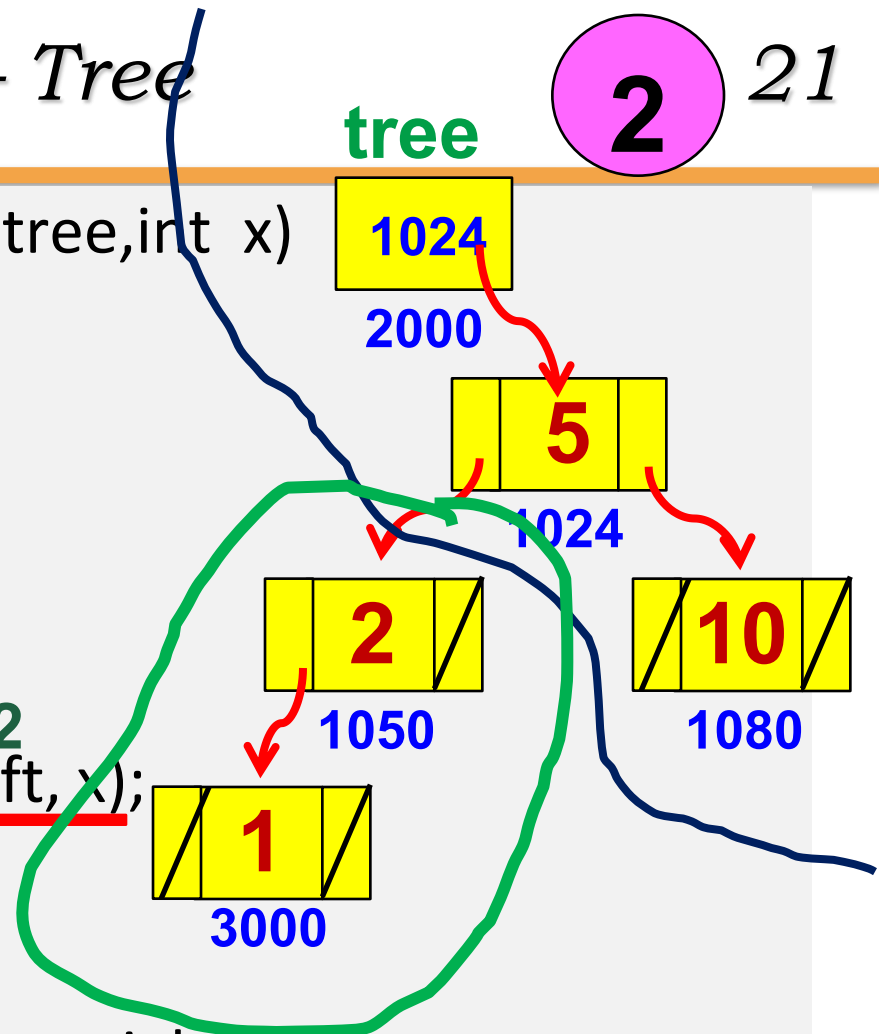
child



```

1 struct node *dTree(struct node *tree,int x)
2 { struct node *tmpcell, *child;
3   if ( tree==NULL)
4     printf("No Node\n");
5   else
6   { if( x < tree->value)
7     tree->left = dTree(tree->left, x);
8   else
9     if( x > tree->value)
10      tree->right=dTree(tree->right,x );
11   else
12     if( tree->left!=NULL && tree->right !=NULL )

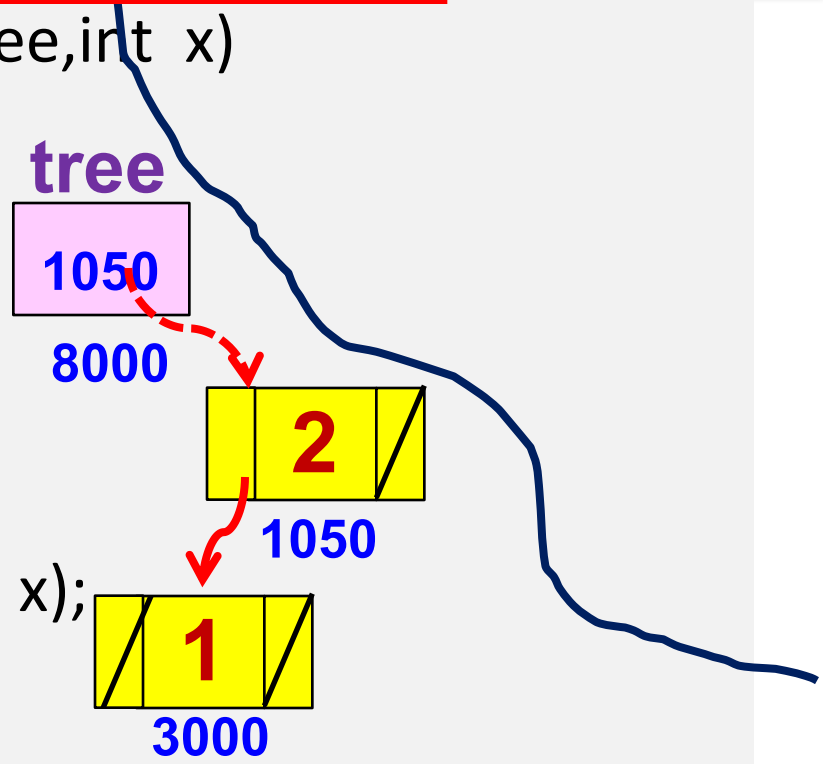
```



```

1 struct node *dTree(struct node *tree,int x)
2 { struct node *tmpcell, *child;
3   if ( tree==NULL)
4     printf("No Node\n");
5   else
6   { if( x < tree->value)
7     tree->left = dTree(tree->left, x);
8   else
9     if( x > tree->value)
10      tree->right=dTree(tree->right,x );
11    else
12      if( tree->left !=NULL && tree->right !=NULL)

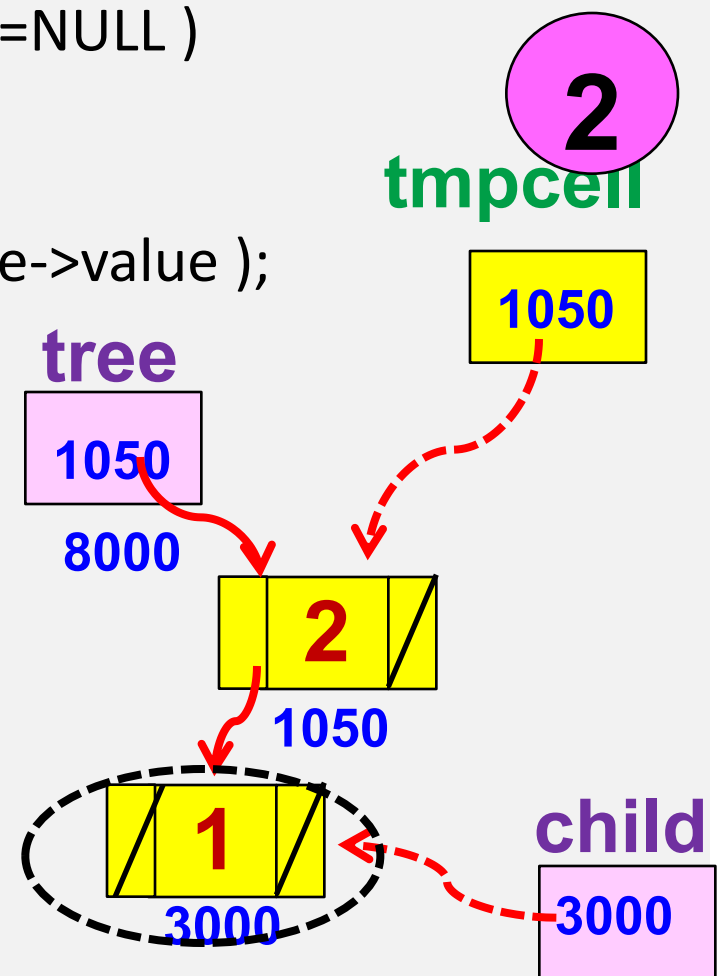
```



```

12 if( tree->left !=NULL && tree->right !=NULL )
13 { tmpcell=find_min(tree->right);
14   tree->value = tmpcell->value;
15   tree->right = dTree(tree->right,tree->value );
16 }
17 else
18 { tmpcell=tree;
19   if( tree->left == NULL )
20     child = tree->right;
21   if(tree->right ==NULL)
22     child = tree->left;
23   delete(tmpcell);
24   return child;
25 }
26 /* end else tree is not NULL */
27 return tree;
28 /* end function */

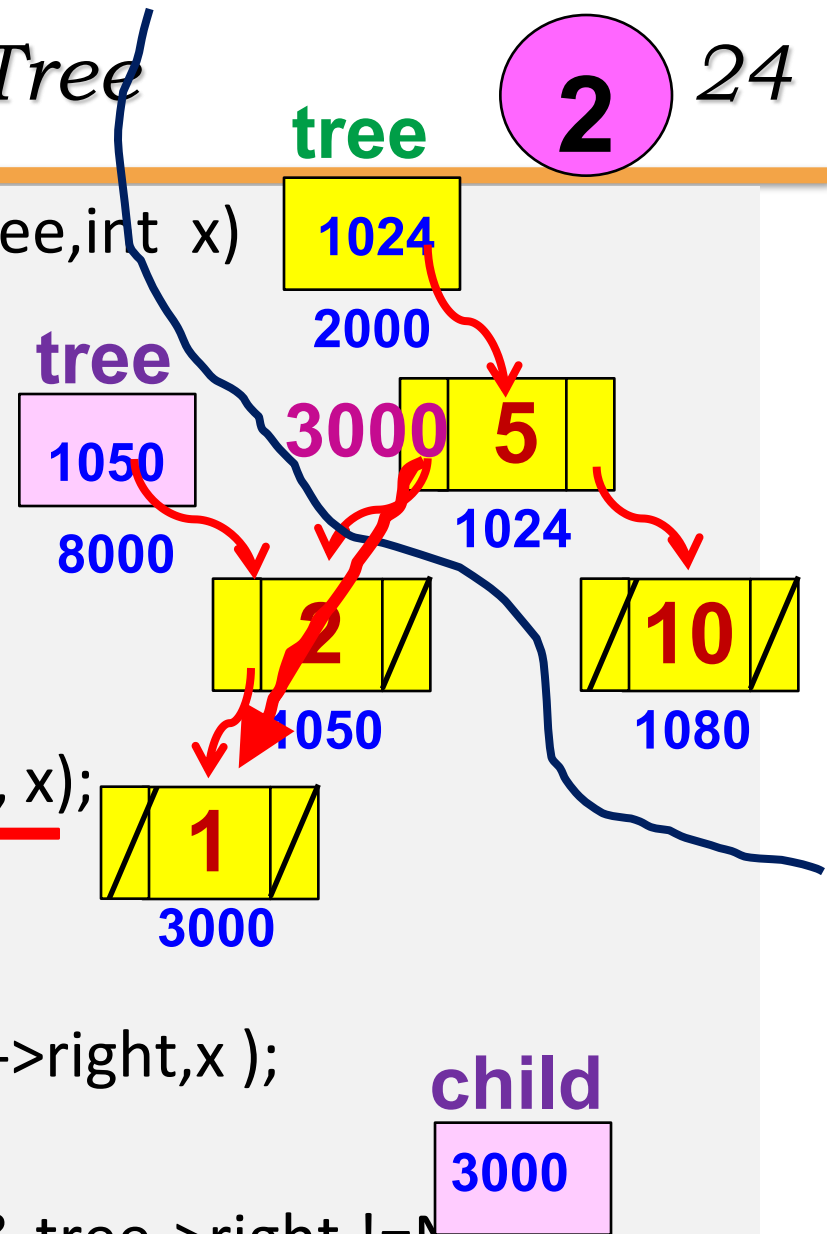
```



```

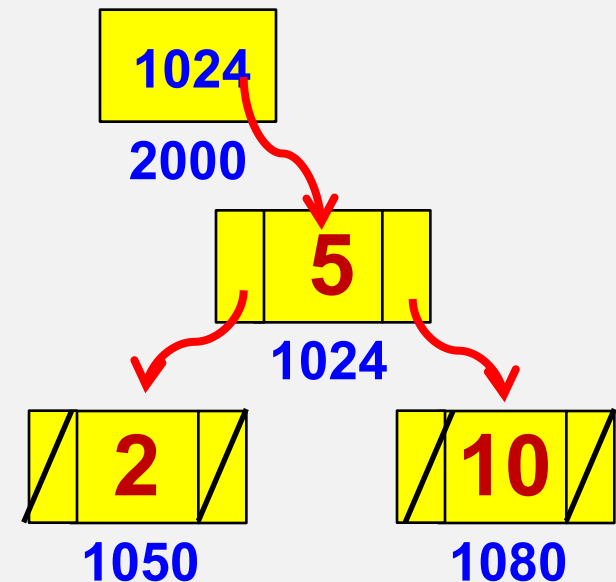
1 struct node *dTree(struct node *tree,int x)
2 { struct node *tmpcell, *child;
3   if ( tree==NULL)
4     printf("No Node\n");
5   else
6   { if( x < tree->value)
7     tree->left = dTree(tree->left, x);
8   else
9     if( x > tree->value)
10      tree->right=dTree(tree->right,x );
11    else
12      if( tree->left !=NULL && tree->right !=NULL )

```

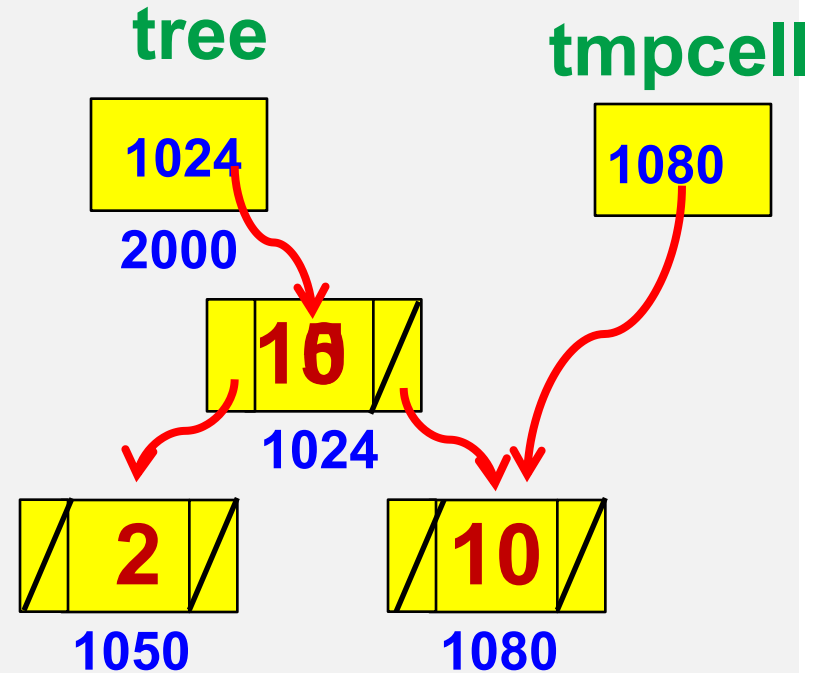




```
1 struct node *dTree(struct node *tree,int x)tree
2 { struct node *tmpcell, *child;
3   if ( tree==NULL)
4     printf("No Node\n");
5   else
6   { if( x < tree->value)
7     tree->left = dTree(tree->left, x);
8   else
9     if( x > tree->value)
10      tree->right=dTree(tree->right,x );
11   else
12     if( tree->left !=NULL && tree->right !=NULL )
```



```
12 if( tree->left && tree->right)
13 { tmpcell=find_min(tree->right);
14   tree->value = tmpcell->value;
15   tree->right = dTree(tree->right,tree->value );
16 }
17 else
18 { tmpcell=tree;
19   if( tree->left == NULL )
20     child = tree->right;
21   if(tree->right ==NULL)
22     child = tree->left;
23   delete(tmpcell);
24   return child;
25 }
26 /* end else tree is not NULL */
27 return tree;
28 /* end function */
```



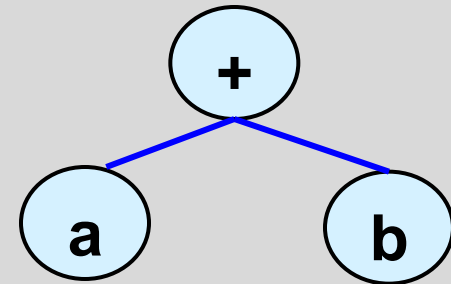


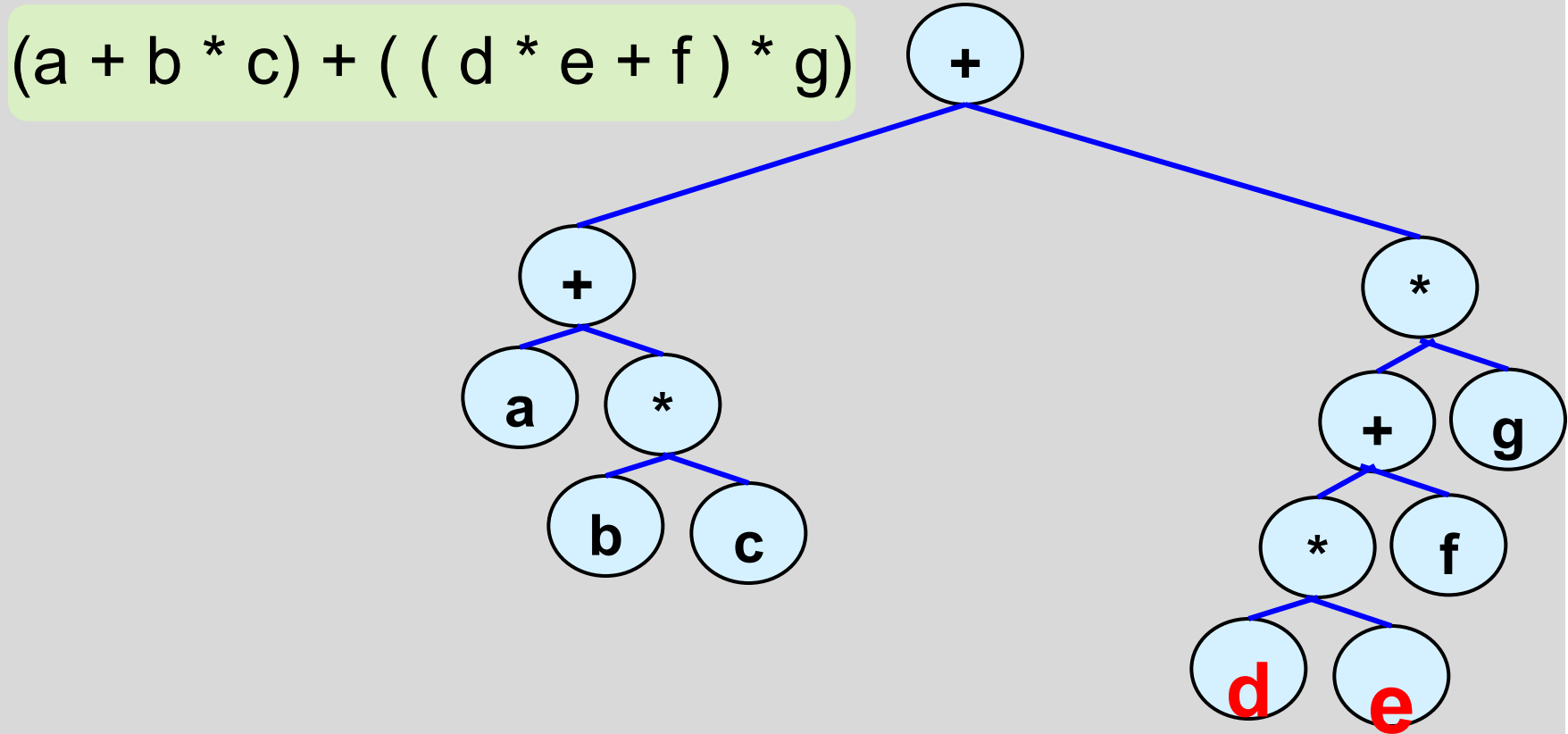
Application

Expression Tree

- Leaves are operand
- Nonleaves are operator

$a + b$

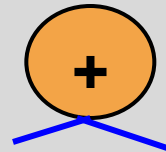
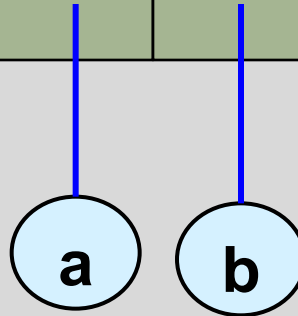
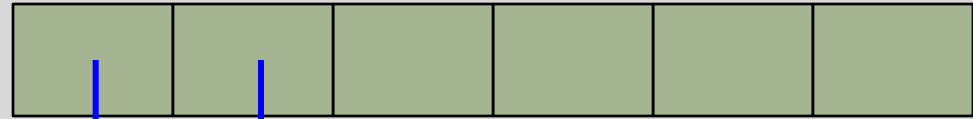




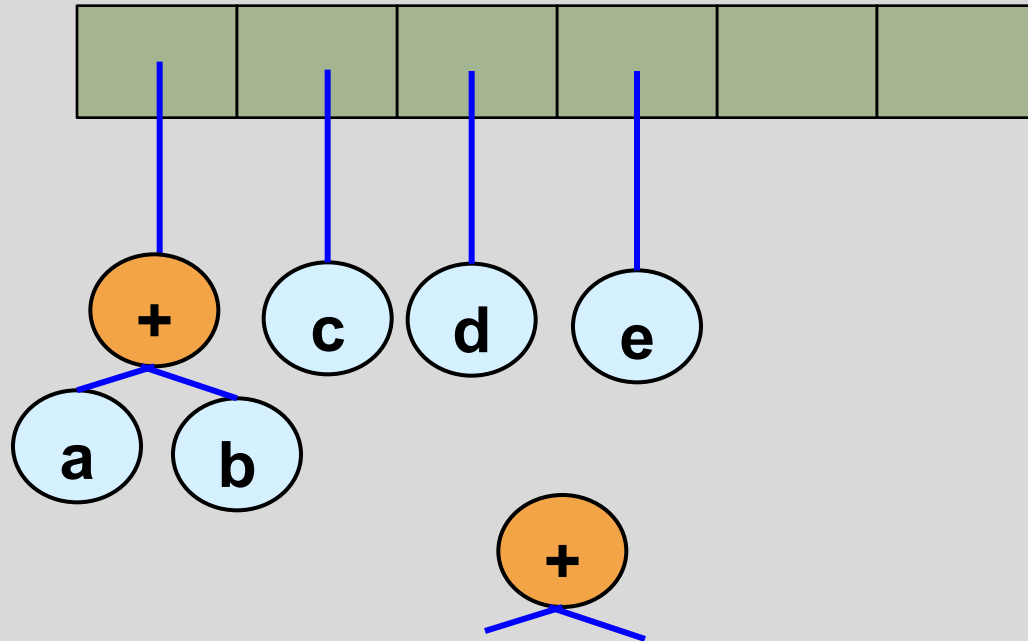


$a + b + c * (d + e)$

a b + c d e + * +

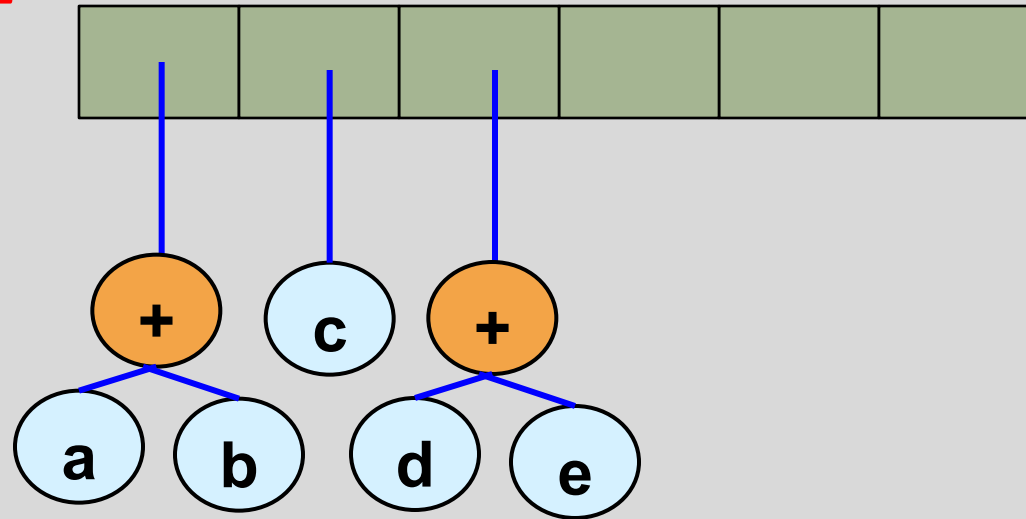


a b + c d e + * +



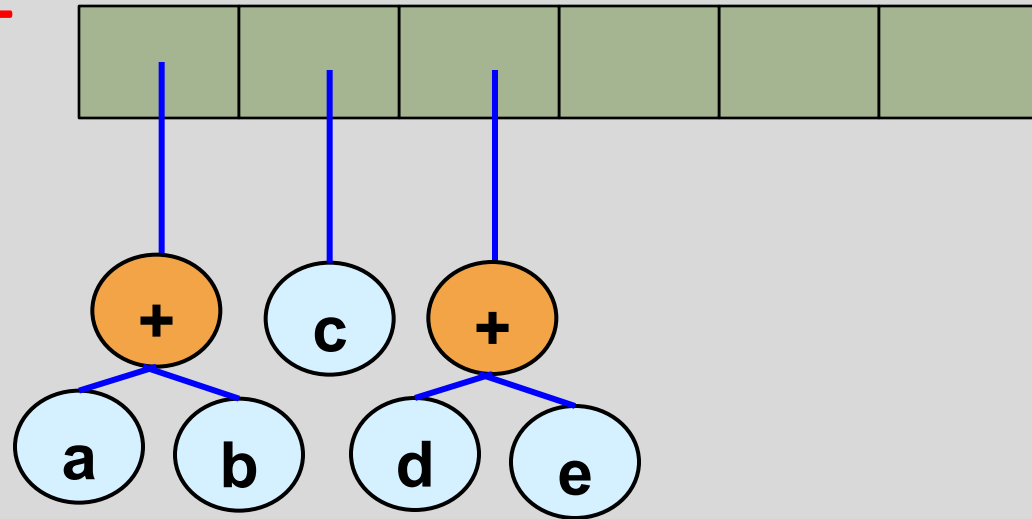


a b + c d e + * +



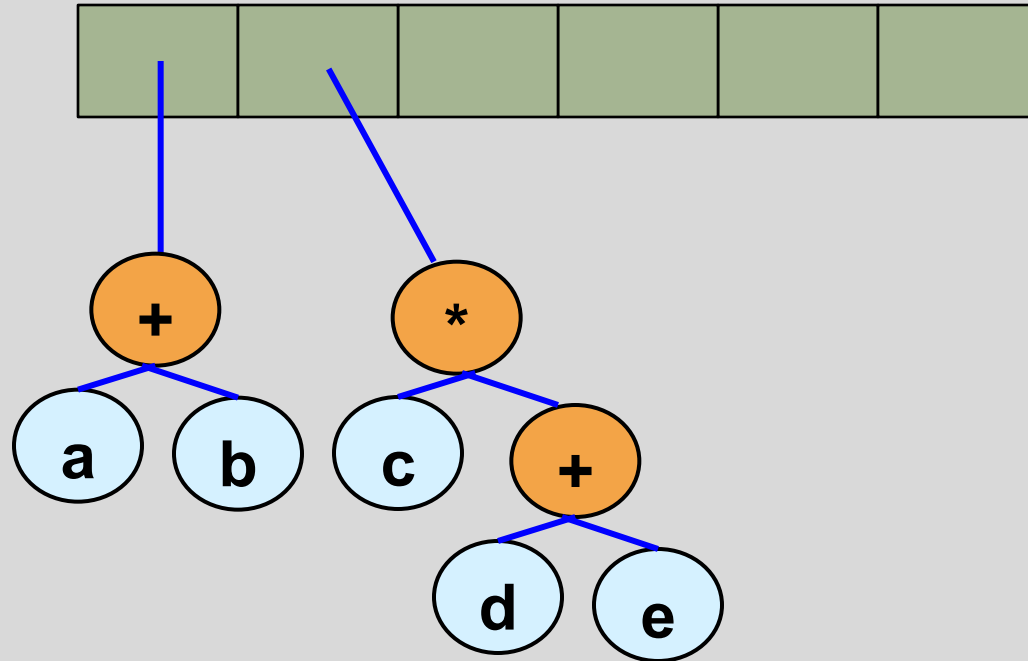


a b + c d e + * +





a b + c d e + * +





a b + c d e + * +

