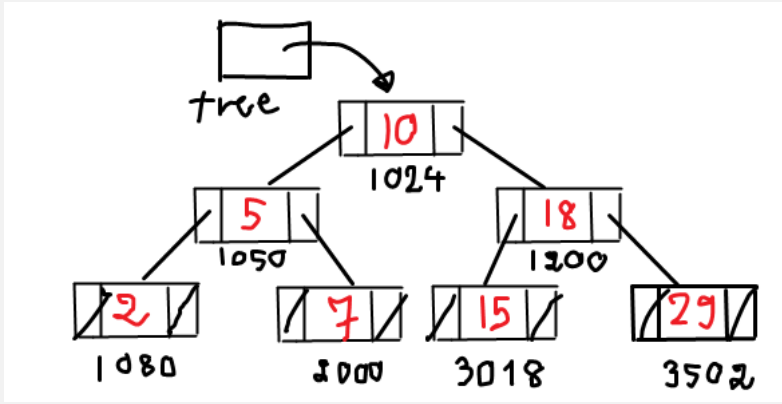
ทบทวน

✳ Binary Tree มีลูก 0, 2

ลูกซ้าย < key < ลูกขวา

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

log n



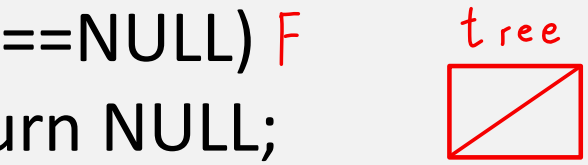
เนื้อหา

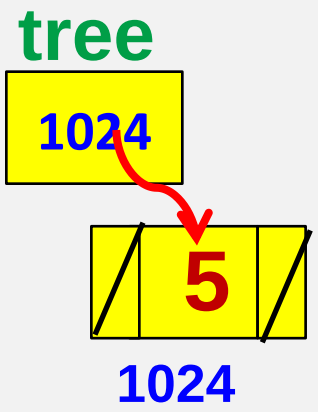
1. การ findmin * ค้นหาข้อมูลที่น้อยที่สุด
2. Delete Tree :
   - ไม่มีลูก
   - ลูก 1 ด้าน
   - ลูก 2 ด้าน
3. Expression Trees * App

1024

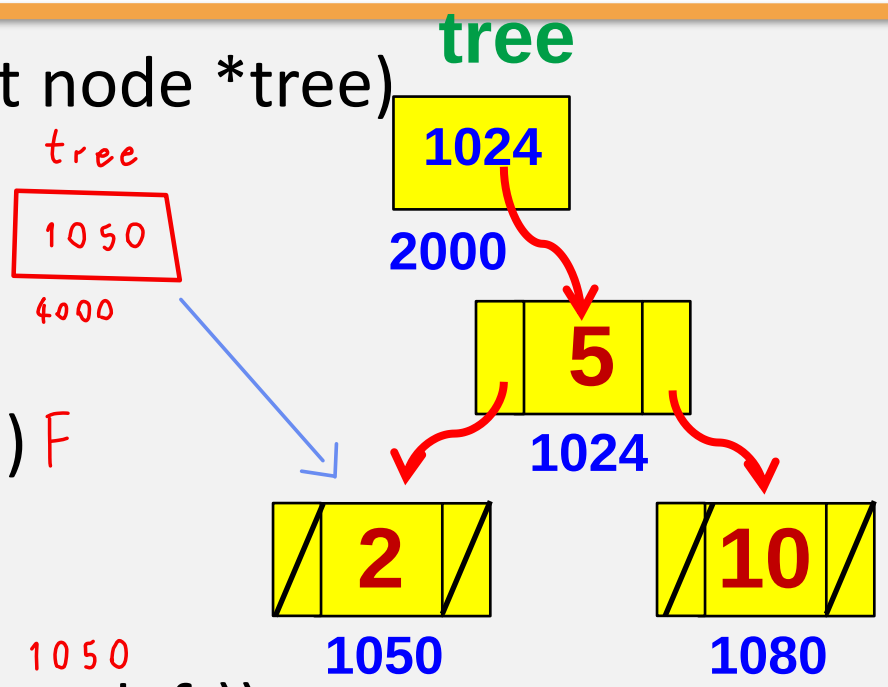
```
   struct node *find_min(struct node *tree)
1  {     if(tree==NULL) F
2           return NULL;
3        else
4           if( tree->left == NULL ) T
5              return tree;   1024 ตำแหน่ง
6           else
7              return (find_min(tree->left));
   }
```

tree



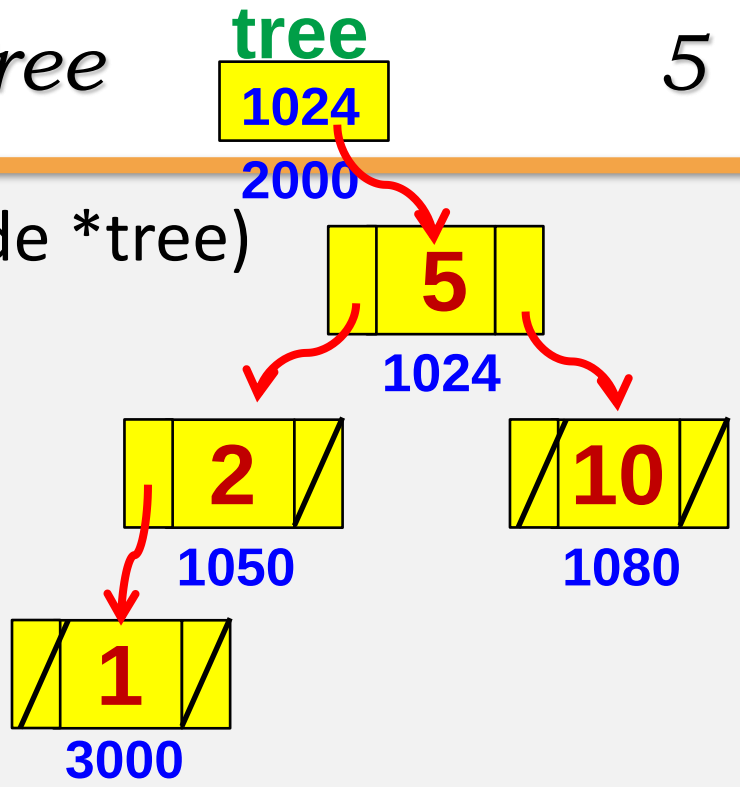


```
   struct node *find_min(struct node *tree)
1  {     if(tree==NULL)
2              return NULL;
3        else
4              if( tree->left == NULL )
5                    return tree;
6              else
7                    return (find_min(tree->left));
   }
```



	tree 1024	tree 1050
1	F	F
2		
3		
4	F	T
5		return 1050
6		
7	return 1050 ไปยัง Main	

```
   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(log n)

Big O ของการ find min

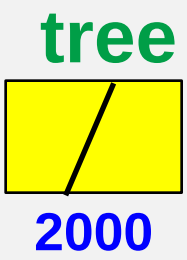
การค้นหา Big O(log n)

Delete

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

Delete

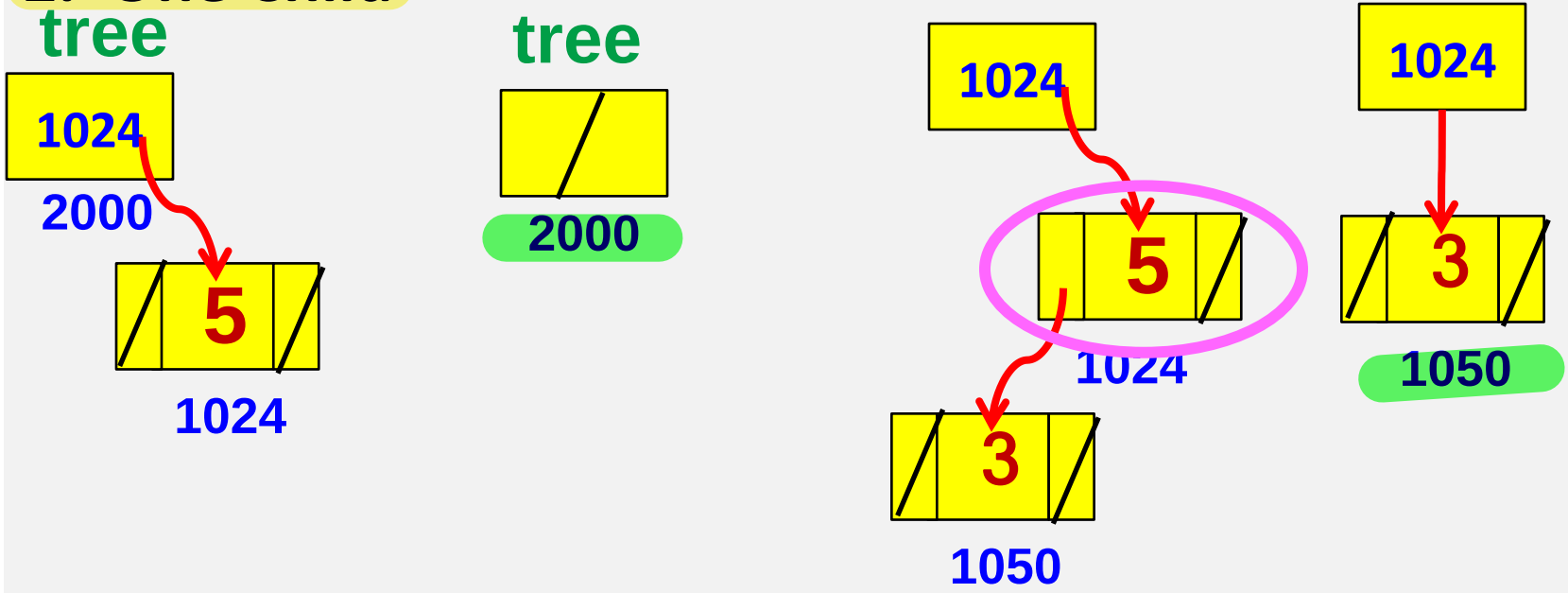
1. No Child



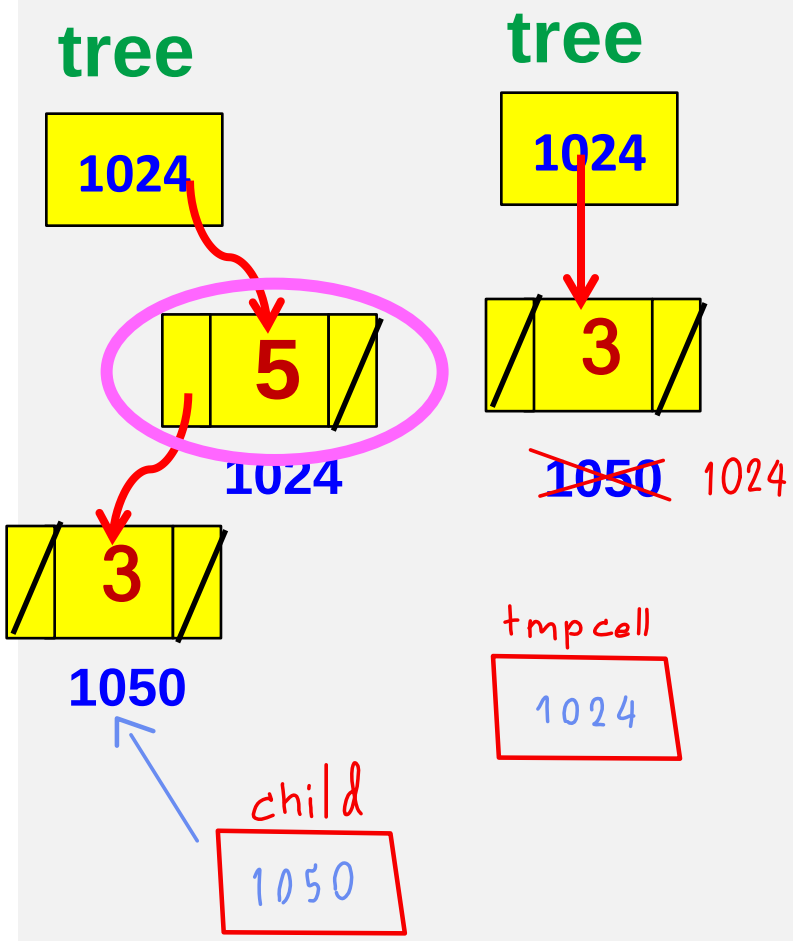

```
if (tree==NULL)
    cout << “No node”;
return tree;
```

Delete

2. One Child



* ลบ 5 เอาลูกของ 5 มาแทนและ return กลับไป



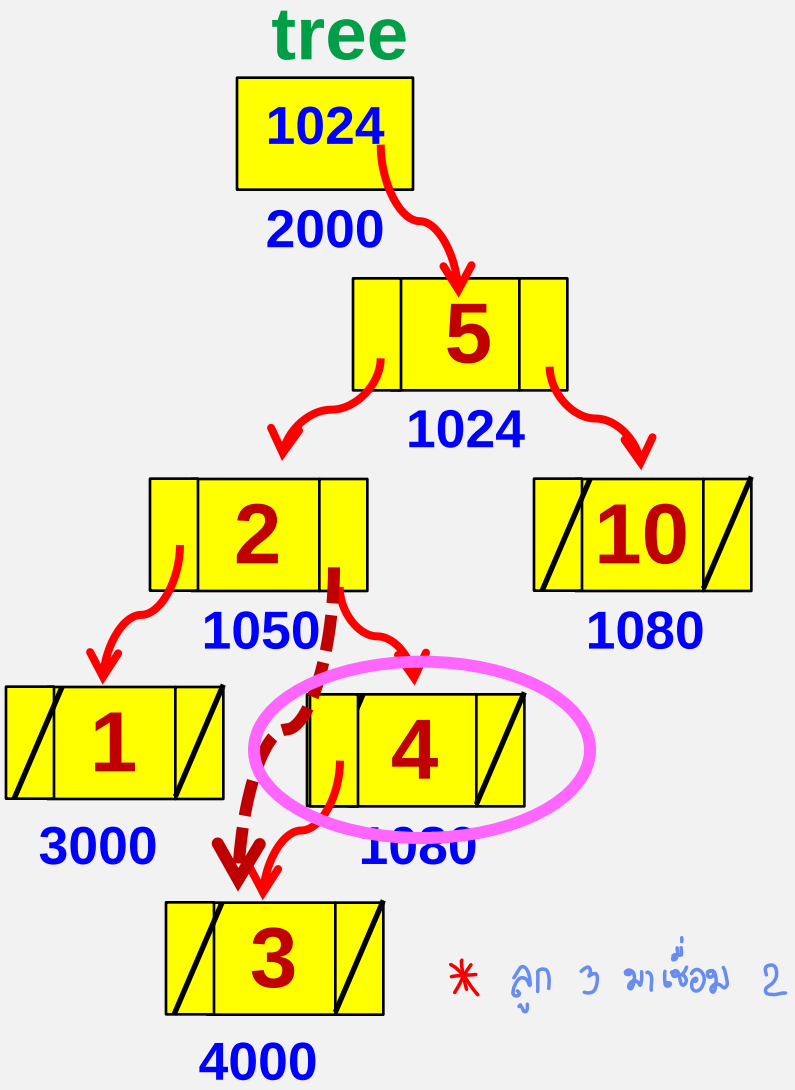
* ลบได้ทุกกรณี ซ้าย, ขวา, ไม่มีลูก

```
1 tmpcell=tree;
2 if( tree->left == NULL ) F
3         child = tree->right;
4 if(tree->right ==NULL) T
5          child = tree->left;
  delete(tmpcell);

  return child;
```

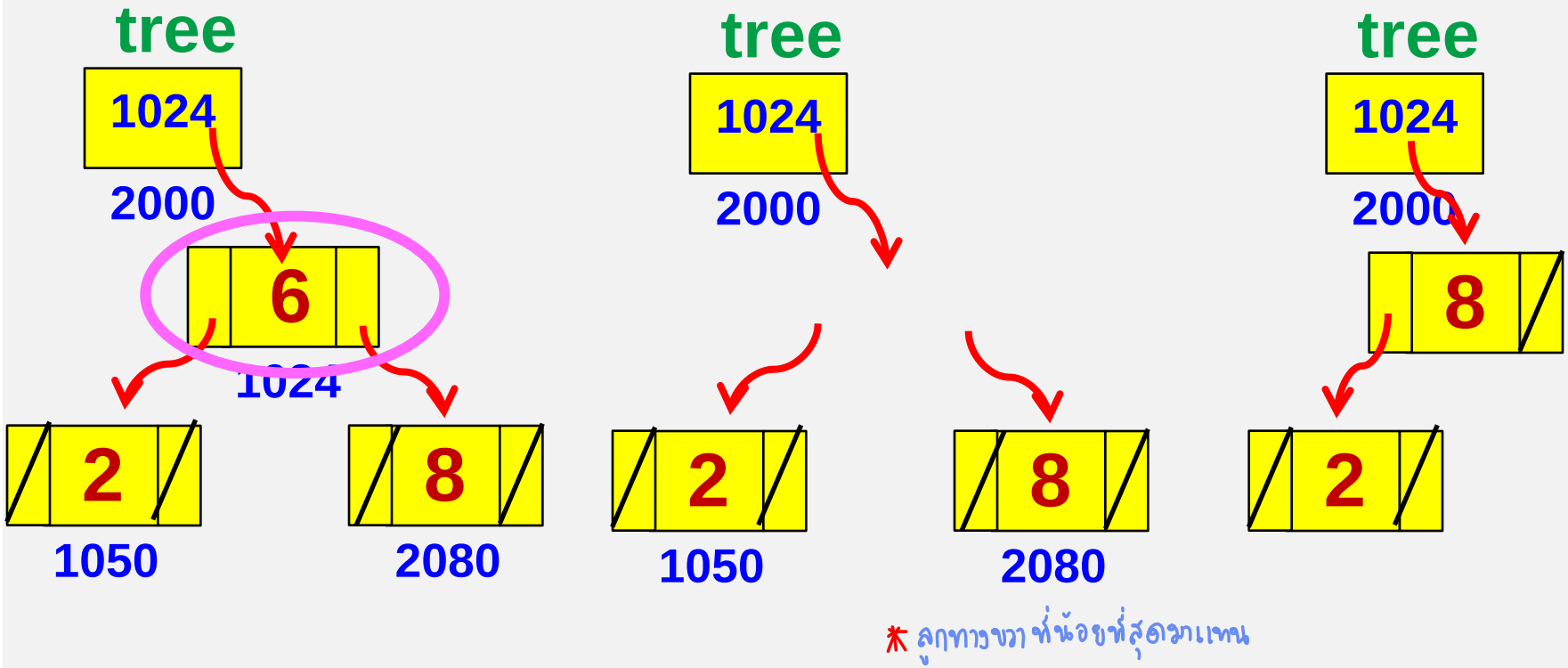
Delete

2. One Child



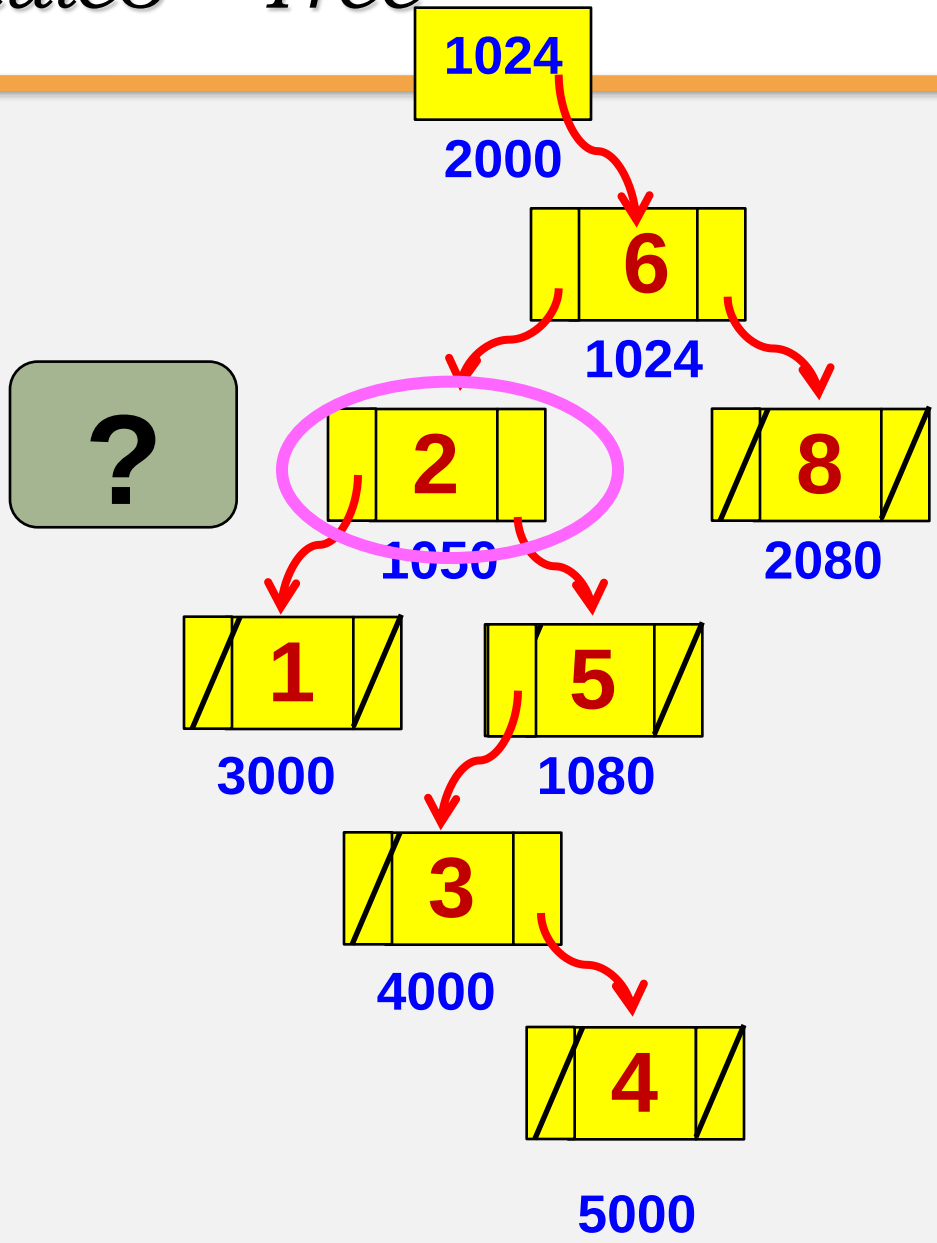
Delete

3. 2 Childs



Delete

3. 2 Childs



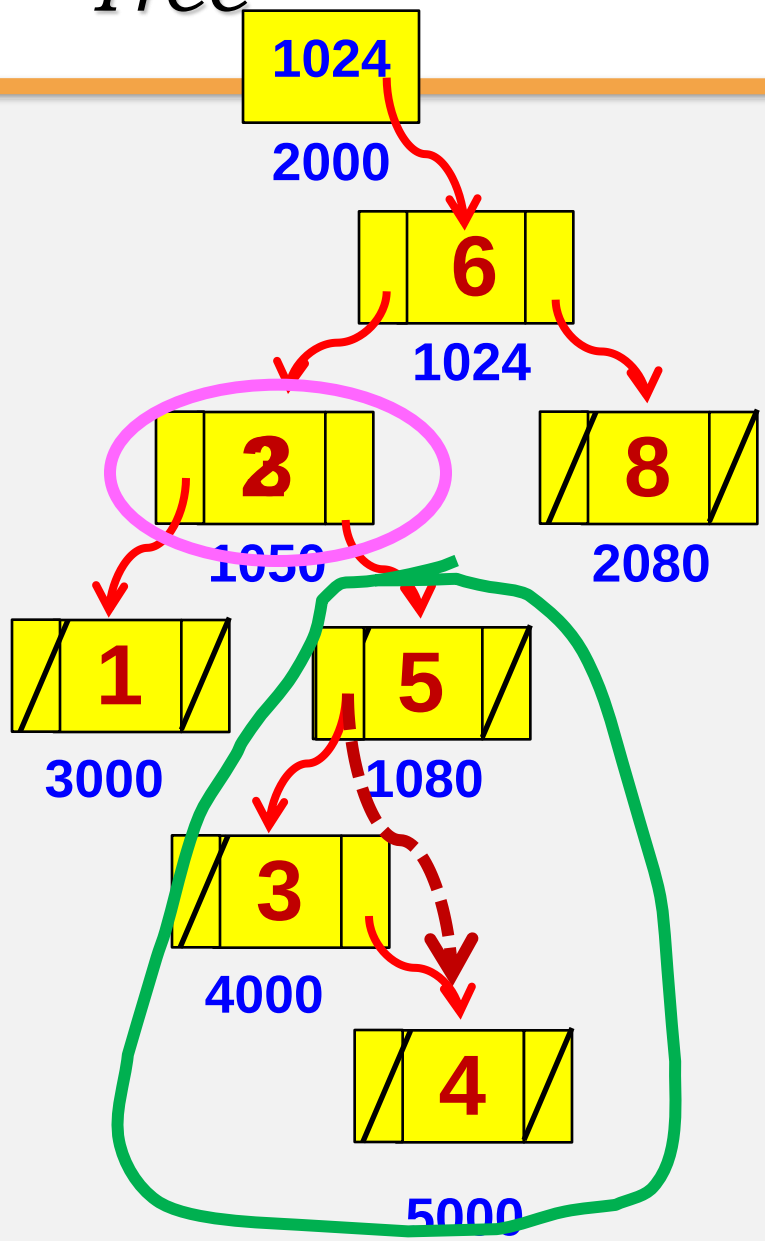
Delete

3. 2 Childs

- นำลูกทีเป็น Min right subtree มาแทนที่

- recursive delete

✳ recursive เพราะเอา 3 ไปแทน 2 แล้วแต่ 3 ตัวเก่า ยัง ค้างอยู่ จึงต้องลบ แบบ one childs จะได้ 4 มาต่อแทน 3

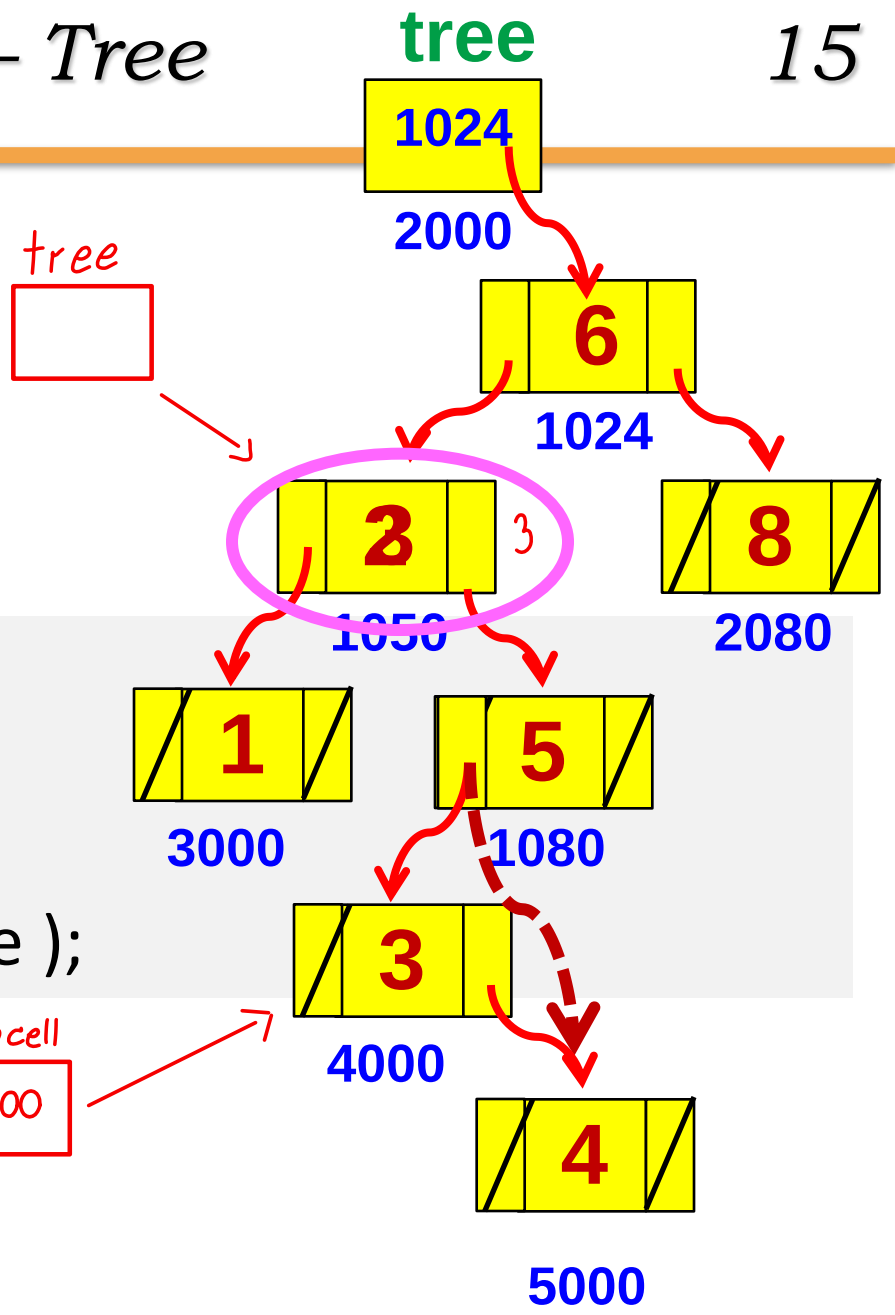


Delete

3. 2 Childs

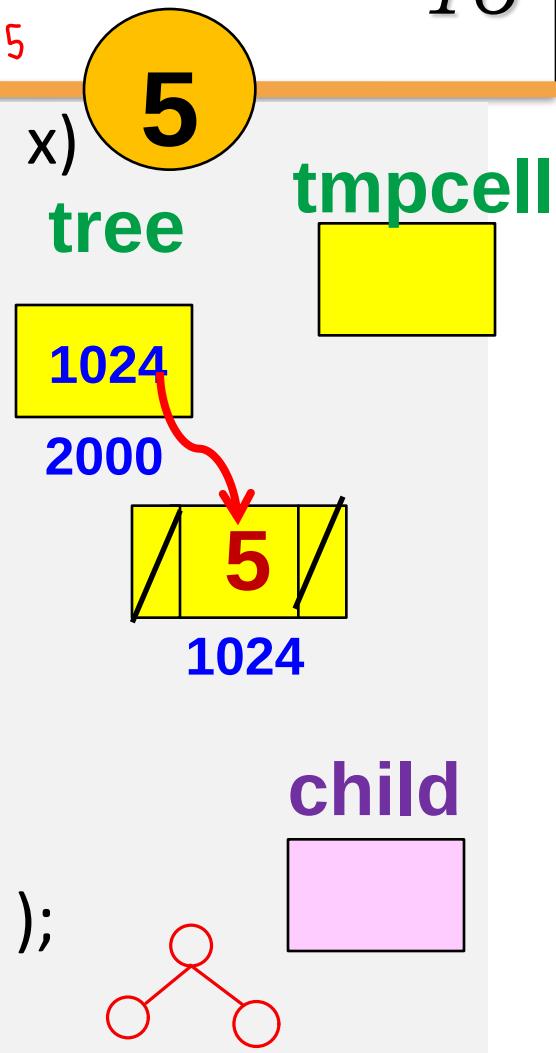
- นำลูกทีเป็น Min right subtree มาแทนที่

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

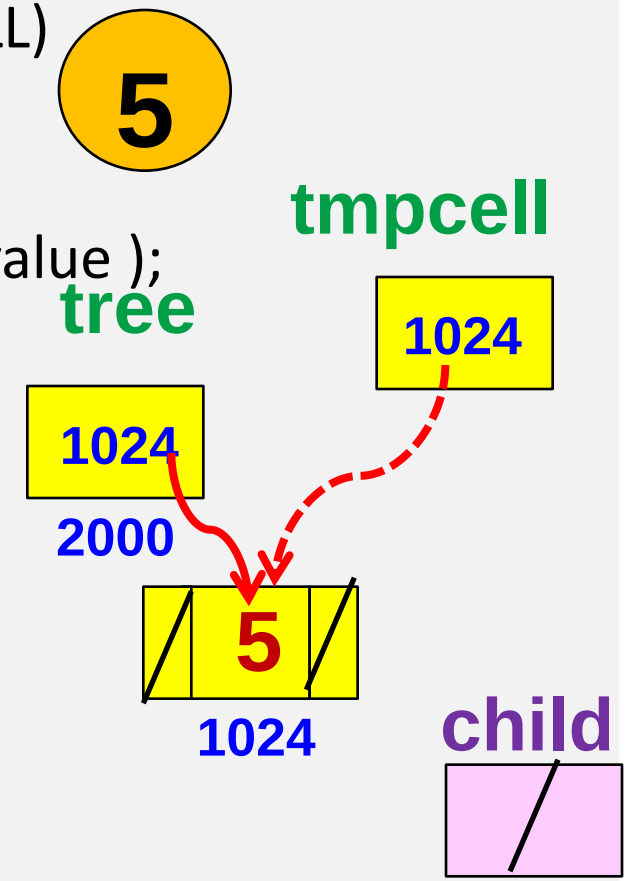


1024 5

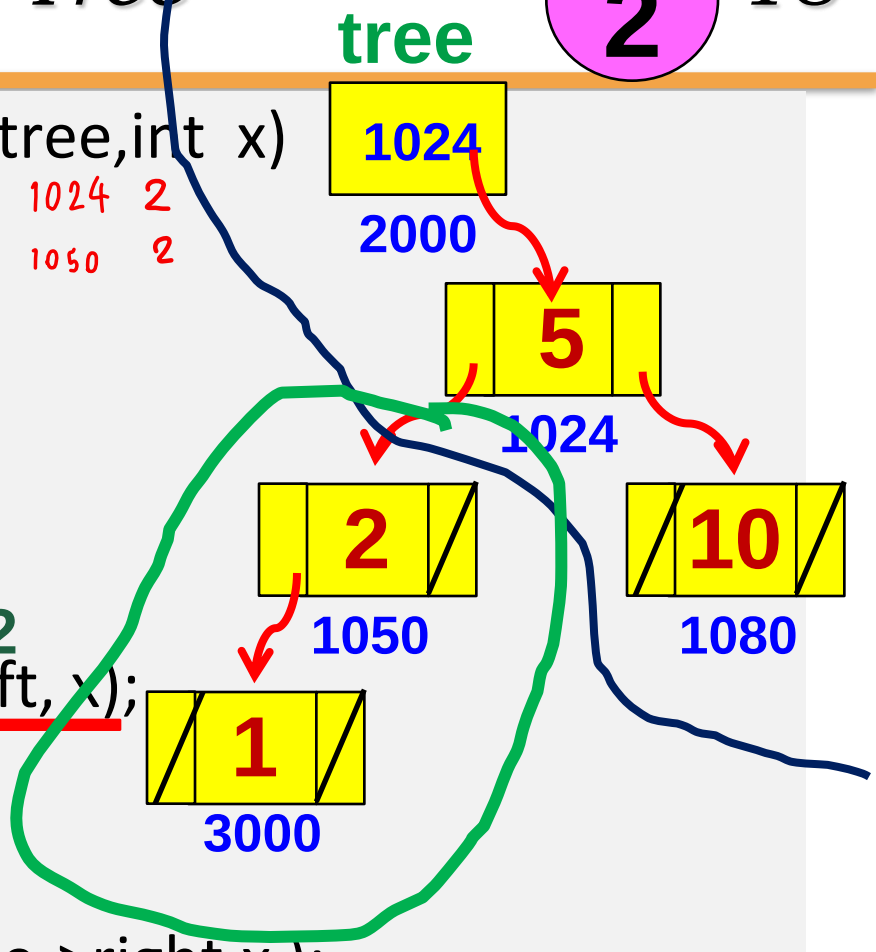
```
1    struct node *dTree(struct node *tree,int  x)
2    {   struct node *tmpcell, *child;
3 3      if ( tree==NULL) F
4                printf("No Node\n");
5 5      else
6 6      {   if( x < tree->value) F
7                tree->left = dTree(tree->left, x);
8 8        else
9 9              if( x > tree->value) F
10                   tree->right=dTree(tree->right,x );
11 11            else
12 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 );
     }
    else  โค้ดลบ ฝาก เดียว
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;  NULL * ไม่มีลูกเลย
     }
    } /* end else tree is not NULL */
27   return tree;
    } /* 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 )
```

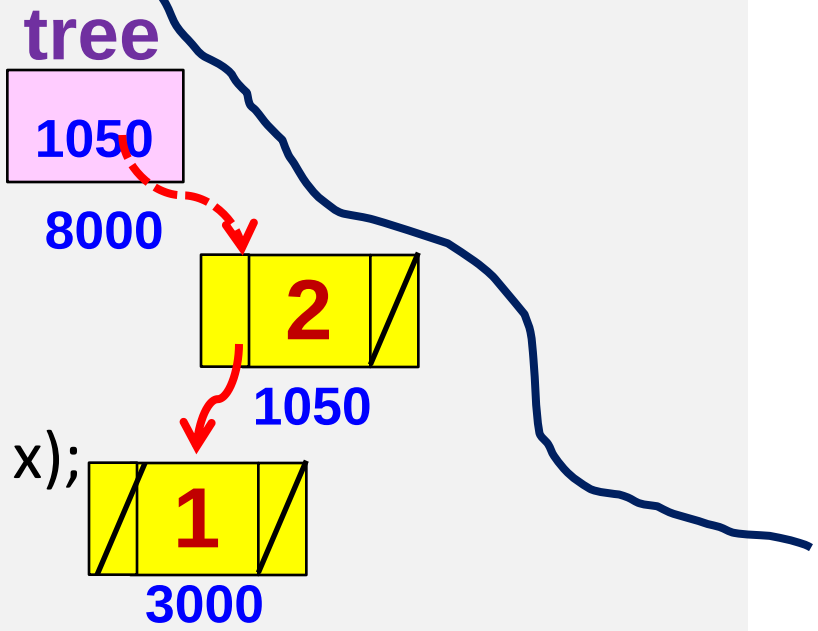


Recursive ครั้งที่ 1 1050 2

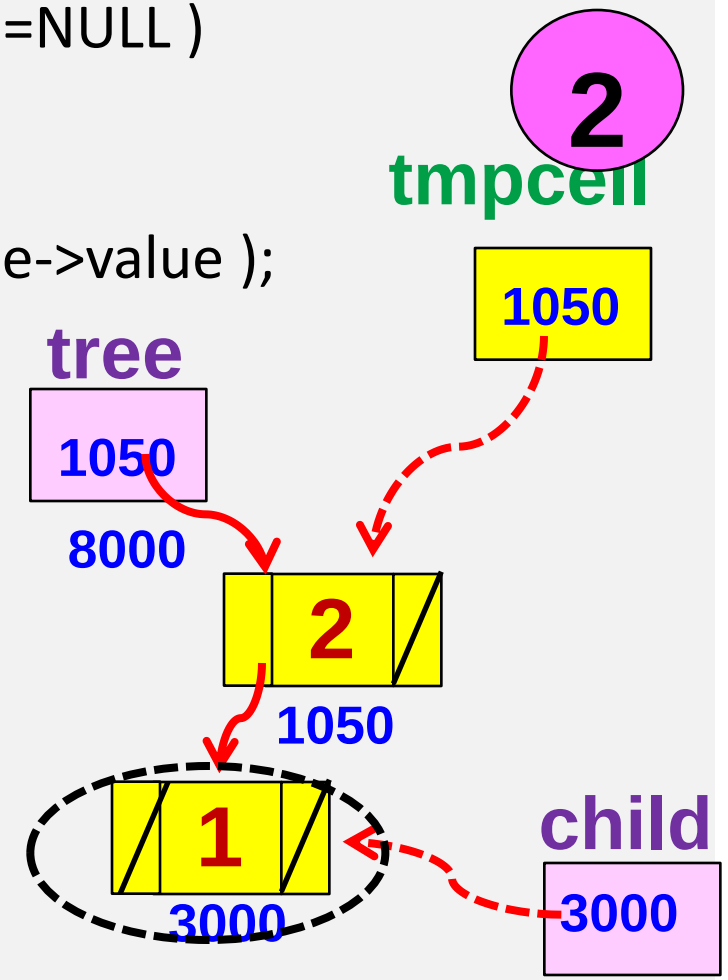
2

```
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)
```

* สร้างตัวแปรใหม่ 8 byte

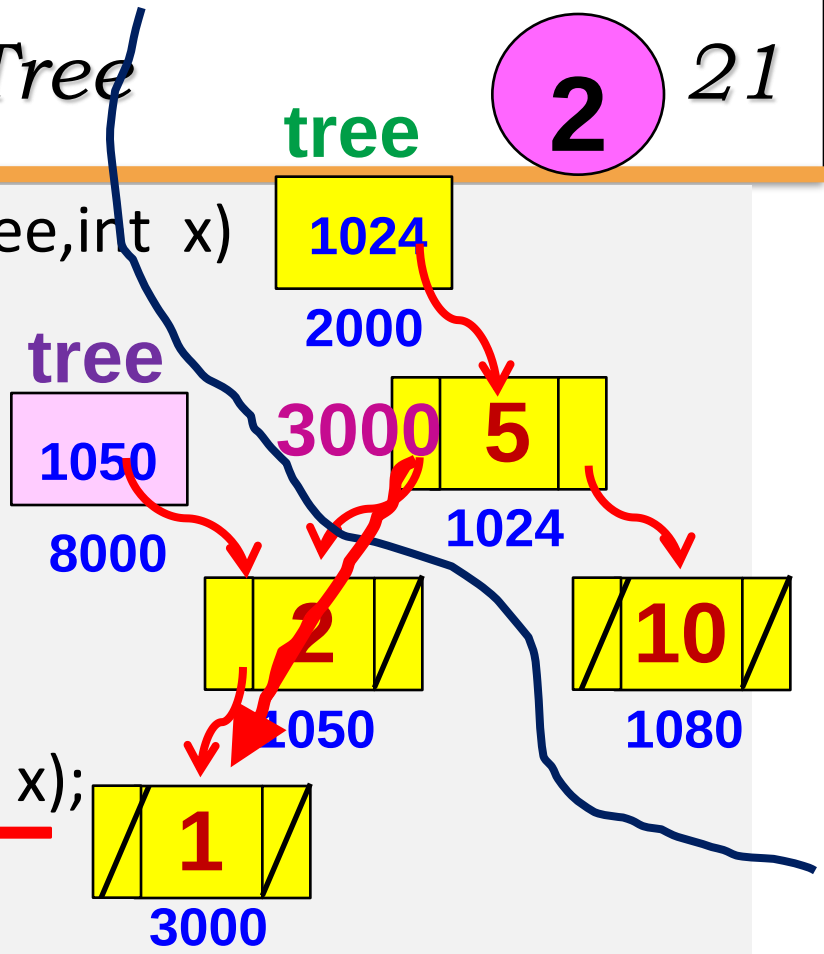


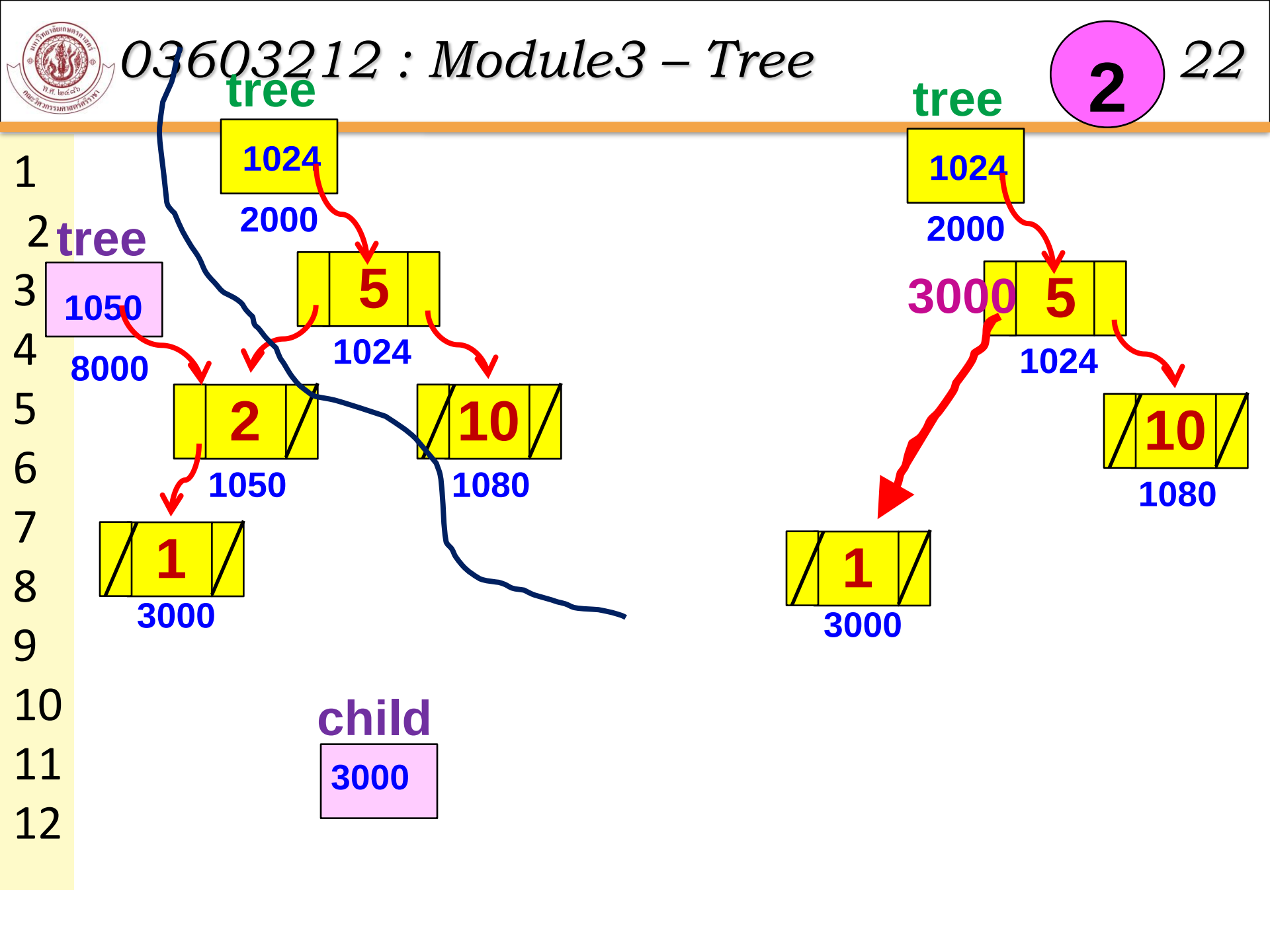
```
12  if( tree->left !=NULL && tree->right !=NULL )
13  {   tmpcell=find_min(tree->right);
14      tree->value = tmpcell->value;
15      tree->right = dTree(tree->right,tree->value );
    }
    else
18  {   tmpcell=tree;  1050
19      if( tree->left == NULL ) F
20          child = tree->right;
21      if(tree->right ==NULL) T
22          child = tree->left;  3000
23      delete(tmpcell);
24      return child;  3000 กลับไป
    }
    } /* end else tree is not NULL */
27  return tree;
    } /* 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 )
```

ไก่ ชาย (เด่นชาย)

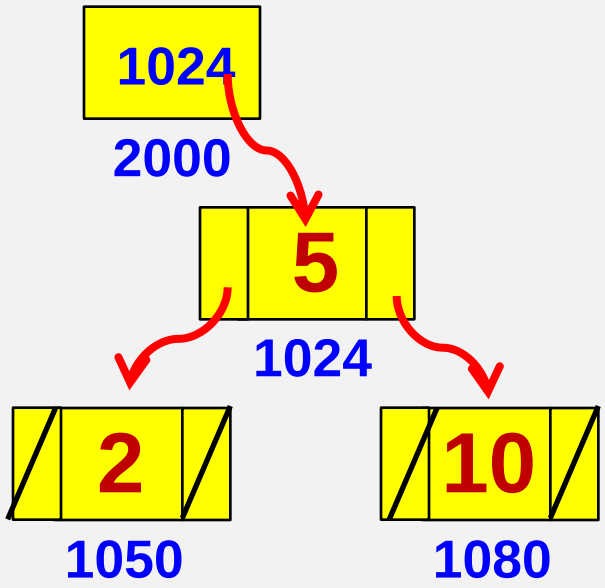



tree
1024
2000
tree
1050
8000
5
1024
2
1050
10
1080
1
3000
child
3000
tree
1024
2000
3000
5
1024
10
1080
1
3000

1024 5

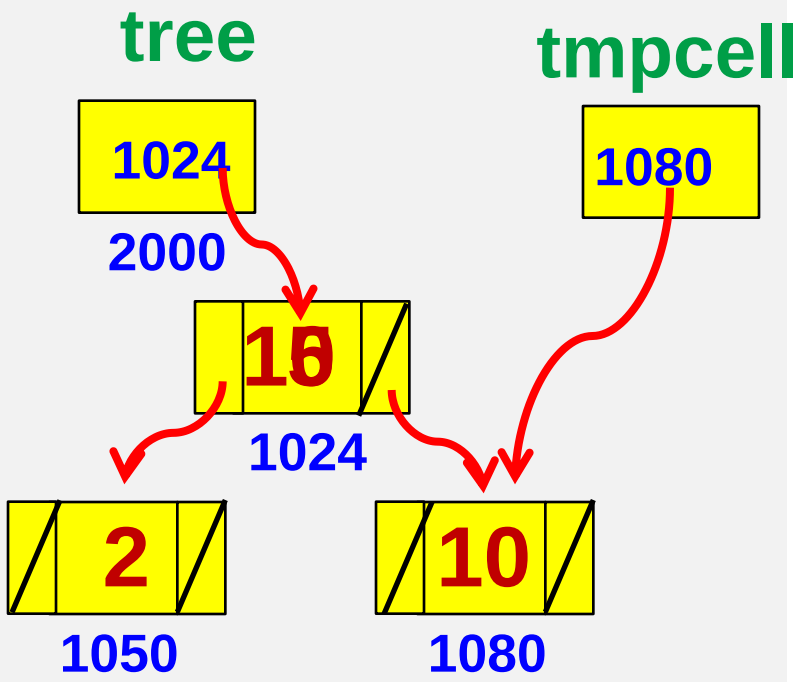
```
1  struct node *dTree(struct node *tree,int  x)
2  {   struct node *tmpcell, *child;
3       if ( tree==NULL)
4            printf("No Node\n");
5       else
6      {   if( x < tree->value)
7             tree->left = dTree(tree->left, x);
8            else
9               if( x > tree->value)
10 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

tree



```
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 );
    }
    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;
     }
    } /* end else tree is not NULL */
27  return tree;
    } /* end function */
```

5

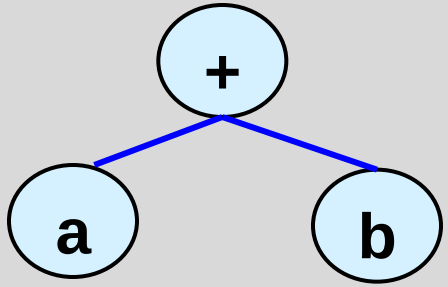


Application

Expression Tree

- Leaves are operand
- Nonleaves are operator

a + b

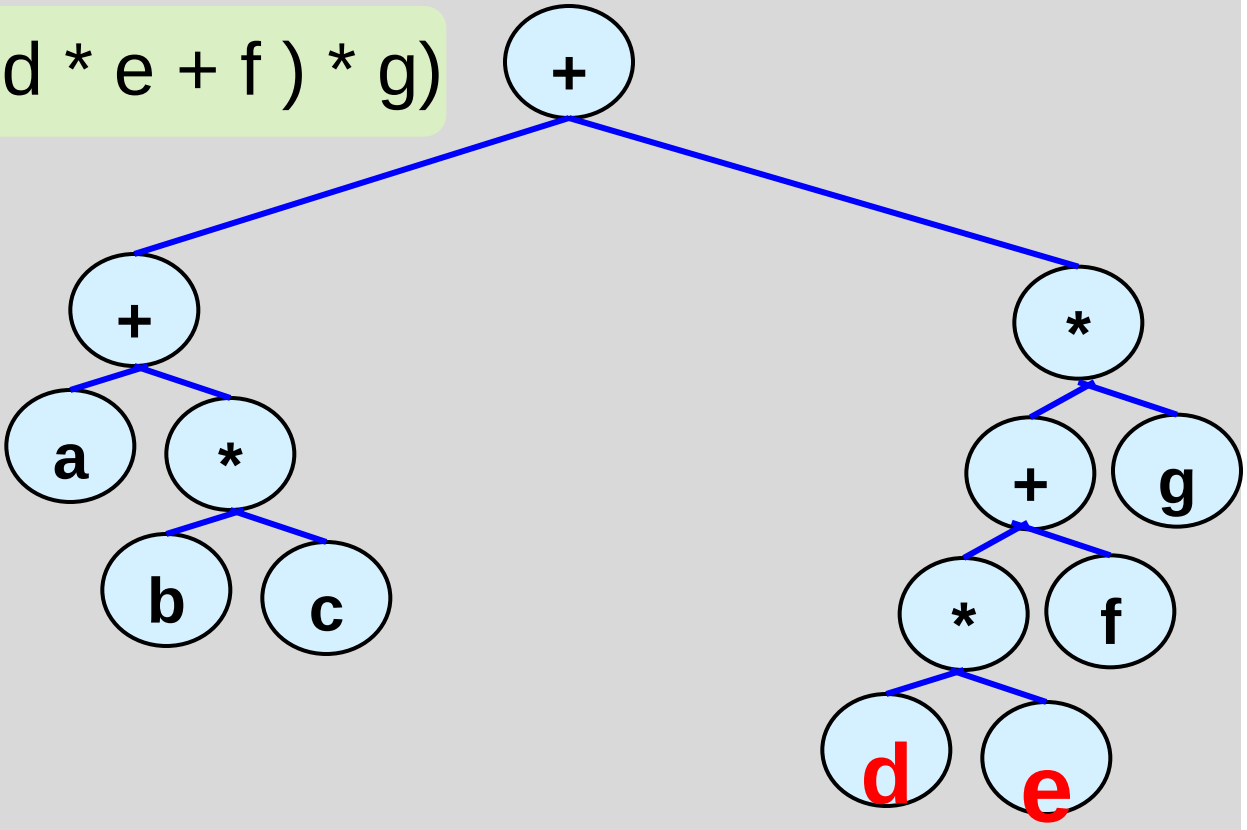


Pre + a b

In a + b

Post a b +

(a + b * c) + ((d * e + f) * g)



เปลี่ยนข้อมูล Postfix ให้เป็น expression tree

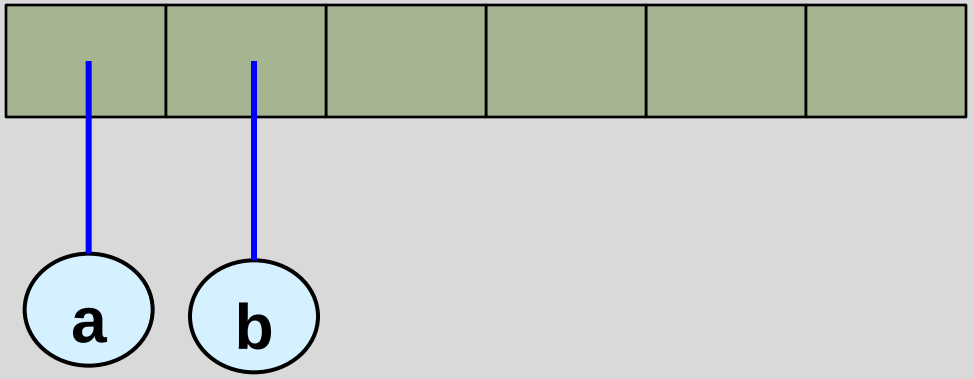
a+b + c*(d+e)

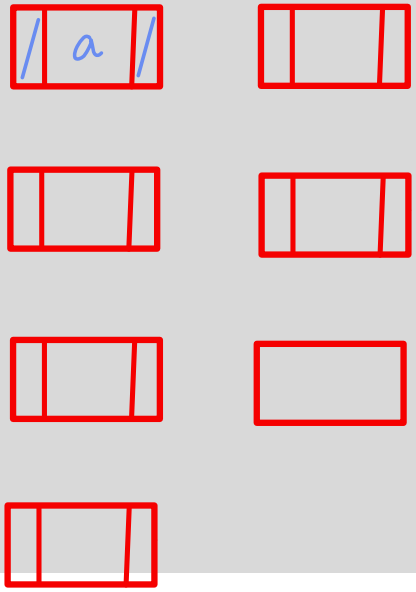
เปลี่ยน infix ⟶ Postfix

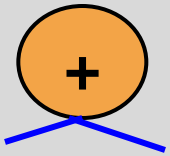
a b + c d e + *+

* เจอ อักขระ push ลง stack

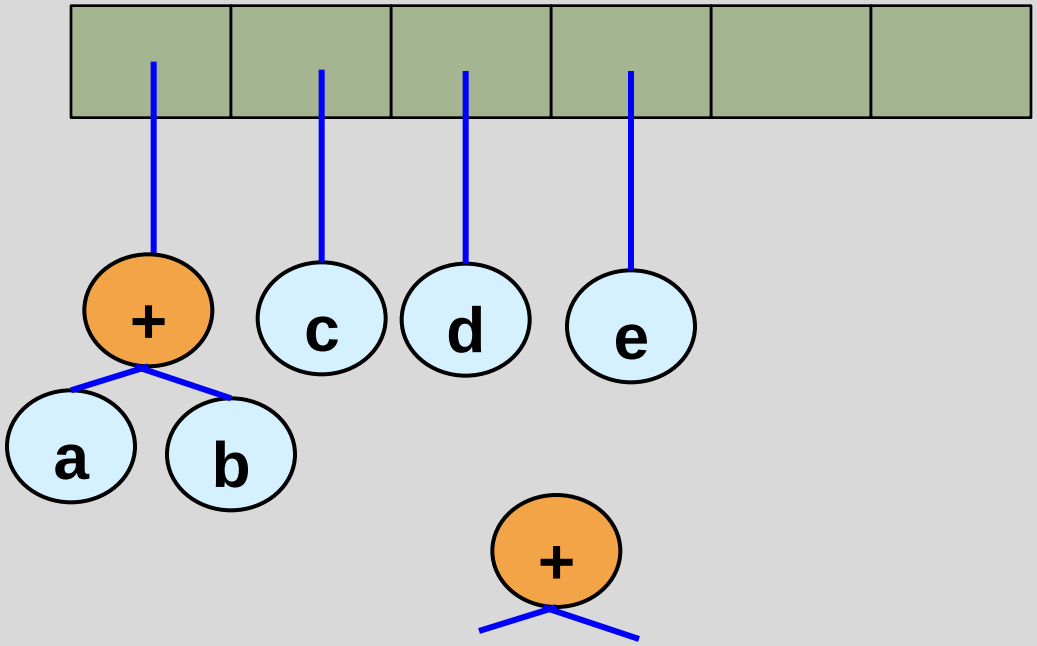
เจอ เครื่องหมาย pop



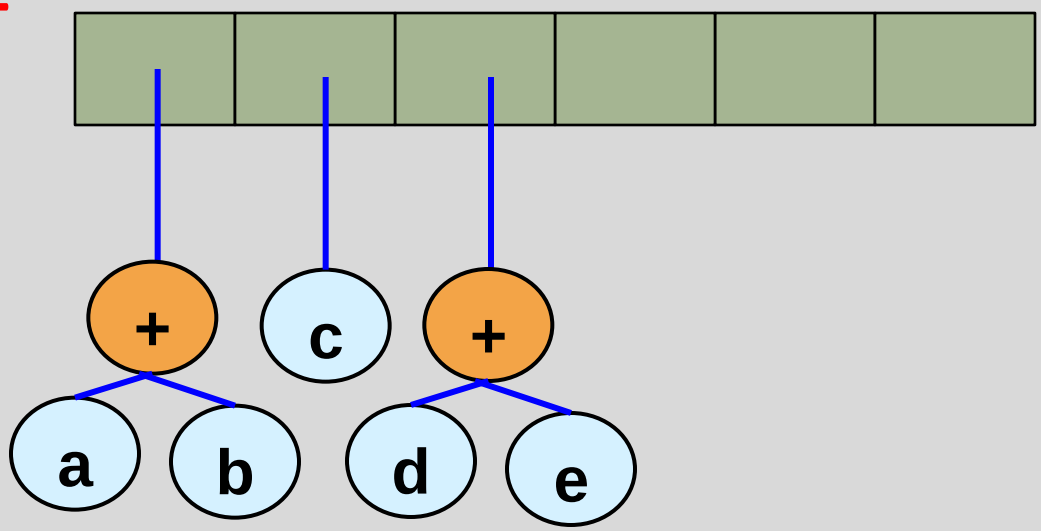




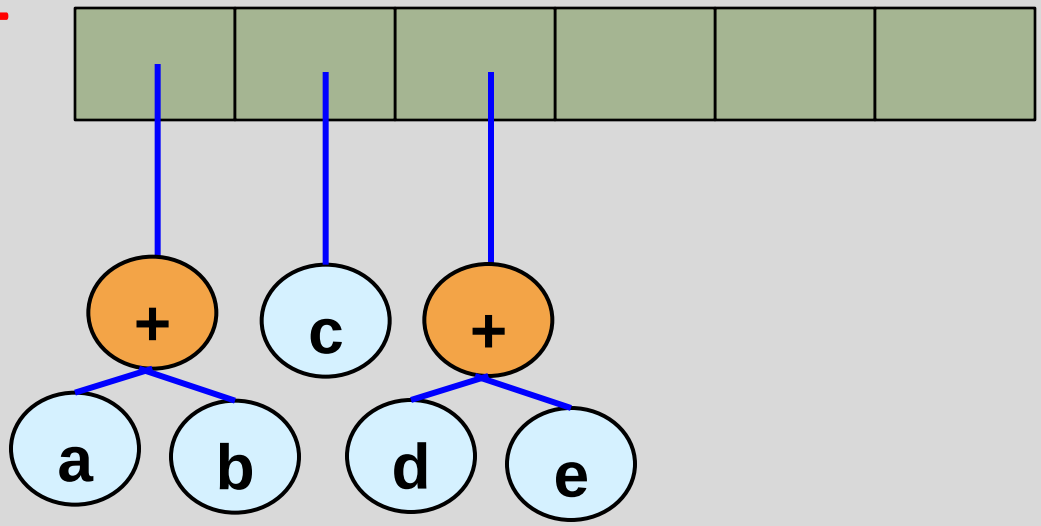
a b + c d e + *+



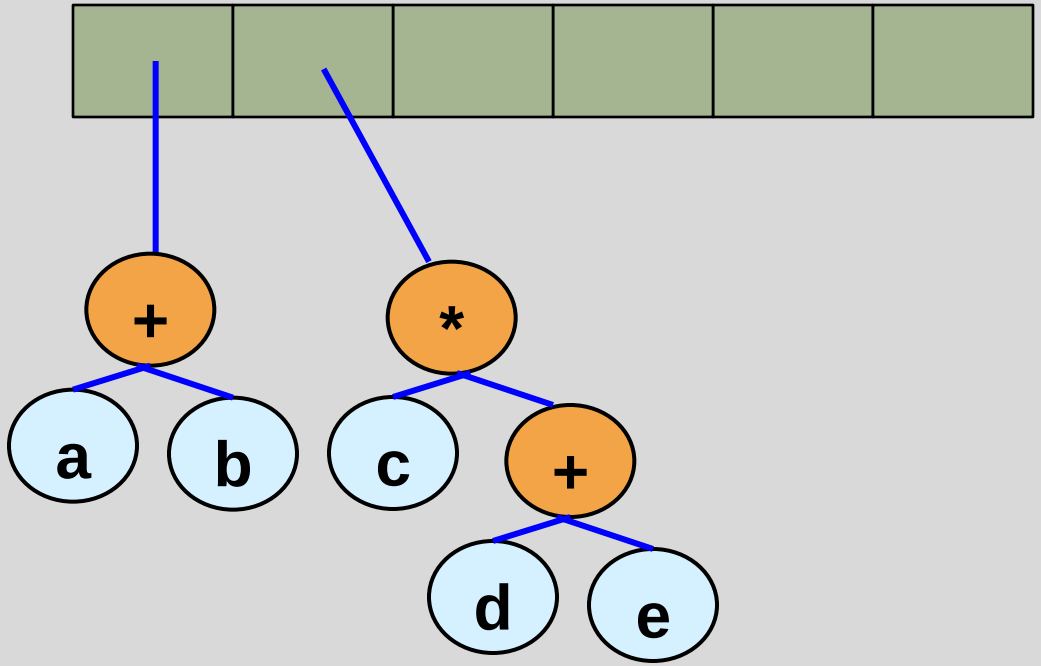
a b + c d e + *+



a b + c d e + *+



a b + c d e + *+



a b + c d e + *+