HANYANG UNIVERSITY
DIVISION OF COMPUTER SCIENCE

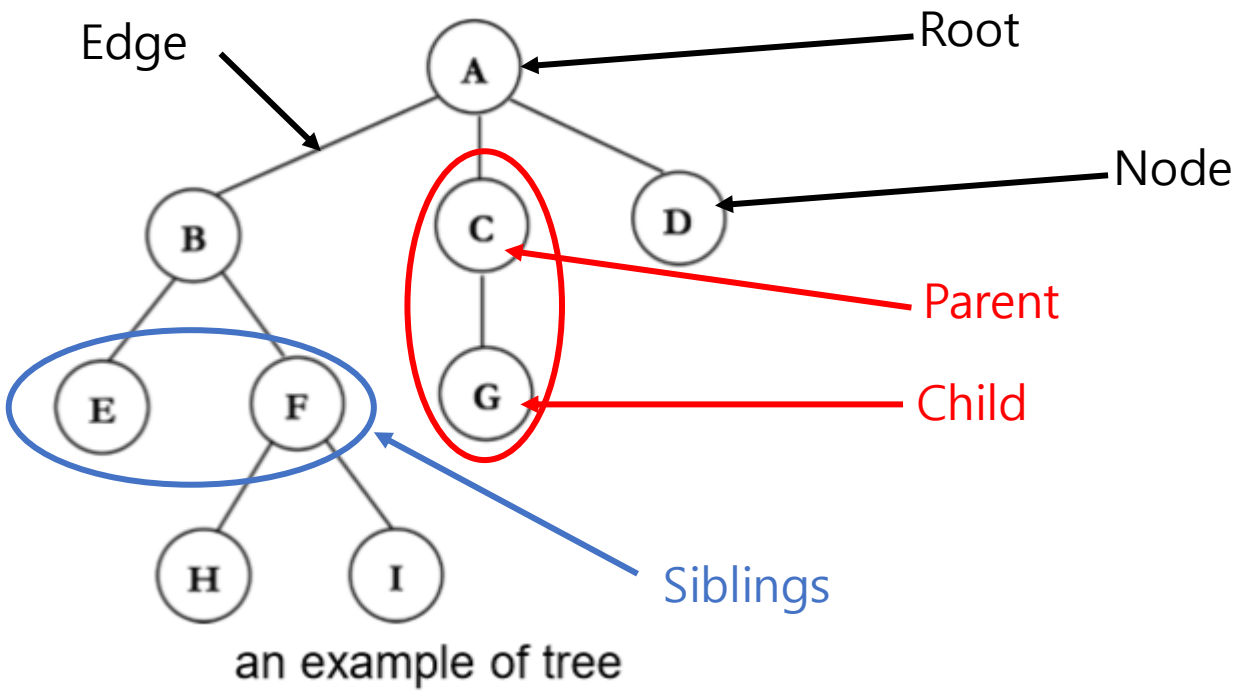
Data Structures

박영준 교수님

Lab4:Tree

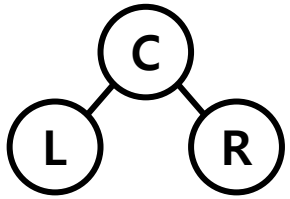
Tree

- A collection of nodes connected by edges without a cycle

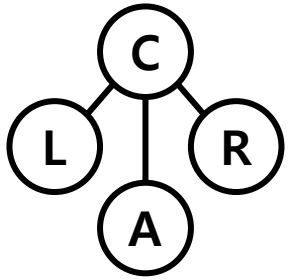


Binary Tree

- Tree data structure in which each node has two children, which are referred to as the left child and the right child.



Binary Tree



Binary Tree X

Binary Tree ADT

- TreeNode *MakeTree(void);
  - Create binary tree node
  - Return tree node
- void SaveData(TreeNode *tree, DATATYPE data);
  - Save data into node
- DATATYPE RetData(TreeNode *tree);
  - Return data from node

Binary Tree ADT

- void MakeSubTreeLeft(TreeNode *tree, TreeNode *subtree);
  - Link subtree node with left edge of tree node
- void MakeSubTreeRight(TreeNode *tree, TreeNode *subtree);
  - Link subtree node with right edge of tree node

Binary Tree ADT

- TreeNode *RetSubTreeLeft(TreeNode *tree);
  - Return left subtree node from tree node
- TreeNode *RetSubTreeRight(TreeNode *tree);
  - Return right subtree node from tree node

Binary Tree

Linked List

```c
#include <stdio.h>
#include <stdlib.h>

typedef int DATATYPE;

typedef struct TreeNode
{
    DATATYPE data;
    struct TreeNode *Left;
    struct TreeNode *Right;
} TreeNode;

TreeNode *MakeTree(void);
DATATYPE RetData(TreeNode *tree);
void SaveData(TreeNode *tree, DATATYPE data);

TreeNode *RetSubTreeLeft(TreeNode *tree);
TreeNode *RetSubTreeRight(TreeNode *tree);

void MakeSubTreeLeft(TreeNode *tree, TreeNode *subtree);
void MakeSubTreeRight(TreeNode *tree, TreeNode *subtree);
```

```c
TreeNode *MakeTree(void)
{
    TreeNode *tree = (TreeNode*)malloc(sizeof(TreeNode));
    tree->Left = NULL;
    tree->Right = NULL;

    return tree;
}//change to pass by argument

DATATYPE RetData(TreeNode *tree)
{
    return tree->data;
}

void SaveData(TreeNode *tree, DATATYPE data)
{
    tree->data = data;
}
```

Binary Tree

Linked List

```
81 TreeNode *RetSubTreeLeft(TreeNode *tree)
82 {
83     return tree->Left;
84 }
85
86 TreeNode *RetSubTreeRight(TreeNode *tree)
87 {
88     return tree->Right;
89 }
90
```

```
91 void MakeSubTreeLeft(TreeNode *tree, TreeNode *subtree)
92 {
93     if(tree->Left != NULL)
94     {
95         free(tree->Left);
96     }
97
98     tree->Left = subtree;
99 }
100
101 void MakeSubTreeRight(TreeNode *tree, TreeNode *subtree)
102 {
103     if(tree->Right != NULL)
104     {
105         free(tree->Right);
106     }
107
108     tree->Right = subtree;
109 }
110
```

Binary Tree

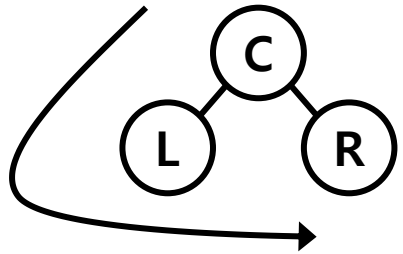
Linked List

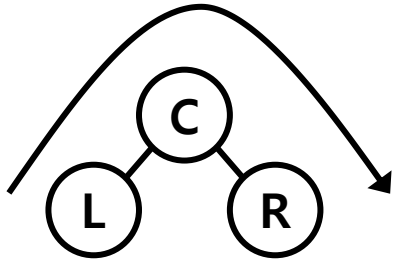
```
23 int main(int argc, char *argv[])
24 {
25     TreeNode *tree1 = MakeTree();
26     TreeNode *tree2 = MakeTree();
27     TreeNode *tree3 = MakeTree();
28     TreeNode *tree4 = MakeTree();
29     TreeNode *tree5 = MakeTree();
30
31     SaveData(tree1, 1);
32     SaveData(tree2, 2);
33     SaveData(tree3, 3);
34     SaveData(tree4, 4);
35     SaveData(tree5, 5);
36
37     MakeSubTreeLeft(tree1, tree2);
38     MakeSubTreeRight(tree1, tree3);
39     MakeSubTreeLeft(tree2, tree4);
40     MakeSubTreeRight(tree2, tree5);
41
```

```
37     MakeSubTreeLeft(tree1, tree2);
38     MakeSubTreeRight(tree1, tree3);
39     MakeSubTreeLeft(tree2, tree4);
40     MakeSubTreeRight(tree2, tree5);
41
42     //print tree
43     printf("%d ", RetData(tree1));
44
45     //print left subtree data of tree1
46     printf("%d ", RetData(RetSubTreeLeft(tree1)));
47
48     //print left subtree data of tree2
49     printf("%d ", RetData(RetSubTreeLeft(RetSubTreeLeft(tree1))));
50
51     //print right subtree data of tree2
52     printf("%d ", RetData(RetSubTreeRight(RetSubTreeLeft(tree1))));
53
54     //print right subtree data of tree1
55     printf("%d ", RetData(RetSubTreeRight(tree1)));
56
57     printf("\n");
58
59     return 0;
60 }
```

Tree Traversal

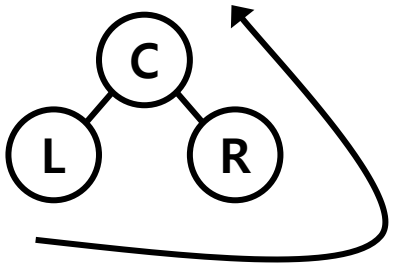
- Preorder traversal
  - Access current node
  - Traverse left subtree recursively
  - Traverse right subtree recursively
- Inorder traversal
  - Traverse left subtree recursively
  - Access current node
  - Traverse right subtree recursively





Tree Traversal

- Postorder traverse
  - Traverse left subtree recursively
  - Traverse right subtree recursively
  - Access current node



Binary Tree Traversal ADT

- void PreorderTraversal(TreeNode *tree);
  - Traverse tree by preorder
- void InorderTraversal(TreeNode *tree);
  - Traverse tree by inorder
- void PostorderTraversal(TreeNode *tree);
  - Traverse tree by postorder

Binary Tree Traversal

Linked List

```c
#include <stdio.h>
#include <stdlib.h>

typedef int DATATYPE;

typedef struct TreeNode
{
    DATATYPE data;
    struct TreeNode *Left;
    struct TreeNode *Right;
} TreeNode;

void PreorderTraversal(TreeNode *tree);
void InorderTraversal(TreeNode *tree);
void PostorderTraversal(TreeNode *tree);

TreeNode *MakeTree(void);
DATATYPE RetData(TreeNode *tree);
void SaveData(TreeNode *tree, DATATYPE data);

TreeNode *RetSubTreeLeft(TreeNode *tree);
TreeNode *RetSubTreeRight(TreeNode *tree);

void MakeSubTreeLeft(TreeNode *tree, TreeNode *subtree);
void MakeSubTreeRight(TreeNode *tree, TreeNode *subtree);
```

Binary Tree Traversal

Linked List

```
58 void PreorderTraversal(TreeNode *tree)
59 {
60     if(tree == NULL)
61     {
62         return ;
63     }
64
65     printf("%d ", tree->data);
66     PreorderTraversal(tree->Left);
67     PreorderTraversal(tree->Right);
68 }
69
70 void InorderTraversal(TreeNode *tree)
71 {
72     if(tree == NULL)
73     {
74         return ;
75     }
76
77     InorderTraversal(tree->Left);
78     printf("%d ", tree->data);
79     InorderTraversal(tree->Right);
80 }
81
```

```
82 void PostorderTraversal(TreeNode *tree)
83 {
84     if(tree == NULL)
85     {
86         return ;
87     }
88
89     PostorderTraversal(tree->Left);
90     PostorderTraversal(tree->Right);
91     printf("%d ", tree->data);
92 }
```

Binary Tree Traversal

Linked List

```
62 TreeNode *MakeTree(void)
63 {
64     TreeNode *tree = (TreeNode*)malloc(sizeof(TreeNode));
65     tree->Left = NULL;
66     tree->Right = NULL;
67
68     return tree;
69 }//change to pass by argument
70
71 DATATYPE RetData(TreeNode *tree)
72 {
73     return tree->data;
74 }
75
76 void SaveData(TreeNode *tree, DATATYPE data)
77 {
78     tree->data = data;
79 }
```

```
81 TreeNode *RetSubTreeLeft(TreeNode *tree)
82 {
83     return tree->Left;
84 }
85
86 TreeNode *RetSubTreeRight(TreeNode *tree)
87 {
88     return tree->Right;
89 }
90
```

Binary Tree Traversal

Linked List

```
 91 void MakeSubTreeLeft(TreeNode *tree, TreeNode *subtree)
 92 {
 93     if(tree->Left != NULL)
 94     {
 95         free(tree->Left);
 96     }
 97
 98     tree->Left = subtree;
 99 }
100
101 void MakeSubTreeRight(TreeNode *tree, TreeNode *subtree)
102 {
103     if(tree->Right != NULL)
104     {
105         free(tree->Right);
106     }
107
108     tree->Right = subtree;
109 }
110
```

Binary Tree Traversal

Linked List

```
27 int main(int argc, char *argv[])
28 {
29     TreeNode *tree1 = MakeTree();
30     TreeNode *tree2 = MakeTree();
31     TreeNode *tree3 = MakeTree();
32     TreeNode *tree4 = MakeTree();
33     TreeNode *tree5 = MakeTree();
34
35     SaveData(tree1, 1);
36     SaveData(tree2, 2);
37     SaveData(tree3, 3);
38     SaveData(tree4, 4);
39     SaveData(tree5, 5);
40
41     MakeSubTreeLeft(tree1, tree2);
42     MakeSubTreeRight(tree1, tree3);
43     MakeSubTreeLeft(tree2, tree4);
44     MakeSubTreeRight(tree2, tree5);
45
46     //print tree
47     printf("preorder : ");
48     PreorderTraversal(tree1);
49     printf("\n");
50
51     printf("inorder : ");
52     InorderTraversal(tree1);
53     printf("\n");
54
55     printf("postorder : ");
56     PostorderTraversal(tree1);
57     printf("\n");
58
59
60     return 0;
61 }
62
```

Lab4: Calculator(tree)

- Submit on GitLab
- Postfix calculator using tree & expression printer
- Create Lab4 directory on your own GitLab project
- Submit file : source code(c only, run on linux)
- Filename : StudentID_lab4.c
- Input file : no

Lab4: Calculator(tree)

- Postfix calculator & expression printer using tree & linked list based stack
- ex) Input postfix equation : 12+7*

  Answer : 21
  Prefix expression : *+127
  Infix expression : 1+2*7
  Postfix expression : 12+7*

Lab4: Calculator(tree) ADT

- TreeNode *MakeExpTree(char exp[]);
  - Pass the postfix equation as an argument
  - Make exp a tree using linked list based stack
  - Return root node of the tree
- int EvalExpTree(TreeNode *tree);
  - Pass the tree returned by MakeExpTree()
  - Calculate postfix equation stored as tree
  - Return answer
  - Using recursive calls

Lab4: Calculator(tree)

- Output example

```
prefix : * + 1 2 7
infix : 1 + 2 * 7
postfix : 1 2 + 7 *
result : 21
```

Lab4: Calculator(tree)

```
80 TreeNode *MakeExpTree(char exp[])
81 {
82     ListStack stack;
83     TreeNode *tree;
84
85     int ExpLen = strlen(exp);
86
87     InitStack(&stack);
88
89     for(int i = 0; i < ExpLen; i++)
90     {
91         tree = MakeTree();
92
93         if(isdigit(exp[i]))
94         {
95             //Note, make tree node with integer data
96
97         }
98         else
99         {
00             //Note, make tree, root - operator, childs - integers inserted in stack
01
02         }
03
04         Push(&stack, tree);
05     }
06
07     return Pop(&stack);
08 }
09
```

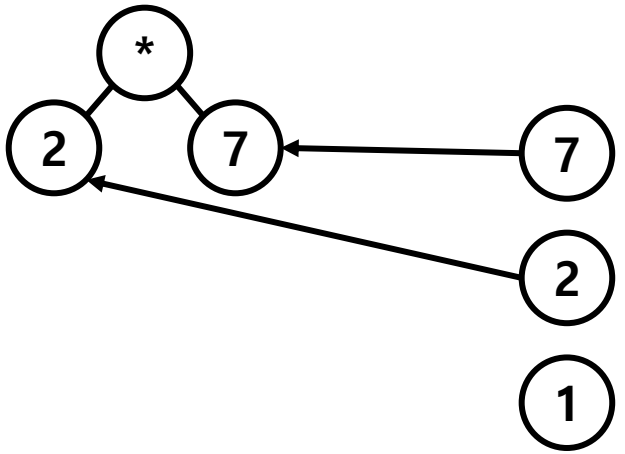
exp[] = 127*+

Stack

Lab4: Calculator(tree)

```
80 TreeNode *MakeExpTree(char exp[])
81 {
82     ListStack stack;
83     TreeNode *tree;
84
85     int ExpLen = strlen(exp);
86
87     InitStack(&stack);
88
89     for(int i = 0; i < ExpLen; i++)
90     {
91         tree = MakeTree();
92
93         if(isdigit(exp[i]))
94         {
95             //Note, make tree node with integer data
96
97         }
98         else
99         {
00             //Note, make tree, root - operator, childs - integers inserted in stack
01
02         }
03
04         Push(&stack, tree);
05     }
06
07     return Pop(&stack);
08 }
09
```

exp[] = 127*+

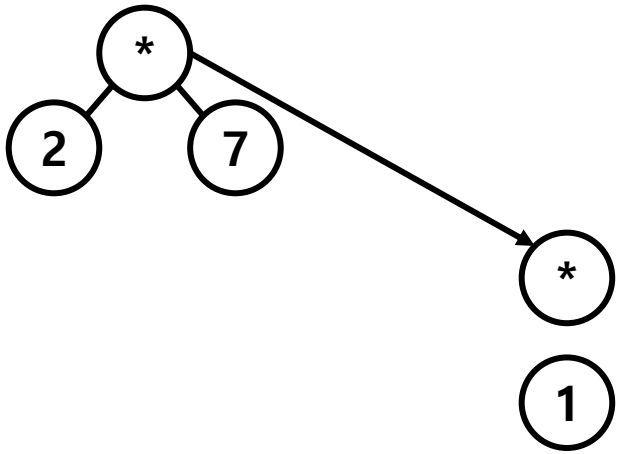


Stack

Lab4: Calculator(tree)

```
80 TreeNode *MakeExpTree(char exp[])
81 {
82     ListStack stack;
83     TreeNode *tree;
84
85     int ExpLen = strlen(exp);
86
87     InitStack(&stack);
88
89     for(int i = 0; i < ExpLen; i++)
90     {
91         tree = MakeTree();
92
93         if(isdigit(exp[i]))
94         {
95             //Note, make tree node with integer data
96
97         }
98         else
99         {
00             //Note, make tree, root - operator, childs - integers inserted in stack
01
02         }
03
04         Push(&stack, tree);
05     }
06
07     return Pop(&stack);
08 }
09
```

exp[] = 127*+

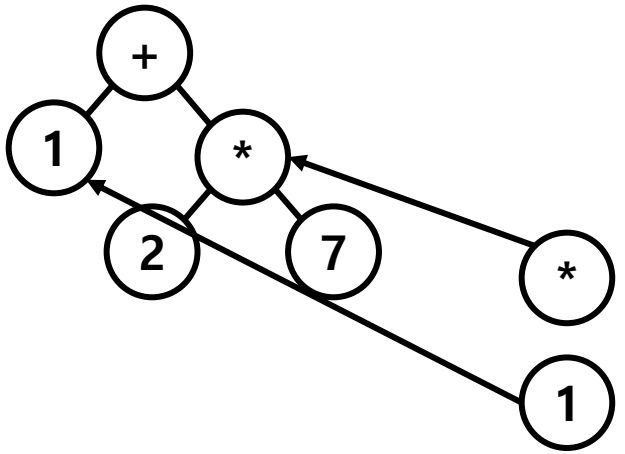


Stack

Lab4: Calculator(tree)

```
80 TreeNode *MakeExpTree(char exp[])
81 {
82     ListStack stack;
83     TreeNode *tree;
84
85     int ExpLen = strlen(exp);
86
87     InitStack(&stack);
88
89     for(int i = 0; i < ExpLen; i++)
90     {
91         tree = MakeTree();
92
93         if(isdigit(exp[i]))
94         {
95             //Note, make tree node with integer data
96
97         }
98         else
99         {
00             //Note, make tree, root - operator, childs - integers inserted in stack
01
02         }
03
04         Push(&stack, tree);
05     }
06
07     return Pop(&stack);
08 }
09
```

exp[] = 127*+

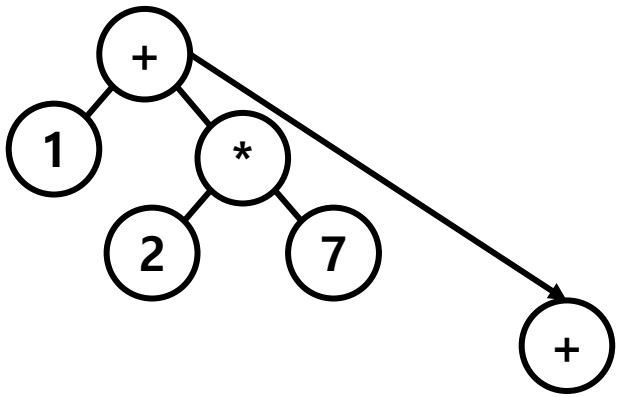


Stack

Lab4: Calculator(tree)

```
80 TreeNode *MakeExpTree(char exp[])
81 {
82     ListStack stack;
83     TreeNode *tree;
84
85     int ExpLen = strlen(exp);
86
87     InitStack(&stack);
88
89     for(int i = 0; i < ExpLen; i++)
90     {
91         tree = MakeTree();
92
93         if(isdigit(exp[i]))
94         {
95             //Note, make tree node with integer data
96
97         }
98         else
99         {
00             //Note, make tree, root - operator, childs - integers inserted in stack
01
02         }
03
04         Push(&stack, tree);
05     }
06
07     return Pop(&stack);
08 }
09
```

exp[] = 127*+



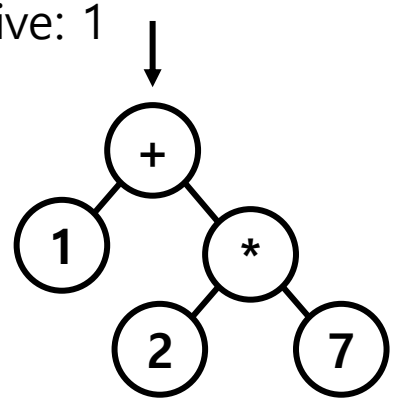
Stack

Lab4: Calculator(tree)

```
110 int EvalExpTree(TreeNode *tree)
111 {
112     //Note, if tree reached one end
113     if(    )
114     {
115         return RetData(tree);
116     }
117
118     int op1 = EvalExpTree(RetSubTreeLeft(tree));
119     int op2 = EvalExpTree(RetSubTreeRight(tree));
120
121     switch(RetData(tree))
122     {
123         case '+':
124             return op1 + op2;
125         case '-':
126             return op1 - op2;
127         case '*':
128             return op1 * op2;
129         case '/':
130             return op1 / op2;
131     }
132
133     return 0;
134 }
135
```

exp[] = 127*+

Recursive: 1



Lab4: Calculator(tree)

```
110 int EvalExpTree(TreeNode *tree)
111 {
112     //Note, if tree reached one end
113     if(    )
114     {
115         return RetData(tree);
116     }
117
118     int op1 = EvalExpTree(RetSubTreeLeft(tree));
119     int op2 = EvalExpTree(RetSubTreeRight(tree));
120
121     switch(RetData(tree))
122     {
123         case '+':
124             return op1 + op2;
125         case '-':
126             return op1 - op2;
127         case '*':
128             return op1 * op2;
129         case '/':
130             return op1 / op2;
131     }
132
133     return 0;
134 }
135
```

exp[] = 127*+

Recursive: 2