
Homework # 6

데이터구조론 (CSE2003-02)

HW#6 과제 안내

- 일정
 - 게시 : 5/4(화) 17:00
 - 제출 마감 : 5/11(화) 15:00 (delay 없음)
 - 채점 결과 확인 : 5/17(월)
 - 이의신청 마감 : 5/24(월) (이의신청 이메일 : greenlife124@yonsei.ac.kr,
[데이타구조론HW#6 이의신청])

HW#6 과제 안내

- 설명
 - 모든 코드의 핵심 부분에는 comment를 달아 설명 할 것 (not option!!)
 - Compiler는 visual studio 2019 이상을 사용하여, HW#6_학번_이름 하나의 파일로 압축하여 제출 할 것
- HW#6_학번_이름
 - HW#6_1 > **BinaryTree.c**, BinaryTree.h, BinaryTreeMain.c
 - HW#6_2 > **BinaryTree.c**, ExpressionMain.c, **ExpressionTree.c**, **LinkedStack.c**, BinaryTree.h, ExpressionTree.h, LinkedStack.h

HW#6_1 이진 트리 구현

- 아래와 같이 실행되도록 **BinaryTree.c** 작성
 - `makeLSubtree()`, `makeRSubtree()`, `getLSubtree()`, `getRSubtree()`, `setData()`, `getData()`, `InorderTraverse()`, `PreorderTraverse()`, `PostorderTraverse()` 작성
- `BinaryTree.h`, `BinaryTreeMain.c` 제공
- 주의 사항
 - HW#6_1 폴더 안에, 주어진 3개의 코드 모두 존재 해야 함

```
=== (1) 출력 ===
root: 1
root의 왼쪽 자식: 2
root의 오른쪽 자식: 3
root의 왼쪽 자식의 왼쪽 자식: 4
root의 왼쪽 자식의 오른쪽 자식: 5
root의 오른쪽 자식의 왼쪽 자식: 6
root의 오른쪽 자식의 오른쪽 자식: -1

=== (2) 중위 순회 ===
4 2 5 1 6 3

=== (3) 전위 순회 ===
1 2 4 5 3 6

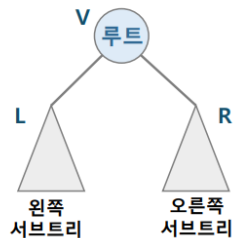
=== (4) 후위 순회 ===
4 5 2 6 3 1

=== (5) tree 소멸 ===
tree 값 삭제: 4
tree 값 삭제: 5
tree 값 삭제: 2
tree 값 삭제: 6
tree 값 삭제: 3
tree 값 삭제: 1
```

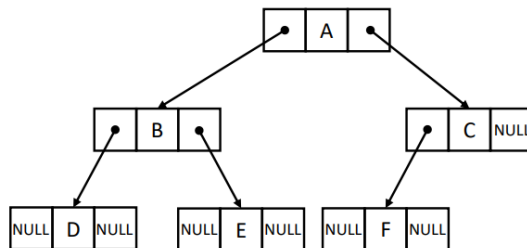
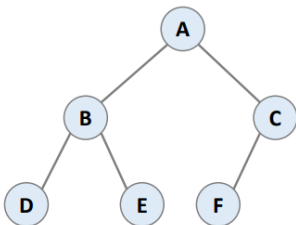
HW#6_1 참고 자료

순회방법 (L: 왼쪽 이동, V: 노드 방문, R: 오른쪽 이동)

- **LVR** : 중위 순회 (inorder traversal)
- **VLR** : 전위 순회 (preorder traversal)
- **LRV** : 후위 순회 (postorder traversal)



완전 이진 트리



Algorithm void inorderTraverse(BinTree* bt)

```
inorderTraverse(bt)
  if (bt≠NULL) then           // 노드가 NULL이면 더 이상 재귀호출 하지 않음
    inorderTraverse(bt.left)   // bt의 왼쪽 서브트리 방문
    visit bt.data              // bt의 루트 data 처리 (예: 출력)
    inorderTraverse(bt.right)  // bt의 오른쪽 서브트리 방문
  endif
end inorderTraverse()
```

Algorithm void preorderTraverse(BinTree* bt)

```
preorderTraverse(bt)
  if (bt≠NULL) then           // 노드가 NULL이면 더 이상 재귀호출 하지 않음
    visit bt.data              // bt의 루트 data 처리 (예: 출력)
    preorderTraverse(bt.left)  // bt의 왼쪽 서브트리 방문
    preorderTraverse(bt.right) // bt의 오른쪽 서브트리 방문
  endif
end preorderTraverse()
```

Algorithm void postorderTraverse(BinTree* bt)

```
postorderTraverse(bt)
  if (bt≠NULL) then           // 노드가 NULL이면 더 이상 재귀호출 하지 않음
    postorderTraverse(bt.left) // bt의 왼쪽 서브트리 방문
    postorderTraverse(bt.right) // bt의 오른쪽 서브트리 방문
    visit bt.data              // bt의 루트 data 처리 (예: 출력)
  endif
end postorderTraverse()
```

HW#6_2 수식 트리 구현

- 아래와 같이 실행되도록 **BinaryTree.c** 작성
 - `makeLSubtree()`, `makeRSubtree()`, `getLSubtree()`, `getRSubtree()`, `setData()`, `getData()`, `InorderTraverse()`, `PreorderTraverse()`, `PostorderTraverse()` 작성
- 아래와 같이 실행되도록 **ExpressionTree.c** 작성
 - `createExpTree()`, `evalExpTree()`, `showPrefixExp()`, `showPostfixExp()` 작성
- 아래와 같이 실행되도록 **LinkedStack.c** 작성
 - `createStack()`, `isFull()`, `isEmpty()`, `push()`, `pop()`, `peek()` 작성
- `ExpressionMain.c`, `BinaryTree.h`, `ExpressionTree.h`, `LinkedStack.h` 제공
- 주의 사항
 - HW#6_2 폴더 안에, 주어진 7개의 코드 모두 존재 해야 함

```
전위 표기법의 수식: * + 1 2 7
중위 표기법의 수식: ( ( 1 + 2 ) * 7 )
후위 표기법의 수식: 1 2 + 7 *
연산 결과: 21
```

HW#6_2 참고 자료

```
Stack* create() {  
    Stack* S = (Stack*)malloc(sizeof(Stack));  
    S->top = NULL;  
    return S;  
}
```

```
int isEmpty(Stack* S){  
    return S->top == NULL;  
}
```

```
int isFull(Stack* S){  
    return 0;  
}
```

```
element peek(Stack* S {  
    element e;  
    if( isEmpty(S) ){  
        printf("[ERROR] Stack is EMPTY!!\n");  
        return ERROR;  
    }  
    else  
        return S->top->data;  
}
```

```
void push(Stack* S, element x) {  
    stackNode* newNode = (stackNode*)malloc(sizeof(stackNode));  
    newNode->data = x;  
    newNode->link = S->top;  
    S->top = newNode;  
}
```


```
element pop(Stack* S) {  
    stackNode* temp; element e;  
    if( isEmpty(S) ){  
        printf("[ERROR] Stack is EMPTY!!\n");  
        return ERROR;  
    } else {  
        temp = S->top;  
        e = temp->data;  
        S->top = temp->link;  
        free(temp);  
        return e;  
    }  
}
```

HW#6_2 참고 자료

```
BinTree* createExpTree(char exp[]){
    Stack* S = createStack();
    BinTree* bNode;
    int expLen = strlen(exp);
    int i;

    for(i=0; i<expLen; i++) {
        bNode = createBT();
        if(isdigit(exp[i]))
            setData(bNode, exp[i]-'0');
        else {
            makeRSubtree(bNode, pop(S));
            makeLSubtree(bNode, pop(S));
            setData(bNode, exp[i]);
        }
        push(S, bNode);
    }

    return pop(S);
}
```



```
int evalExpTree(BinTree* bt){
    int op1, op2;

    op1 = getData(getLSubtree(bt));
    op2 = getData(getRSubtree(bt));

    switch(getData(bt)) {
        case '+':
            return op1 + op2;
        case '-':
            return op1 - op2;
        case '*':
            return op1 * op2;
        case '/':
            return op1 / op2;
    }

    return 0;
}
```