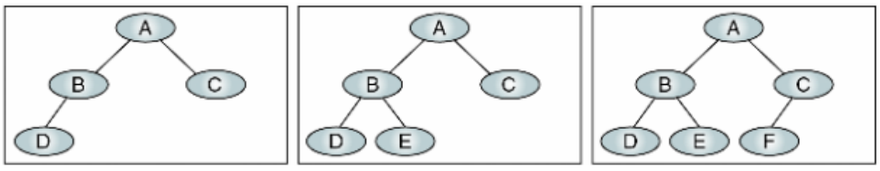
HW 7 관련 (TA 작성)

[관련 개념]

Binary Tree: Tree + Only 2 subtree

Complete Binary Tree:

- **Complete binary tree: a tree that has minimum height for its nodes and all nodes in the last level are found on the left**



Depth First search를 구현하라!

- Preorder, Inorder, Postorder (make it recursively!)

Preorder Traversal

- Algorithm

```
void Preorder(TreeNode *root)
{
   if(root == NULL)
     return;

   printf("%s", root->data);
   Preorder(root->left);
   Preorder(root->right);
}
```

Inorder Traversal

- Algorithm

```
void Inorder(TreeNode *root)
{
   if(root == NULL)
      return;
   Inorder(root->left);
   printf("%s", root->data);
   Inorder(root->right);
}
```

Postorder Traversal

- Algorithm

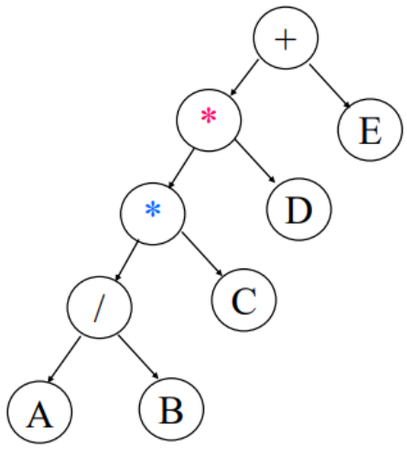
```
void Postorder(TreeNode *root)
{
   if(root == NULL)
     return;

   Postorder(root->left);
   Postorder(root->right);
   printf("%s", root->data);
}
```

Breadth-First Traversal

Breadth-First Traversal

- **Algorithm in C**

```
void LevelOrder(TreeNode *root)
{
  Queue *queue = NULL;
  if(root == NULL)
    return;
  queue = CreateQueue(…);
  while(root){
    Process(root->data);
    if(root->left)
      Enqueue(queue, root->left);
    if(root->right)
      Enqueue(queue, root->right);
    if(!IsEmptyQueue(queue))
      root = Dequeue(queue);
    else
      root = NULL;
  }
  DestroyQueue(queue);
}
```

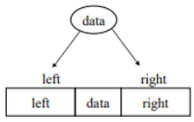


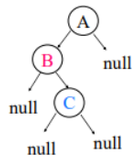
complete Binary-Tree

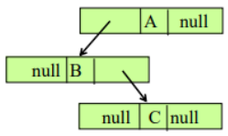
Representation of Binary Tree

- **Each node has data and pointers to left/right child**

```
typedef struct tTreeNode {
   Element data;
   struct tTreeNode *left, *right;
} TreeNode ;
```







CreateTree

- **Algorithm**

```
TreeNode* CreateTree(TreeNode *left, Element item, TreeNode *right)
{
   TreeNode *pNewNode = (TreeNode*)malloc(sizeof(TreeNode));
   if(pNewNode == NULL)
     return NULL;

   strcpy(pNewNode->data, item);      // assuming Element is char[]

   pNewNode->left = left;
   pNewNode->right = right;

   return pNewNode;
}
```

[HW7 채점 기준]

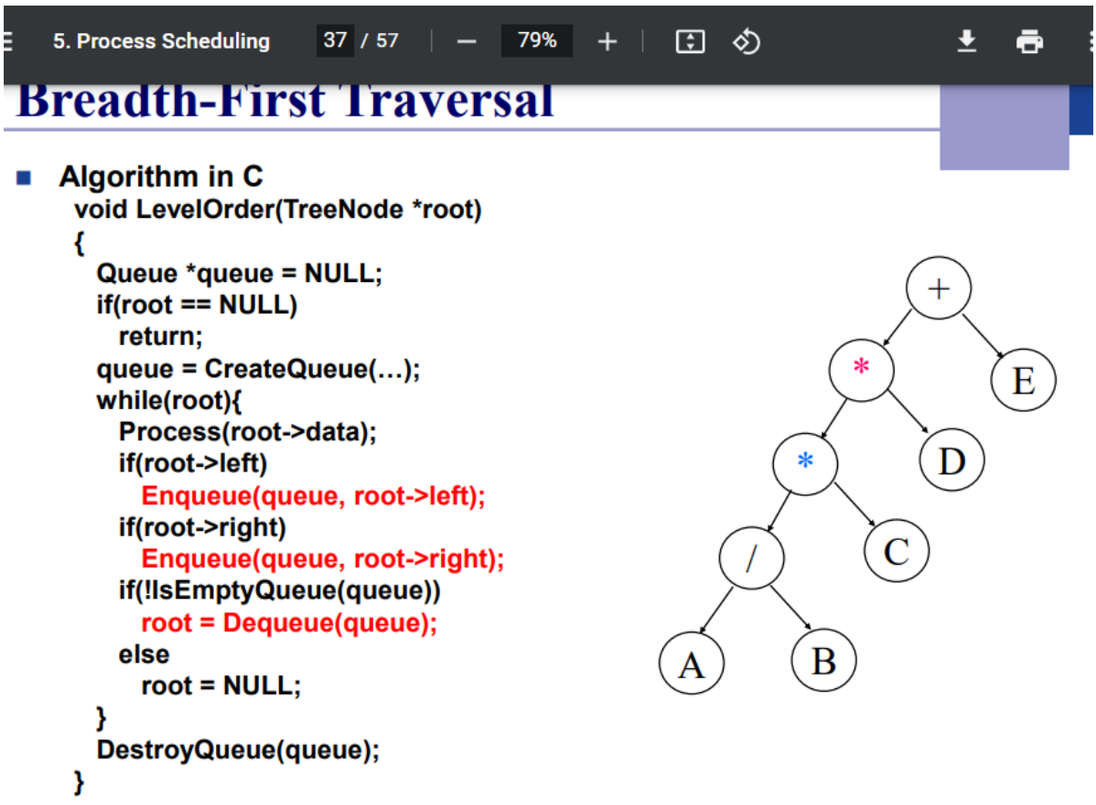
만점 (50)

- **완전 이진 트리 (Complete Binary tree) 구현 (Binary_tree.h) (20)**
  - Binary Tree 구현 (10)
  - Complete Binary Tree 삽입 (10)
  - ※ 채점은 Binary_Tree.h를 기준으로 하며, 없는 경우 제출한 전체 코드를 포함했습니다. ADT 함수 DestroyTree는 감점 기준에 포함하지는 않았습니다.

- **Orders in Depth-First Traversal (15)**
  - Preorder traversal (⬛ Root → left subtree → right subtree) (5)
  - Inorder traversal (⬛ Left subtree → root → right subtree) (5)
  - Postorder traversal (⬛ Left subtree → right subtree → root) (5)
  - ※ 채점은 “Input String: abcdefg”를 기준으로 하며 하드코딩이 의심되면 여러 가지 입력을 통해 더 검증합니다.

```
$ ./main
Input a string : abcdefg

Pre-order : abdecfg

In-order : dbeafcg

Post-order : debfgca

Breadth First : abcdefg
```

- **Breadth-First Traversal (15)**
  - LevelOrder 구현 (10)

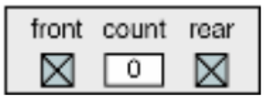


  - Queue 구조 구현 (5)

Linked List Implementation of CreateQueue

- **CreateQueue**

```
Queue* CreateQueue(int size)
// size is not used
{
    Queue *pNewQueue = (Queue*)malloc(sizeof(Queue));
    if(pNewQueue == NULL)
      return NULL;

    pNewQueue->count = 0;
    pNewQueue->front = pNewQueue->rear = NULL;

    return pNewQueue;
}
```



Linked List Implementation of Enqueue

- **Enqueue**

```
void Enqueue(Queue *pQueue, Element item)
{
    QueueNode *pNewNode = (QueueNode*) malloc(sizeof(QueueNode));
    if(pNewNode == NULL)
      return;
    pNewNode->data = item;
    pNewNode->next = NULL;

    if(pQueue->count <= 0){
      pQueue->front = pQueue->rear = pNewNode;
    } else {
      pQueue->rear->next = pNewNode;
      pQueue->rear = pNewNode;
    }

    pQueue->count++;
}
```

Linked List Implementation of Dequeue

- **Dequeue**

```
Element Dequeue(Queue *pQueue)
{
    QueueNode *pFront = NULL;
    Element item = 0;

    if(pQueue->count <= 0)
      return 0;                    // queue empty

    pFront = pQueue->front;
    item = pFront->data;

    if(pQueue->count == 1){
      pQueue->front = pQueue->rear = NULL;
    } else {
      pQueue->front = pFront->next;
    }

    free(pFront);
    pQueue->count--;

    return item;
}
```