본 Lab 은 저의 강의 경험과 학생들의 의견 및 Stanford CS106 과 Harvard CS50 같은 강의에서 수집된 자료를 토대로 작성되었습니다. 본 Lab 에 문제가 있거나, 질문 혹은 의견이 있다면, 언제든지 알려 주시면 감사하겠습니다. 강의 개선에 많은 도움이 되겠습니다. idebtor@gmail.com

Lab on BT

목차

소개

본 Lab 은 서로 연관된 3 개의 Lab 으로 구성되어 있습니다. 사용자가 이진 탐색 트리를 상호적으로 테스트할 수 있도록 tree.cpp 의 이진 트리(BT), 이진 탐색 트리(BST), 그리고 AVL 트리를 다루는 함수를 완성하세요. 다음 파일들이 제공됩니다.

- **driver.cpp** : tests BT/BST/AVL 트리 구현을 상호적으로 테스트합니다. 수정 금지.
- **tree.cpp** : BST/AVL 트리 구현을 위한 뼈대 코드.
- treenode.h : 기본 트리 구조와 key 자료형 정의. 수정 금지
- tree.h : BT, BST, AVL 트리에 대한 ADTs 정의. 수정 금지.
- treeprint.cpp : 콘솔에 트리 그림 출력
- treex.exe : 참고용 실행 답안

자신이 작성한 프로그램이 제공된 treex.exe 와 같이 작동해야 합니다. 자신의 tree.cpp 가 tree.h 와 treeDriver.cpp 와 호환되어야 합니다. 따라서, tree.h 와 tree.cpp 파일의 함수 시그니처와 반환 유형은 수정하지 않아야 합니다.

treeDriver.cpp 의 **build_tree_by_args()**는 명령 인수를 가져와서 위에 나온 것과 같이 **BT, BST** 또는 **AVL** 트리를 빌드합니다. 트리에 대한 인수가 제공되지 않으면 기본적으로 BT 로 시작합니다.

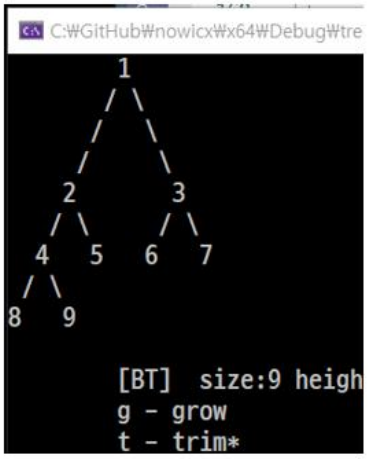
```
PS C:\GitHub\nowicx\psets\pset10-12tree> ./treex -b 1 2 3 4
    1
   / \
  2   3
 /
4

        Menu [BT]  size:4 height:2 min:1 max:4
        g - grow                a - grow a leaf    [BT]
        t - trim*               d - trim a leaf    [BT]
        G - grow N              A - grow by Level  [BT]
        T - trim N              f - find node      [BT]
        o - BST or AVL?         p - find path&back [BT]
        r - rebalance tree**    l - traverse       [BT]
        L - LCA*                B - LCA*           [BT]
        m - menu [BST]/[AVL]**  C - convert BT to BST*
        c - clear               s - show mode:[tree]
        Command(q to quit):
```

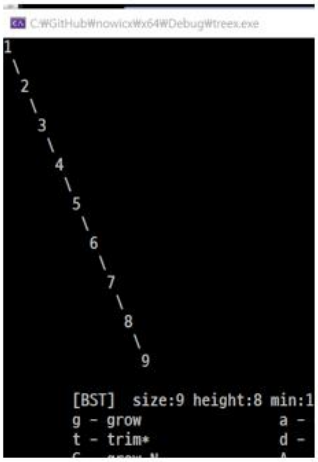
다음과 같은 3 가지 옵션을 사용하면 트리 프로그램 실행 시 자동으로 3 개의 다른 트리를 만들 수 있습니다.

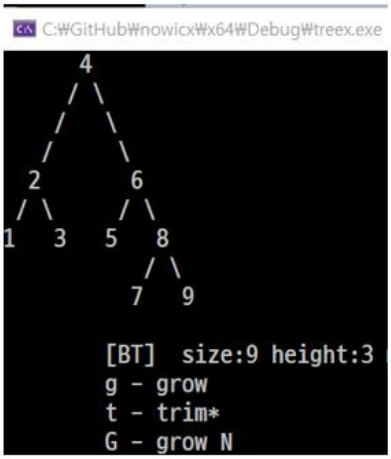
./treex -b 1 2 3 4 5 6 7 8 9

./treex -s 1 2 3 4 5 6 7 8 9

./treex -a 1 2 3 4 5 6 7 8 9







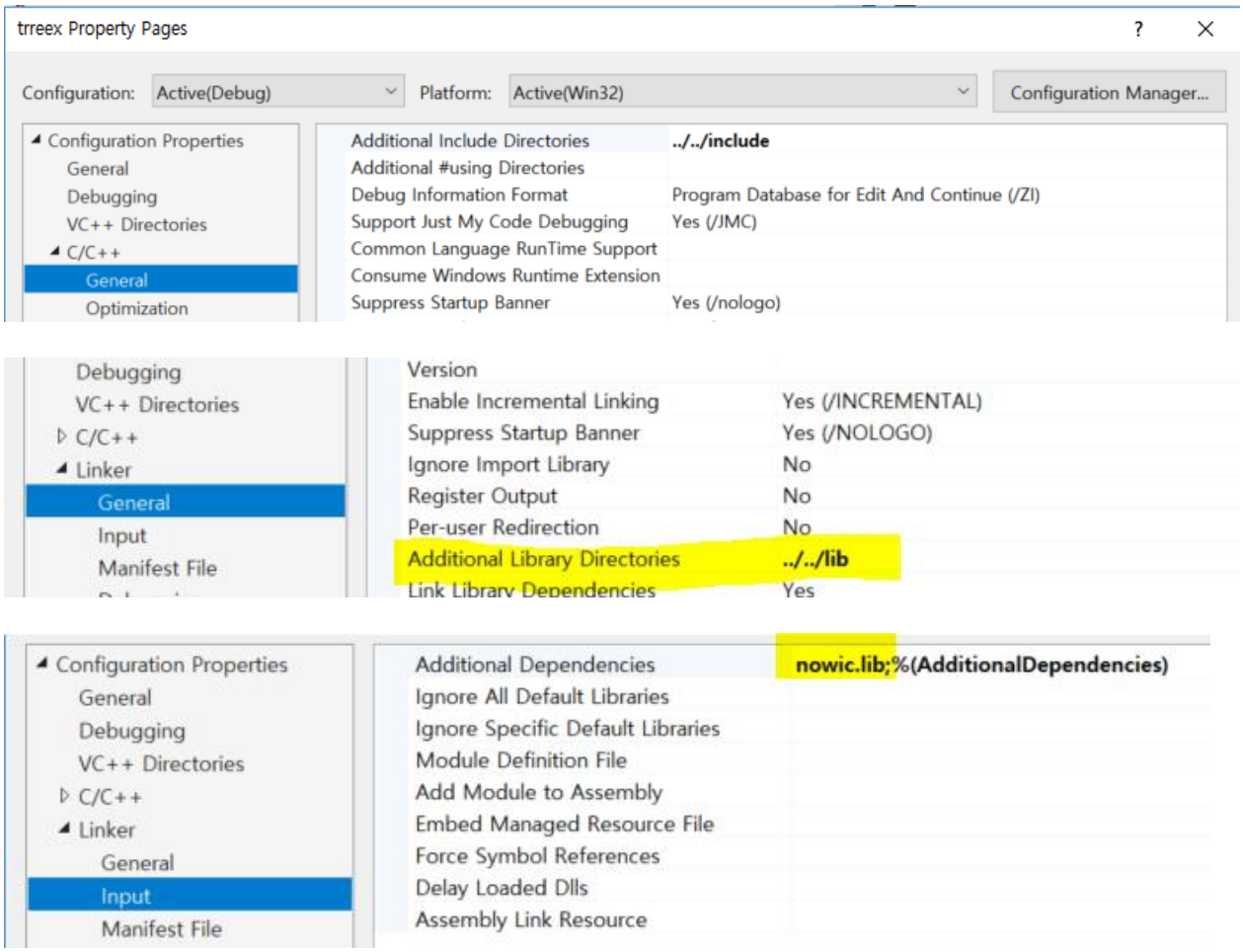
JumpStart

Jump-start 의 경우, 먼저 tree 라는 이름의 프로젝트를 생성합니다. 평소와 같이 다음 작업을 수행합니다.

- Add ~/include at
  - Project Property → C/C++ → General → Additional Include Directories
- Add ~/lib at
  - Project Property → Linker → General → Additional Library Directories
- Add nowic.lib at

  - Project Property → Linker → Input → Additional Dependencies
- Add /D "DEBUG" at
  - Project Property → C/C++ → Command Line

예시:

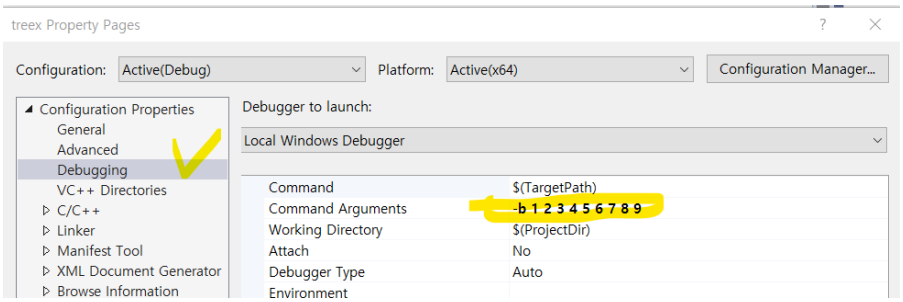


Add ~.h files under 프로젝트 'Header Files' 아래에 ~.h 파일을 추가하고 프로젝트 'Source Files' 아래에 ~.cpp 파일을 추가하면 프로젝트를 빌드할 수 있습니다.

Step 0: 디버깅을 위한 트리를 쉽게 만드는 방법

초반에는 디버깅을 목적으로 동일한 트리를 매번 생성하는 경우가 종종 있습니다. 트리의 초기 key 값을 다음 경로를 통해 지정할 수 있습니다.

Project Properties → Debugging → Command Argument



Step 1.1: BT 기본 작업

tree.cpp 의 일부 함수들은 이미 구현되어 있습니다. 아래 명시된 함수들을 구현하세요. 필요한 경우 (특히 재귀의 경우) 추가적인 도우미 함수를 작성하여 사용하세요. 이상적으로, 이 Lab 에서 모든 코드는 tree.cpp 에 작성합니다.

[BT]가 있는 메뉴 항목은 대개 3 가지 유형의 트리에 모두 적용됩니다 (다소 느리게 작동할 수 있습니다). 예를 들어, find(), findPath(), findPathBack(), maximum(), 그리고 minimum() 등등이 있습니다. 다음 함수들을 테스트하여 올바르게 작동하는지 확인하고, 개발 환경에 익숙해지세요. 한국어로 작성된 재귀 트리에 대한 참고 문헌을 확인해 보세요.

- `clear()`
- `size()`
- `height()`
- `minimumBT(), maximumBT();`
- `containsBT(), findBT()`
- `inorder(), preorder(), postorder()`

Step 1.2: levelorder() - iteration version

이 순회(traversal)는 하위 레벨로 내려가기 전에 해당 레벨에 존재하는 모든 노드를 방문합니다. 이 탐색은 탐색 트리가 다음 깊이로 이동하기 전에 각 깊이에서 최대한으로 넓어지므로 너비 우선 탐색(BFS)이라고 부릅니다. 이 작업에는 지정된 깊이에서 노드의 최대 개수에 비례하는 공간이 필요합니다. 이 크기는 총 노드의 개수 / 2 만큼 커질 수 있습니다.

알고리즘 (Iteration):

- 빈 queue 를 생성하고 루트 노드를 push 합니다.
- Queue 가 비워질 때까지 다음을 반복합니다.
  - Queue 에서 노드를 pop 하고 출력/저장하세요.
  - Pop 한 노드의 왼쪽 자식 노드가 null 이 아니라면 push 하세요.
  - Pop 한 노드의 오른쪽 자식 노드가 null 이 아니라면 push 하세요.

```
// level order traversal of a given binary tree using iteration.
void levelorder(tree root, vector<int>& vec) {
  Visit the root.
```

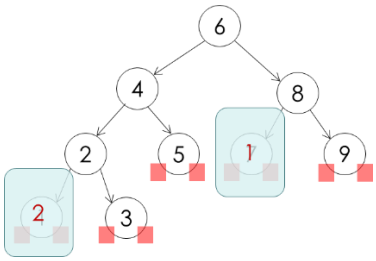
```
  if it is not null, push it to queue.
  while queue is not empty
    queue.front() - get the node from the queue
    visit the node (save the key in vec).
    if its left child is not null, push it to queue.
   if its right child is not null, push it to queue.
   queue.pop() - remove the node in the queue.
  }
}
```

레벨 순회를 이해하고 구현하면 그다음 함수인 growBT()가 levelorder() 함수의 또 다른 변형임을 알 수 있습니다.

Step 1.3: growBT() - 레벨 순서대로 노드 추가하기

이 함수는 key 를 가진 노드를 삽입하고 이진 트리의 루트를 반환합니다.

아래에 나온 트리의 경우, 레벨 순서대로 노드가 추가될 때 처음 추가될 자리는 빈 노드인 8 의 왼쪽 자식 노드입니다. 그다음으로 추가될 자리는 빈 노드인 2 의 왼쪽 자식 노드입니다.



핵심은 queue 를 이용하여 지정된 트리를 반복적으로 레벨 순회하는 것입니다. 필요하다면 다음 알고리즘을 참고하세요.

```
First, push the root to the queue.
Then, while the queue is not empty,
    Get the front() node on the queue
    If the left child of the node is empty,
       make new key as left child of the node. - break and return;
    else
       add it to queue to process later since it is not nullptr.
    If the right child is empty,
       make new key as right child of the node. - break and return;
    else
       add it to queue to process later since it is not nullptr.
    Make sure that you pop the queue finished.
    Do this until you find a node whose either left or right is empty.
```

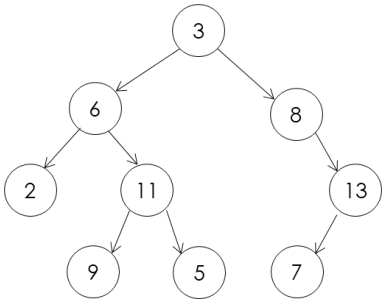
코드의 도입부는 아래와 같이 작성할 수 있습니다.

```
tree growBT(tree root, int key) {
  if (root == nullptr)
    return new TreeNode(key);
  queue<tree> q;
  q.push(root);
  while (!q.empty()) {
```

```
    // your code here
  }
  return root; // returns the root node
}
```

Step 1.4: findPath() & findPathBack()

findPath(): 고유 키를 가진 이진 트리가 주어졌을 때, 루트부터 지정된 노드 x 로 가는 경로를 반환합니다. 예를 들어, 노드 2 로 가는 경로는 [3, 6, 2], 노드 9 → [3, 6, 11, 9], 노드 13 → [3, 8, 13]입니다.



Intuition:
현재 노드를 벡터(path)에 push 합니다. 현재 노드가 x 라면 true 를 반환합니다. 트리의 왼쪽과 오른쪽을 재귀적으로 반복해 내려가면서 x 를 찾습니다. x 를 찾으면 true 를 반환하고, 찾지 못하면 현재 노드를 제거합니다. 이 알고리즘은 preorder()의 개념에서 유래되었습니다.

알고리즘:

- **루트 = nullptr** 이면 false 를 반환합니다. [base case]
- 루트의 키를 **벡터**에 push 합니다.
- **루트의 키 = x** 이면 true 를 반환합니다.
- 루트의 왼쪽 또는 오른쪽 하위 트리에서 x 를 재귀적으로 찾습니다.
    - 루트의 왼쪽 또는 오른쪽 하위 트리에 **x** 가 존재하면 true 를 반환합니다.
      x 가 존재하지 않으면 루트의 키를 **벡터**에서 제거하고 false 를 반환합니다.

findPathBack(): 유사한 알고리즘을 이용하여 루트로 가는 경로를 찾을 수 있습니다.

Intuition:
현재 노드가 x 이거나 트리의 왼쪽 또는 오른쪽 하위 트리를 재귀적으로 탐색하다가 x 를 찾으면, 현재 노드를 **벡터**에 push 합니다. x 를 찾지 못하면 false 를 반환합니다. 이 알고리즘은 postorder()의 개념에서 유래되었습니다. 노드 x 를 찾은 뒤 루트를 역추적합니다.

알고리즘:

- **루트 = nullptr** 이면 false 를 반환합니다. [base case]
- **루트의 키 = x** 이거나 (재귀적으로 탐색하는 동안) **x** 가 루트의 왼쪽 또는 오른쪽 하위 트리에 존재하면,
    - 루트의 키를 **벡터**에 push 합니다. (이때 재귀적 역추적을 수행합니다)

    - true 를 반환합니다.
- 존재하지 않으면
    - false 를 반환합니다.

노트: 두 함수를 독립적으로 코딩해야 합니다. 한 함수가 다른 함수를 호출하여 reverse 하지 않아야 합니다.

과제 제출

- 소스 파일 상단에 아래와 같이 아너 코드 문장을 적고 서명하세요.
  On my honor, I pledge that I have neither received nor provided improper assistance in the completion of this assignment.
  서명: _____ 분반: _____ 학번: _____
- 제출하기 전에 코드가 제대로 컴파일이 되고 실행되는지 확인하세요. 제출 직전에 급하게 코드를 수정한 후 코드가 제대로 컴파일이 될 거라고 짐작하지 않는 게 좋습니다. "거의" 작동하는 코드도 틀린 것입니다.
- 과제가 컴파일 및 실행된다면, 마감 기한 전까지 과제의 일부만 완성했더라도 제출하기 바랍니다. 컴파일 및 실행되지 않는다면 제출하지 마세요. 마감 시간 이후 24 시간 이내 제출하면, 만점에서 25% 감점하고 채점합니다. 그 이상 늦은 것은 채점하지 않으며, 0 점 처리합니다.
- 제출 후, **마감 기한 전까지** 수정 및 재제출이 가능합니다. 파일 하나만 수정하더라도 해당 파일과 관련된 파일들을 모두 재제출해야 합니다. 재제출 횟수는 제한 없습니다. 마감 기한 전에 **가장 마지막으로** 제출된 파일을 채점할 것입니다.

제출 파일 목록

- **Lab - Binary Tree:** Step 1.1 ~ 1.4
    - tree.cpp

참고 문헌

1. Recursion :
2. Recursion: