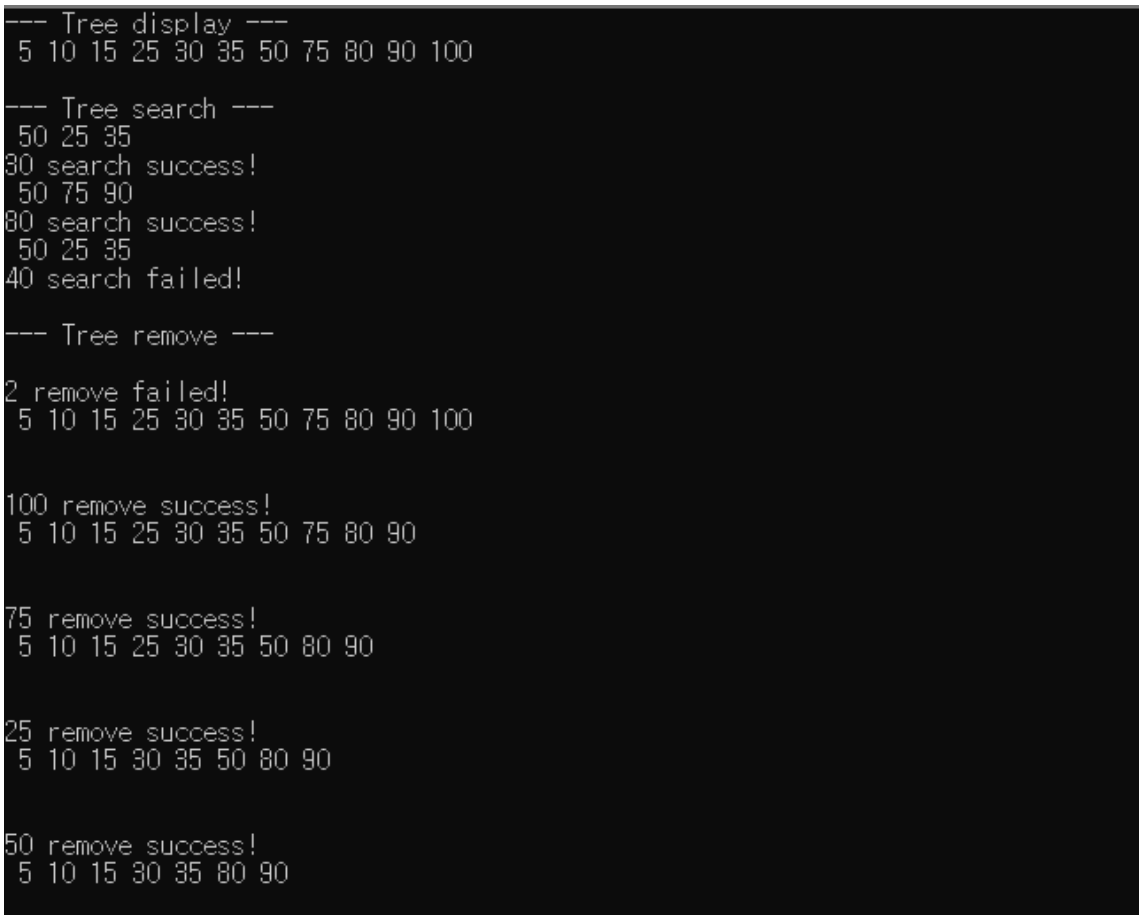
HW10

2019203102 유지성

실행 결과



코드 설명

```
56  void Tree::search(int data) {
57      search(this->rootnode, data);
58      return;
59  }
60  void Tree::search(TreeNode* curNode, int data) {
61      if (curNode == NULL) {
62          std::cout << "₩n" << data << " search failed!₩n";
63          return;
64      }
65
66      if (curNode->getData() == data) {
67          std::cout << "₩n" << data << " search success!₩n";
68          return;
69      }
70      else if (curNode->getData() < data) {
71          std::cout << " " << curNode->getData();
72          search(curNode->getRight(), data);
73      }
74      else {
75          std::cout << " " << curNode->getData();
76          search(curNode->getLeft(), data);
77      }
78  }
```

search는 재귀함수를 이용해 구현하였다. 만약 트리의 잎부분까지 갔는데도 찾으려는 대상이 없으면 search failed를 출력하도록 하였다. 노드의 값 출력은 루트 노드에서부터 거쳐온 값들을 순서대로 출력하도록 하였다.

```
79  void Tree::remove(int data) {
80      remove(NULL, this->rootnode, data, NONE);
81      return;
82  }
83  void Tree::remove(TreeNode* beforeNode, TreeNode* curNode, int data, int flag) {
84      if (curNode == NULL) {
85          std::cout << "₩n" << data << " remove failed!₩n";
86          return;
87      }
88
```

remove 함수 또한 재귀함수를 이용해 구현하였다. search와 마찬가지로 찾는 대상이 없으면 실패 문자를 출력하도록 하였다.

remove 함수는 찾는 값과 node의 data값의 차이에 따라 크게 두 가지로 구현하였다.

찾는 데이터의 값이 node의 data와 일치하지 않을 때

```
136     else if (curNode->getData() < data) {
137         remove(curNode, curNode->getRight(), data, RIGHT);
138     }
139     else {
140         remove(curNode, curNode->getLeft(), data, LEFT);
141     }
```

각각의 상황에 맞게 재귀적으로 구현하였다.

찾는 데이터의 값이 node의 data와 일치할 때

```
89      if (curNode->getData() == data) {
90          if (curNode->getLeft() != NULL) {
91              TreeNode* oldNode = new TreeNode();
92              TreeNode* newNode = curNode;
93              TreeNode* tempNode = new TreeNode();
94
95              oldNode->setLeft(curNode->getLeft());
96              oldNode->setRight(curNode->getRight());
97
98              tempNode->setData(curNode->getData());
99              tempNode->setLeft(curNode->getLeft());
100             tempNode->setRight(curNode->getRight());
101
102             tempNode = remove_max_left(tempNode, curNode->getLeft(), newNode);
103             tempNode->setLeft(oldNode->getLeft());
104             tempNode->setRight(oldNode->getRight());
105
106             if (beforeNode != NULL) {
107                 if (flag == LEFT)
108                     beforeNode->setLeft(tempNode);
109                 else
110                     beforeNode->setRight(tempNode);
111                 delete oldNode;
112             }
113             else {
114                 this->rootnode = tempNode;
115                 delete curNode;
116             }
```

현재 node의 left가 비어있지 않을 때 :

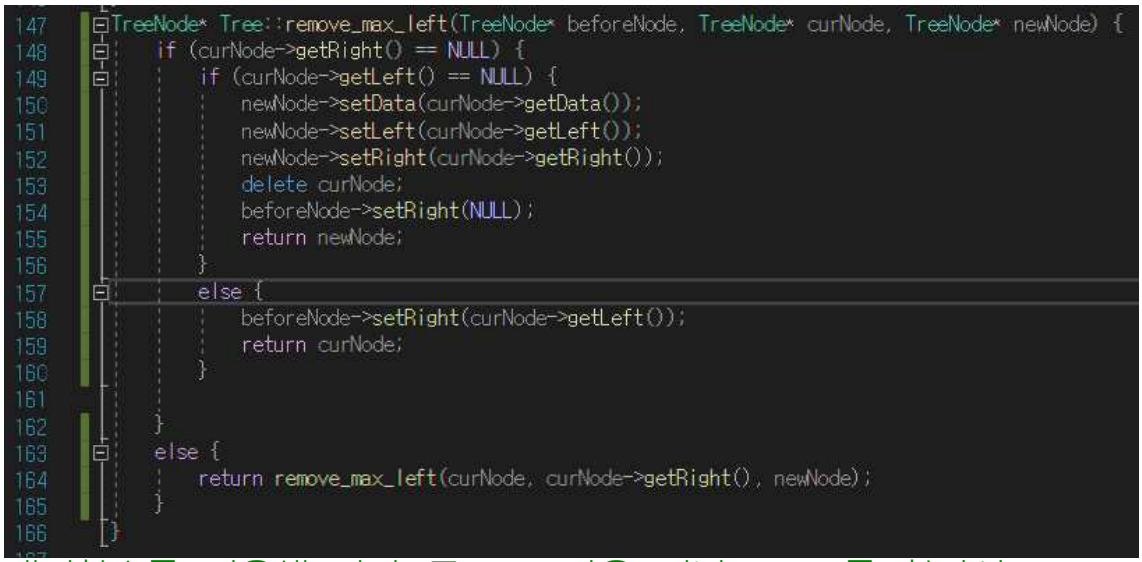
현재 node의 left, right 값을 저장해줄 oldNode를 만든다.

또한 현재 node의 값을 그대로 복사한 tempNode를 만든다.

remove_max_left를 이용해 왼쪽 트리중에서 data가 가장 큰

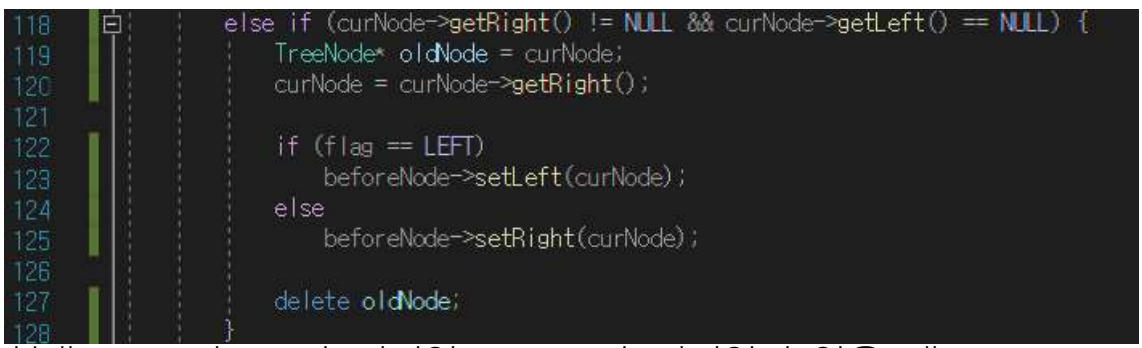
node를 받아온다. remove_max_left를 이용해 받아온 node에는 자리 옮기기 전 node의 정보가 저장되어 있으므로 left값과 right 값을 바꿔준다. 그 다음 현재 노드의 이전 노드의 left 혹은 right가 현재 노드를 가리키도록 바꿔준다. 만약 현재 노드가 root라면 이전 노드가 없으므로 이전 노드값은 수정하지 않고 tree의 rootnode를 바꿔준다.

remove_max_left



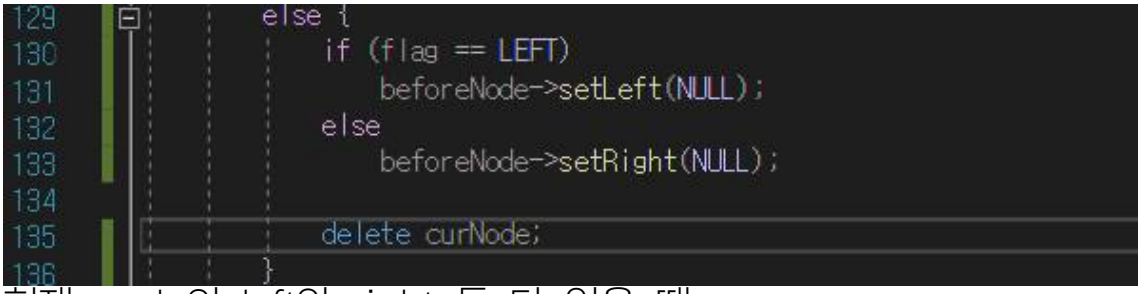
```
147 TreeNode* Tree::remove_max_left(TreeNode* beforeNode, TreeNode* curNode, TreeNode* newNode) {
148     if (curNode->getRight() == NULL) {
149         if (curNode->getLeft() == NULL) {
150             newNode->setData(curNode->getData());
151             newNode->setLeft(curNode->getLeft());
152             newNode->setRight(curNode->getRight());
153             delete curNode;
154             beforeNode->setRight(NULL);
155             return newNode;
156         }
157         else {
158             beforeNode->setRight(curNode->getLeft());
159             return curNode;
160         }
161
162     }
163     else {
164         return remove_max_left(curNode, curNode->getRight(), newNode);
165     }
166 }
```

재귀함수를 이용해 가장 큰 data값을 지닌 node를 찾아서 return 해 준다. 함수를 사용하면서 node간의 연결이 끊어지지않게 유지시켜준다.



```
118         else if (curNode->getRight() != NULL && curNode->getLeft() == NULL) {
119             TreeNode* oldNode = curNode;
120             curNode = curNode->getRight();
121
122             if (flag == LEFT)
123                 beforeNode->setLeft(curNode);
124             else
125                 beforeNode->setRight(curNode);
126
127             delete oldNode;
128         }
```

현재 node의 left가 비어있고 right가 비어있지 않을 때:
현재 node를 right node로 바꾸고 난 다음 이전 node와 결합시켜준다.



현재 node의 left와 right 둘 다 없을 때:

현재 node를 삭제하고 이전 node와의 연결을 NULL로 초기화한다.

고찰

remove를 구현하는 것이 상당히 어려웠다. 분명히 다 이해하고 제대로 코딩하였다고 생각했는데 런타임시 생기는 알수 없는 버그 때문에 애를 많이 먹었다. 이것저것 고치다보니 코드의 가독성이 떨어진 것 같아 아쉽다.