자료구조 HW7

B711222 박조은
Hongik University
mrnglory@mail.hongik.ac.kr

November 14, 2019

I. List of Source Files

- hw7
  - hw7.cpp
  - bt.h

II. hw7

i. hw7.cpp

```cpp
#include "bt.h"
#include <iostream>
using namespace std;

int main()
{
	Tree<double> tree;
	double dval;

	cout << "Enter doubles:\n";

	while(cin >> dval)
		tree.Insert(dval);

	cout << endl << "Preorder traversal: ";
		tree.Preorder();

	cout << endl << "Inorder traversal: ";
		tree.Inorder();

	cout << endl << "Postorder traversal: ";
		tree.Postorder();

	cout << endl << "Levelorder traversal: ";
		tree.Levelorder();

	cout << endl;
}
```

ii. bt.h

```cpp
#ifndef TREE_H
#define TREE_H
#include <iostream>
#include <queue>
using namespace std;

template <class T>

struct Node
{
    Node(T d, Node<T> *left = 0, Node<T> *right = 0)
        : data(d), leftChild(left), rightChild(right) {}

    Node<T> *leftChild;
    T data;
    Node<T> *rightChild;
    Node<T> *pre;
};

template <class T>

class Tree
{
public:
    Tree() {root = 0; } // empty tree

    void Insert(T &value)
    {
        Insert(root, value);
    }

    void Preorder()
    {
        Preorder(root);
    }

    void Inorder()
    {
        Inorder(root);
    }

    void Postorder()
    {
        Postorder(root);
    }
```

```
46
47      void Levelorder();
48
49  private:
50      void Visit(Node<T> *);
51      void Insert(Node<T> *&, T &);
52      void Preorder(Node<T> *);
53      void Inorder(Node<T> *);
54      void Postorder(Node<T> *);
55
56      Node<T> *root;
57  };
58
59  template <class T>
60  void Tree<T>::Visit(Node<T> *ptr)
61  {
62      cout << ptr -> data << " ";
63  }
64
65  template <class T>
66  void Tree<T>::Insert(Node<T>* &ptr, T &value)
67  {
68      // Insert 의 helper 함수
69      if (ptr == 0)
70          ptr = new Node<T>(value);
71
72      else if (value < ptr -> data)
73          Insert(ptr -> leftChild, value);
74
75      else if (value > ptr -> data)
76          Insert(ptr -> rightChild, value);
77
78      else
79          cout << endl << "Duplicate value" << value << "
                ↪ ignored\n";
80  }
81
82  // Preorder, Inorder, Postorder, Levelorder 함수를
        ↪ 구현하시오.
83  // Levelorder(교재 p266 참조하되 STL 큐를 이용) 를
        ↪ 구현하시오.
84
85  template <class T>
86  void Tree<T>::Preorder(Node<T> *currentNode)
87  {
88      // Workhorse traverses the subtree rooted at
            ↪ currentNode.
89      // The workhorse is declared
90      // as a private member function of Tree.
91
92      if(currentNode)
93      {
94          Visit(currentNode);
95          Preorder(currentNode -> leftChild);
96          Preorder(currentNode -> rightChild);
97      }
98  }
99
100 template <class T>
101 void Tree<T>::Postorder(Node<T> *currentNode)
102 {
103     // Workhorse traverses the subtree rooted at
            ↪ currentNode.
104     // The workhorse is declared
105     // as a private member function of Tree.
106
107     if(currentNode)
108     {
109         Postorder(currentNode -> leftChild);
110         Postorder(currentNode -> rightChild);
111         Visit(currentNode);
112     }
113 }
114
115 template <class T>
116 void Tree<T>::Inorder(Node<T> *root)
117 {
118     // Workhorse traverses the subtree rooted at
            ↪ currentNode.
119     // The workhorse is declared
120     // as a priate member function of Tree.
121
122     Node<T> * currentNode, * pre;
123     currentNode = root;
124
125     if(root == NULL)
126         return;
127
128     while(currentNode != NULL)
129     {
130         if(currentNode -> leftChild == NULL)
131         {
132             Visit(currentNode);
133             currentNode = currentNode -> rightChild;
134         }
135
136         else
137         {
138             pre = currentNode -> leftChild;
139
140             while(pre -> rightChild != NULL && pre
                    ↪ -> rightChild != currentNode)
141                 pre = pre -> rightChild;
142
143             if(pre -> rightChild == NULL)
144             {
145                 pre -> rightChild = currentNode;
146                 currentNode = currentNode ->
                        ↪ leftChild;
147             }
148
149             else
```

```
150                 {
151                     pre -> rightChild = NULL;
152                     Visit(currentNode);
153                     currentNode = currentNode ->
                            ↪ rightChild;
154                 }
155             }
156         }
157     }
158
159     template <class T>
160     void Tree<T>::Levelorder()
161     {
162         queue<Node<T>*> q;
163         Node<T> * currentNode = root;
164
165         while(currentNode)
166         {
167             Visit(currentNode);
168
169             if(currentNode -> leftChild)
170                 q.push(currentNode -> leftChild);
171
172             if(currentNode -> rightChild)
173                 q.push(currentNode -> rightChild);
174
175             if(q.empty())
176                 return;
177
178             currentNode = q.front(); // 큐에서 꺼내자.
179             q.pop();
180         }
181     }
182
183     #endif
```

```
/*
 *
 * line 9-18: Node 정의
 * line 17: Non recursion inorder traversal of Threaded binary tree 를 위해 추가한 포인터, predecessor Node
 * line 22-57: Tree 정의
 * line 60-63: Visit 시 출력 패턴 정의
 * line 66-80: 입력 값을 left 혹은 right child 로서 Insert 하는 경우를 커버하는 함수 정의
 * line 86-98: root -> left child -> right child visiting Preorder function definition
 * line 108-120: left child -> right child -> root visiting Postorder function definition
 * line 122-157: Inorder traversal of Threaded binary tree (a.k.a. Morris inorder traversal using threading)
 * line 130-134, 138: header node = most left node 의 leftChild 및 most right node 의 rightChild 가 가리키게 함.
 * line 159-181: queue 사용하여 root -> left child -> right child 방문하되, 각 레벨별로 most left node 에서 most right node 모두를 방문함.

*/
```

iii. Results

iii.1 makefile

```
1  hw7: hw7.o
2          g++ -o hw7 hw7.o
3  hw7.o: bt.h
```

iii.2 compile

```
1  [B711222@localhost hw7d]$ hw7
2  Enter doubles:
3  35.3 15.7 81.5 4.5 66.7 91.2 2.3 5.2 88.2 94.5
4
5  Preorder traversal: 35.3 15.7 4.5 2.3 5.2 81.5 66.7 91.2 88.2
       ↪ 94.5
6  Inorder traversal: 2.3 4.5 5.2 15.7 35.3 66.7 81.5 88.2 91.2
       ↪ 94.5
7  Postorder traversal: 2.3 5.2 4.5 15.7 66.7 88.2 94.5 91.2 81.5
       ↪ 35.3
8  Levelorder traversal: 35.3 15.7 81.5 4.5 66.7 91.2 2.3 5.2
       ↪ 88.2 94.5
```