O-O Programming & Data Structure Lab. 10

10. 서적 정보 관리 및 탐색을 위한 균형화된 이진 탐색 트리

10.1 class Book

```
class Book
{
	friend ostream& operator<<(ostream& fout, Book& bk)
	{
		fout.setf(ios::left);
		fout << "[" << setw(8) << bk.title << ", " << setw(8) << bk.author << ", " << bk.pubDate << "]";
		return fout;
	}
public:
	Book(string bk_title, string bk_author, Date dt) :
		title(bk_title), author(bk_author), pubDate(dt){}
	string& getTitle() { return title; }
	string getAuthor() { return author; }
	Date getPubDate() { return pubDate; }
	void setTitle(string bk_title) { title = bk_title; }
	void setAuthor(string bk_author) { author = bk_author; }
private:
	string title;
	string author;
	Date pubDate;    // date of publish
};
```

10.2 T_Entry

```
template<typename K, typename V>
class T_Entry
{
	friend ostream& operator<<(ostream& fout, T_Entry<K, V>& entry)
	{
		if (entry.getValue() != NULL)
			fout << "(" << setw(8) << entry.getKey() << ": " << *(entry.getValue()) << ")";
		return fout;
	}
public:
	T_Entry(K key, V value) { _key = key; _value = value; }
	T_Entry() { } // default constructor
	~T_Entry() {}
	void setKey(const K& key) { _key = key; }
	void setValue(const V& value) { _value = value; }
	K getKey() const { return _key; }
	V getValue() const { return _value; }
	bool operator>(const T_Entry<K, V>& right) const { return (_key > right.getKey()); }
	bool operator>=(const T_Entry<K, V>& right) const { return (_key >= right.getKey()); }
	bool operator<(const T_Entry<K, V>& right) const { return (_key < right.getKey()); }
	bool operator<=(const T_Entry<K, V>& right) const { return (_key <= right.getKey()); }
	bool operator==(const T_Entry<K, V>& right) const { return ((_key == right.getKey()) && (_value ==
right.getValue())); }
	T_Entry<K, V>& operator=(T_Entry<K, V>& right);
	void fprint(ostream fout);
private:
	K _key;
	V _value;
};
```

10.3 Binary Search Tree Node

```
template<typename K, typename V>
class T_BSTN {    // a node of the tree
public:
	T_BSTN() : entry(), pPr(NULL), pLc(NULL), pRc(NULL) { } // default constructor
	T_BSTN(T_Entry<K, V> e) : entry(e), pPr(NULL), pLc(NULL), pRc(NULL) { } // constructor
	K getKey() { return entry.getKey(); }
```

```
        V getValue() { return entry.getValue(); }
        T_Entry<K, V>& getEntry() { return entry; }
        void setEntry(T_Entry<K, V> e) { entry = e; }
        T_BSTN<K, V>* getpPr() { return pPr; }
        T_BSTN<K, V>* getpLc() { return pLc; }
        T_BSTN<K, V>* getpRc() { return pRc; }
        T_BSTN<K, V>** getppLc() { return &pLc; }
        T_BSTN<K, V>** getppRc() { return &pRc; }
        void setpPr(T_BSTN<K, V>* pTN) { pPr = pTN; }
        void setpLc(T_BSTN<K, V>* pTN) { pLc = pTN; }
        void setpRc(T_BSTN<K, V>* pTN) { pRc = pTN; }
        T_Entry<K, V>& operator*() { return entry; }
private:
        T_Entry<K, V>   entry;          // element value
        T_BSTN<K, V>*     pPr;          // parent
        T_BSTN<K, V>*     pLc;          // left child
        T_BSTN<K, V>*     pRc;          // right child
};
```

10.4 Binary Search Tree with Rebalancing

```
template<typename K, typename V>
class T_BST
{
public:
        T_BST(string nm) : _root(NULL), num_entry(0), name(nm) {}    // constructor
        string getName() { return name;   }
        int size() const { return num_entry; }
        bool empty() const { return num_entry == 0; }
        void clear();
        T_BSTN<K, V>* getRoot() { return _root;   }
        T_BSTN<K, V>** getRootAddr() { return &_root; }
        T_Entry<K, V>& getRootEntry() { return _root->getEntry(); }
        T_BSTN<K, V>* eraseBSTN(T_BSTN<K, V>** pp);
        void insertInOrder(const T_Entry<K, V> entry);
        void insertAndRebalance(T_Entry<K, V> e);
        void traversal_inOrder(T_BSTN<K, V>* p, T_Array<V>& array_value);
        void traversal_preOrder(T_BSTN<K, V>* pos, T_Array<V>& array_value);
        void traversal_postOrder(T_BSTN<K, V>* pos, T_Array<V>& array_value);
        T_BSTN<K, V>* searchBSTN(K k);
        T_Entry<K, V>& minEntry();
        T_Entry<K, V>& maxEntry();
        void fprint_with_Depth(ostream& fout);
        void fprint_inOrder(ostream& fout);
protected:
        T_BSTN<K, V>* _maxBSTN(T_BSTN<K, V>* subRoot);
        T_BSTN<K, V>* _minBSTN(T_BSTN<K, V>* subRoot);
        T_BSTN<K, V>* _insertInOrder(T_BSTN<K, V>** p, T_BSTN<K, V>* parenPos, const T_Entry<K, V> e);
        T_BSTN<K, V>* _insertAndRebalance(T_BSTN<K, V>** ppTN, T_BSTN<K, V>* pPr, T_Entry<K, V> e);
        T_BSTN<K, V>* _rotate_LL(T_BSTN<K, V> *pCurSubRoot);
        T_BSTN<K, V>* _rotate_RR(T_BSTN<K, V> *pCurSubRoot);
        T_BSTN<K, V>* _rotate_RL(T_BSTN<K, V> *pCurSubRoot);
        T_BSTN<K, V>* _rotate_LR(T_BSTN<K, V> *pCurSubRoot);
        int _getHeight(T_BSTN<K, V>* pTN);
        int _getHeightDiff(T_BSTN<K, V>* pTN);
        T_BSTN<K, V>* _reBalance(T_BSTN<K, V>** ppTN);
        T_BSTN<K, V>* _searchBSTN(T_BSTN<K, V>* pos, K k);
        void _fprint_with_Depth(T_BSTN<K, V>* pTN, ostream& fout, int depth);
        void _fprint_inOrder(T_BSTN<K, V>* pTN, ostream& fout);
private:
        T_BSTN<K, V>* _root; // pointer to the root
        int num_entry; // number of tree nodes
        string name;
};   // end of class T_BST
```

10.5 main() function

```
/** main.cpp    */
#include <iostream>
#include <fstream>
#include <iomanip>
#include <string>
#include "T_BSTv5.h"
#include "T_Entry.h"
#include "T_Array.h"
#include "Book.h"
#include "Date.h"

using namespace std;
#define NUM_BOOKS 15

void main()
{
	Book books[NUM_BOOKS] =
	{
		//Book( string bk_title, string bk_author, Date dt)
		Book(string("Book_01"), string("Kim"), Date(2020, 1, 1)),
		Book(string("Book_02"), string("Kim"), Date(2010, 1, 1)),
		Book(string("Book_03"), string("Kim"), Date(2013, 1, 1)),
		Book(string("Book_04"), string("Lee"), Date(2011, 1, 1)),
		Book(string("Book_05"), string("Hwang"), Date(2001, 1, 1)),
		Book(string("Book_06"), string("Choi"), Date(2003, 1, 1)),
		Book(string("Book_07"), string("Park"), Date(2009, 1, 1)),
		Book(string("Book_08"), string("Brown"), Date(2012, 1, 1)),
		Book(string("Book_09"), string("Alpha"), Date(1980, 1, 1)),
		Book(string("Book_10"), string("Chalie"), Date(1970, 1, 1)),
		Book(string("Book_11"), string("Tango"), Date(1985, 1, 1)),
		Book(string("Book_12"), string("Yankee"), Date(1977, 1, 1)),
		Book(string("Book_13"), string("Zulu"), Date(2018, 1, 1)),
		Book(string("Book_14"), string("Foxtrot"), Date(2015, 1, 1)),
		Book(string("Book_15"), string("Delta"), Date(2000, 1, 1)),
		/* */
	};

	ofstream fout("output.txt");
	if (fout.fail())
	{
		cout << "Fail to create output.txt for results !!" << endl;
		exit;
	}


	fout << "Input books[] array : " << endl;
	for (int i = 0; i < NUM_BOOKS; i++)
	{
		fout << books[i] << endl;
	}
	fout << endl;

	fout << endl << "Balanced Binary Search Tree (BBST) with key book-title" << endl;
	T_Entry<string, Book*> entry_title_pBK;
	T_BST<string, Book*> BBST_BK_keyTitle("BBST_BK_keyTitle");
	T_BSTN<string, Book*> *pRoot, **ppBBST_BK_root;
	ppBBST_BK_root = BBST_BK_keyTitle.getRootAddr();
	for (int i = 0; i < NUM_BOOKS; i++)
	{
		entry_title_pBK.setKey(books[i].getTitle());
		entry_title_pBK.setValue(&books[i]);
		//fout << "Insert inOrder (" << setw(3) << books[i] << ") into " << BBST_BK_keyTitle.getName() << endl;
		BBST_BK_keyTitle.insertAndRebalance(entry_title_pBK);
	}
	fout << "\nEntries in " << BBST_BK_keyTitle.getName() << " (in order of Book Title) : " << endl;
	BBST_BK_keyTitle.fprint_inOrder(fout);
	BBST_BK_keyTitle.fprint_with_Depth(fout);

	fout << endl << "Balanced Binary Search Tree (BBST) with key book-author" << endl;
```

```
        T_Entry<string, Book*> entry_Author_pBK;
    T_BST<string, Book*> BBST_BK_keyAuthor("BBST_BK_keyAuthor");
    T_BSTN<string, Book*> **ppRoot_BBST_BK_keyAuthor, *pBBST_BK_keyAuthor;
    ppRoot_BBST_BK_keyAuthor = BBST_BK_keyAuthor.getRootAddr();
    for (int i = 0; i < NUM_BOOKS; i++)
    {
        entry_Author_pBK.setKey(books[i].getAuthor());
        entry_Author_pBK.setValue(&books[i]);
        //fout << "Insert inOrder (" << setw(3) << books[i] << ") into " << BBST_BK_keyTitle.getName() << endl;
        BBST_BK_keyAuthor.insertAndRebalance(entry_Author_pBK);
    }
    fout << "\nEntries in " << BBST_BK_keyAuthor.getName() << " (in order of Book Author) : " << endl;
    BBST_BK_keyAuthor.fprint_inOrder(fout);
    BBST_BK_keyAuthor.fprint_with_Depth(fout);

    // Testing Search on Binary Search Tree
    string author = books[0].getAuthor();
    Date d1, d2;
    Book *pBk;
    T_Array<Book *> array_pBook(1, string("Array_Book"));
    d1.setDate(2010, 1, 1);
    d2.setDate(2015, 12, 31);
    pBBST_BK_keyAuthor = BBST_BK_keyAuthor.searchBSTN(author);
    BBST_BK_keyAuthor.traversal_inOrder(pBBST_BK_keyAuthor, array_pBook);
    fout << "Books of author (" << author << ") published during " << d1 << " ~ " <<d2 << ":" << endl;
    for (int i = 0; i < array_pBook.size(); i++)
    {
        if (array_pBook[i]->getAuthor() == author)
        {
            pBk = array_pBook[i];
            if ((pBk->getPubDate() >= d1) && (pBk->getPubDate() <= d2))
                fout << *(array_pBook[i]) << endl;
        }
    }

    fout << endl << "Balanced Binary Search Tree (BBST) with key publication date" << endl;
    T_Entry<Date, Book*> entry_PubDate_pBK;
    T_BST<Date, Book*> BBST_BK_keyPubDate("BBST_BK_keyPubDate");
    T_BSTN<Date, Book*> **ppRoot_BBST_BK_keyPubDate;
    ppRoot_BBST_BK_keyPubDate = BBST_BK_keyPubDate.getRootAddr();
    for (int i = 0; i < NUM_BOOKS; i++)
    {
        entry_PubDate_pBK.setKey(books[i].getPubDate());
        entry_PubDate_pBK.setValue(&books[i]);
        //fout << "Insert inOrder (" << setw(3) << books[i] << ") into " << BBST_BK_keyTitle.getName() << endl;
        BBST_BK_keyPubDate.insertAndRebalance(entry_PubDate_pBK);
    }
    fout << "\nEntries in " << BBST_BK_keyPubDate.getName() << " (in order of Book Publication Date) : " << endl;
    BBST_BK_keyPubDate.fprint_inOrder(fout);
    //BBST_BK_keyPubDate.fprint_with_Depth(fout);

    fout << "\nRemoving the root entry in sequence ..." << endl;
    for (int i = 0; i < NUM_BOOKS; i++)
    {
        pRoot = BBST_BK_keyTitle.getRoot();
        entry_title_pBK = pRoot->getEntry();
        fout << "\nremove " << entry_title_pBK << endl;
        BBST_BK_keyTitle.eraseBSTN(&pRoot);
        BBST_BK_keyTitle.fprint_with_Depth(fout);
    }

    fout << "\nClearing BBST_BKs . . . " << endl;
    BBST_BK_keyTitle.clear();
    BBST_BK_keyAuthor.clear();
    BBST_BK_keyPubDate.clear();
    fout << "All BBST_BKs cleared !! " << endl;

    fout.close();
}
```

10.6 Example output

```
Input books[] array :
[Book_01 , Kim     , (2020.1 .1 ) ]
[Book_02 , Kim     , (2010.1 .1 ) ]
[Book_03 , Kim     , (2013.1 .1 ) ]
[Book_04 , Lee     , (2011.1 .1 ) ]
[Book_05 , Hwang   , (2001.1 .1 ) ]
[Book_06 , Choi    , (2003.1 .1 ) ]
[Book_07 , Park    , (2009.1 .1 ) ]
[Book_08 , Brown   , (2012.1 .1 ) ]
[Book_09 , Alpha   , (1980.1 .1 ) ]
[Book_10 , Chalie  , (1970.1 .1 ) ]
[Book_11 , Tango   , (1985.1 .1 ) ]
[Book_12 , Yankee  , (1977.1 .1 ) ]
[Book_13 , Zulu    , (2018.1 .1 ) ]
[Book_14 , Foxtrot , (2015.1 .1 ) ]
[Book_15 , Delta   , (2000.1 .1 ) ]


Balanced Binary Search Tree (BBST) with key book-title

Entries in BBST_BK_keyTitle (in order of Book Title) :
BBST_BK_keyTitle : current size (15)
            (Book_15 : [Book_15 , Delta   , (2000.1 .1 ) ])
        (Book_14 : [Book_14 , Foxtrot , (2015.1 .1 ) ])
            (Book_13 : [Book_13 , Zulu    , (2018.1 .1 ) ])
    (Book_12 : [Book_12 , Yankee  , (1977.1 .1 ) ])
            (Book_11 : [Book_11 , Tango   , (1985.1 .1 ) ])
        (Book_10 : [Book_10 , Chalie  , (1970.1 .1 ) ])
            (Book_09 : [Book_09 , Alpha   , (1980.1 .1 ) ])
(Book_08 : [Book_08 , Brown   , (2012.1 .1 ) ])
            (Book_07 : [Book_07 , Park    , (2009.1 .1 ) ])
        (Book_06 : [Book_06 , Choi    , (2003.1 .1 ) ])
            (Book_05 : [Book_05 , Hwang   , (2001.1 .1 ) ])
    (Book_04 : [Book_04 , Lee     , (2011.1 .1 ) ])
            (Book_03 : [Book_03 , Kim     , (2013.1 .1 ) ])
        (Book_02 : [Book_02 , Kim     , (2010.1 .1 ) ])
            (Book_01 : [Book_01 , Kim     , (2020.1 .1 ) ])

Balanced Binary Search Tree (BBST) with key book-author

Entries in BBST_BK_keyAuthor (in order of Book Author) :
BBST_BK_keyAuthor : current size (15)
          (Zulu    : [Book_13 , Zulu    , (2018.1 .1 ) ])
       (Yankee  : [Book_12 , Yankee  , (1977.1 .1 ) ])
    (Tango   : [Book_11 , Tango   , (1985.1 .1 ) ])
          (Park    : [Book_07 , Park    , (2009.1 .1 ) ])
       (Lee     : [Book_04 , Lee     , (2011.1 .1 ) ])
          (Kim     : [Book_03 , Kim     , (2013.1 .1 ) ])
(Kim     : [Book_02 , Kim     , (2010.1 .1 ) ])
          (Kim     : [Book_01 , Kim     , (2020.1 .1 ) ])
       (Hwang   : [Book_05 , Hwang   , (2001.1 .1 ) ])
          (Foxtrot : [Book_14 , Foxtrot , (2015.1 .1 ) ])
              (Delta   : [Book_15 , Delta   , (2000.1 .1 ) ])
    (Choi    : [Book_06 , Choi    , (2003.1 .1 ) ])
          (Chalie  : [Book_10 , Chalie  , (1970.1 .1 ) ])
       (Brown   : [Book_08 , Brown   , (2012.1 .1 ) ])
          (Alpha   : [Book_09 , Alpha   , (1980.1 .1 ) ])
Books of author (Kim) published during (2010.1 .1 )  ~ (2015.12.31) :
[Book_02 , Kim     , (2010.1 .1 ) ]
[Book_03 , Kim     , (2013.1 .1 ) ]
```

```
Removing the root entry in sequence ...

remove (Book_08 : [Book_08 , Brown   , (2012.1 .1 ) ])
BBST_BK_keyTitle : current size (14)
            (Book_15 : [Book_15 , Delta   , (2000.1 .1 ) ])
        (Book_14 : [Book_14 , Foxtrot , (2015.1 .1 ) ])
            (Book_13 : [Book_13 , Zulu    , (2018.1 .1 ) ])
    (Book_12 : [Book_12 , Yankee  , (1977.1 .1 ) ])
            (Book_11 : [Book_11 , Tango   , (1985.1 .1 ) ])
        (Book_10 : [Book_10 , Chalie  , (1970.1 .1 ) ])
(Book_09 : [Book_09 , Alpha   , (1980.1 .1 ) ])
            (Book_07 : [Book_07 , Park    , (2009.1 .1 ) ])
        (Book_06 : [Book_06 , Choi    , (2003.1 .1 ) ])
            (Book_05 : [Book_05 , Hwang   , (2001.1 .1 ) ])
    (Book_04 : [Book_04 , Lee     , (2011.1 .1 ) ])
            (Book_03 : [Book_03 , Kim     , (2013.1 .1 ) ])
        (Book_02 : [Book_02 , Kim     , (2010.1 .1 ) ])
            (Book_01 : [Book_01 , Kim     , (2020.1 .1 ) ])

remove (Book_09 : [Book_09 , Alpha   , (1980.1 .1 ) ])
BBST_BK_keyTitle : current size (13)
            (Book_15 : [Book_15 , Delta   , (2000.1 .1 ) ])
        (Book_14 : [Book_14 , Foxtrot , (2015.1 .1 ) ])
            (Book_13 : [Book_13 , Zulu    , (2018.1 .1 ) ])
    (Book_12 : [Book_12 , Yankee  , (1977.1 .1 ) ])
        (Book_11 : [Book_11 , Tango   , (1985.1 .1 ) ])
(Book_10 : [Book_10 , Chalie  , (1970.1 .1 ) ])
            (Book_07 : [Book_07 , Park    , (2009.1 .1 ) ])
        (Book_06 : [Book_06 , Choi    , (2003.1 .1 ) ])
            (Book_05 : [Book_05 , Hwang   , (2001.1 .1 ) ])
    (Book_04 : [Book_04 , Lee     , (2011.1 .1 ) ])
            (Book_03 : [Book_03 , Kim     , (2013.1 .1 ) ])
        (Book_02 : [Book_02 , Kim     , (2010.1 .1 ) ])
            (Book_01 : [Book_01 , Kim     , (2020.1 .1 ) ])

. . . . .

remove (Book_14 : [Book_14 , Foxtrot , (2015.1 .1 ) ])
BBST_BK_keyTitle : current size (4)
    (Book_15 : [Book_15 , Delta   , (2000.1 .1 ) ])
(Book_03 : [Book_03 , Kim     , (2013.1 .1 ) ])
    (Book_02 : [Book_02 , Kim     , (2010.1 .1 ) ])
        (Book_01 : [Book_01 , Kim     , (2020.1 .1 ) ])

remove (Book_03 : [Book_03 , Kim     , (2013.1 .1 ) ])
BBST_BK_keyTitle : current size (3)
    (Book_15 : [Book_15 , Delta   , (2000.1 .1 ) ])
(Book_02 : [Book_02 , Kim     , (2010.1 .1 ) ])
    (Book_01 : [Book_01 , Kim     , (2020.1 .1 ) ])

remove (Book_02 : [Book_02 , Kim     , (2010.1 .1 ) ])
BBST_BK_keyTitle : current size (2)
(Book_15 : [Book_15 , Delta   , (2000.1 .1 ) ])
    (Book_01 : [Book_01 , Kim     , (2020.1 .1 ) ])

remove (Book_15 : [Book_15 , Delta   , (2000.1 .1 ) ])
BBST_BK_keyTitle : current size (1)
(Book_01 : [Book_01 , Kim     , (2020.1 .1 ) ])

remove (Book_01 : [Book_01 , Kim     , (2020.1 .1 ) ])
BBST_BK_keyTitle is empty now !!

Clearing BBST_BKs . . .
All BBST_BKs cleared !!
```

<Oral Test 10>

(1) 이진 탐색 트리에서의 새로운 트리 노드 입력에서 재 균형화 (re-balancing)를 실행하지 않는 경우 어떤 문제점이 있는가에 대하여 설명하라.

<Key Points>

(1) 탐색, 삽입, 삭제 동작의 성능 관점에서 비교하고, Big Oh 방식으로 표시 할 것

항목	균형화를 하지 않을 때	균형화를 할 때
탐색 (search)	- 평균: - 최대:	- 평균: - 최대:
트리노드 삽입 (insert)	- 평균: - 최대:	- 평균: - 최대:
트리노드 삭제 (erase)	- 평균: - 최대:	- 평균: - 최대:

(2) 이진 탐색 트리에서의 트리노드 삭제에서 재 균형화 (re-balancing)를 실행하는 세부 기능에 대하여 설명하라. 트리노드들의 관계를 그림으로 표현하고, psedo code 로 설명하라.

<Key Points>

(1) 삭제 대상 노드의 자식이 없는 경우

(2) 삭제 대상 노드의 자식이 하나만 있는 경우

(3) 삭제 대상 노드의 자식이 모두 있는 경우, 노드들의 서브트리 height 계산 및 height-difference 계산

(4) 왼쪽 자식 노드의 height가 더 높은 경우, in-order predecessor (ioPd) 찾기 및 노드 재배치

(5) 오른쪽 자식 노드의 height가 같거나 더 높은 경우, in-order successor(ioSs) 찾기 및 노드 재배치

(3) 이진 탐색 트리에서의 새로운 트리노드 입력에서 재 균형화를 실행하는 세부 절차인 rotate_LL, rotate_RR 에 대하여 설명하라. 트리노드들의 관계를 그림으로 표현하고 (rotate 를 수행하기 전과, 수행한 후 비교), psedo code 로 설명하라.

<Key Points>

(1) 왼쪽으로 연속 편중된 경우에 대한 rotate_LL()

(2) 오른쪽으로 연속 편중된 경우에 대한 rotate_RR()

(4) 이진 탐색 트리에서의 새로운 트리노드 입력에서 재 균형화를 실행하는 세부 절차인 rotate_LR, rotate_RL 에 대하여 설명하라. 트리노드들의 관계를 (rotate 를 수행하기 전과, 수행한 후 비교) 그림으로 표현하여 설명하라.

<Key Points>

(1) 왼쪽으로 편중되어 있고, 내부에서 오른쪽으로 편중되어 있는 경우에 대한 rotate_LR()

(2) 오른쪽으로 편중되어 있고, 내부에서 왼쪽으로 편중되어 있는 경우에 대한 rotate_RL()