객체지향프로그래밍과 자료구조

Ch 10. 연결형 리스트와 이진 탐색 트리

정보통신공학과

교수 김 영 탁

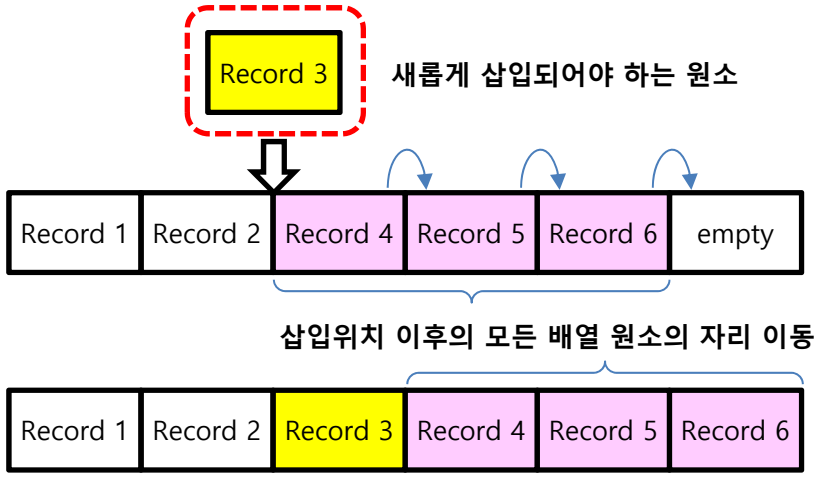
(Tel : +82-53-810-2497; E-mail : ytkim@yu.ac.kr)

Outline

- 배열 (array)에서의 원소 삽입과 삭제 성능 분석
- 연결형 리스트 Linked List (LL)
- 연결형 리스트에서의 반복자 (iterator)
- C++ 기반 이중 연결형 리스트 (Doubly Linked List, DLL) 구현
- 이중연결형리스트 템플릿 (T_DLL)의 응용
  - T_DLL-based Queue
- 이진탐색트리 (Binary Search Tree, BST)
- 이진 탐색트리에서의 자료 탐색을 위한 순차방문 (Traversal)
- C++ 기반 이진탐색트리 구현
  - T_Entry<K, V>
  - T_BSTN<K, V>
  - T_BST<K, V>
- 이진탐색트리의 재균형화 (rebalancing)

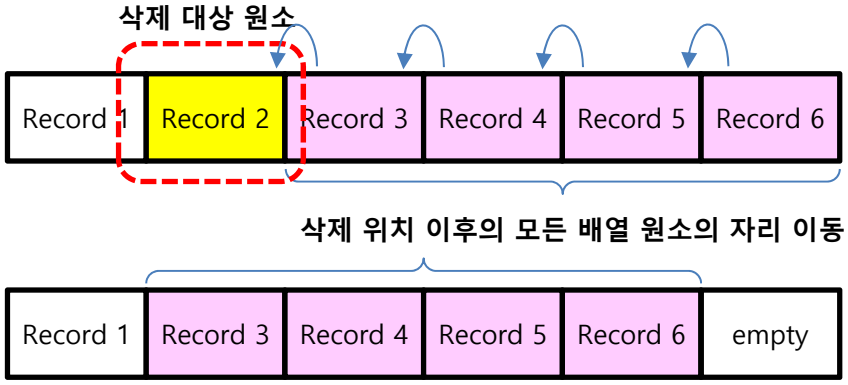
배열의 원소 삽입과 삭제 성능 분석

◆ 새로운 원소의 삽입



Operation	Complexity in Array
search	O(n)
insert	O(n)
remove	O(n)

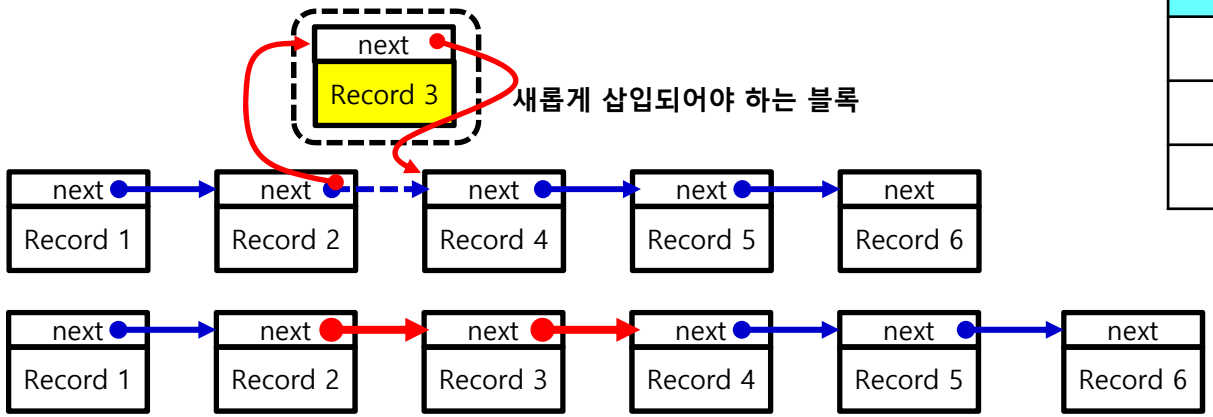
◆ 배열 원소의 삭제



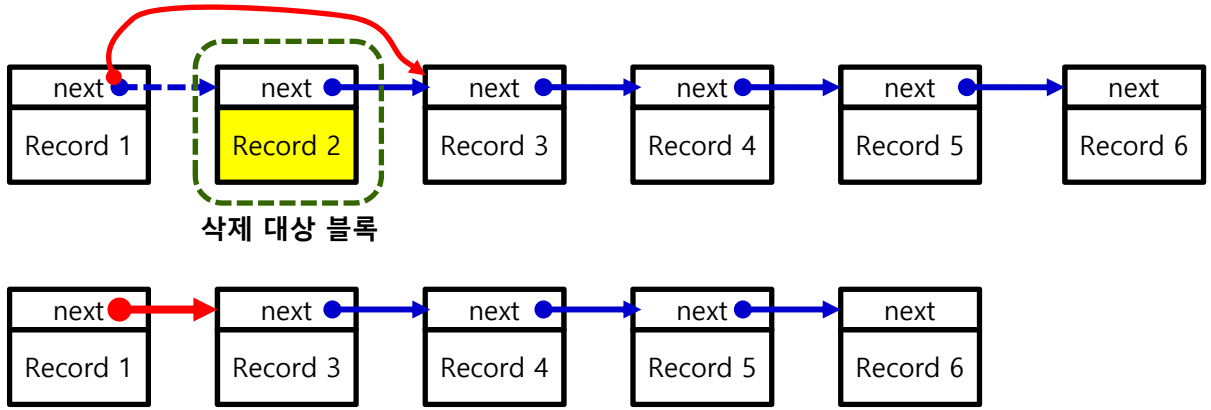
연결형 리스트에서의 삽입과 삭제

◆ Insert in Linked List

Operation	Complexity in Linked List
search	O(n)
insert	O(1)
remove	O(1)

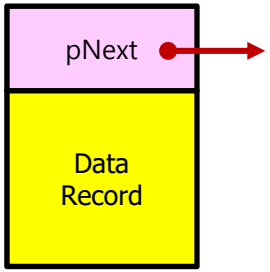


◆ Delete/Remove in Linked List

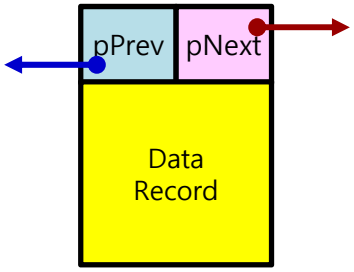


List Node

- List Node = data field + link field (next, prev)
- List Node with Data

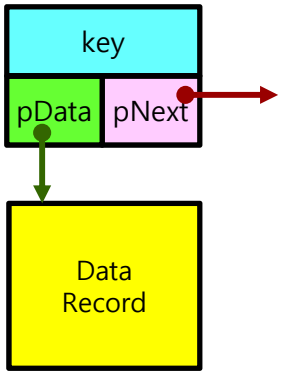


(a) List Node with Data for Singly Linked List

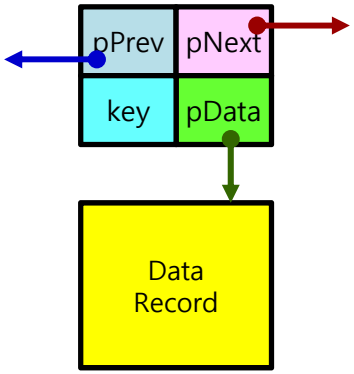


(b) Doubly Linked List Node with Data
for Doubly Linked List

- List Node with Data Pointer



(a) List Node with Data Pointer
for Singly Linked List



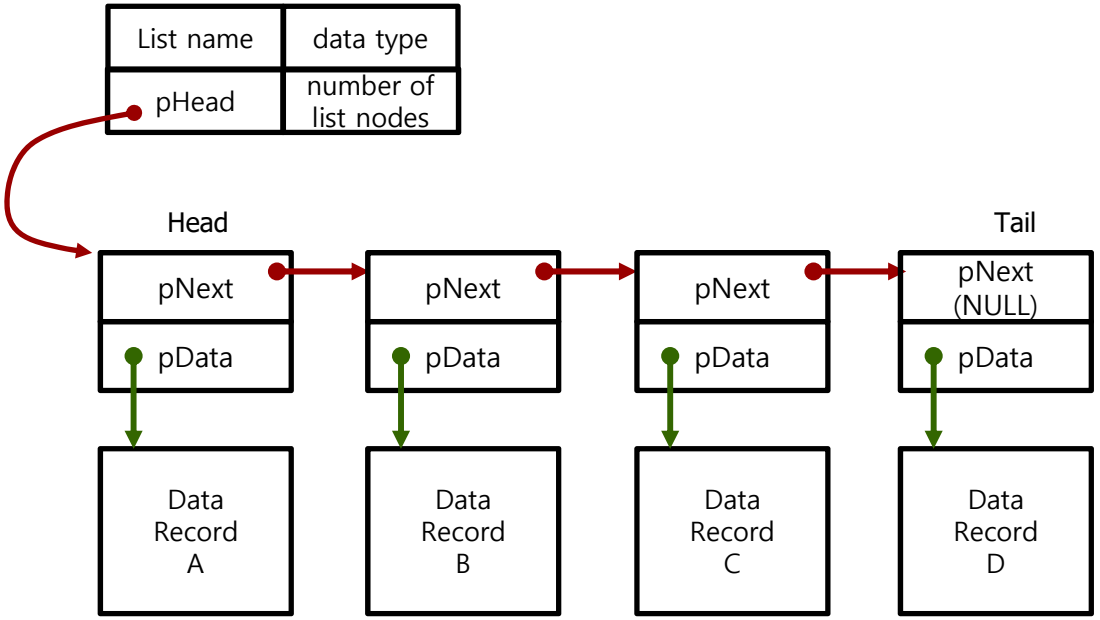
(b) List Node with Data Pointer
for Doubly Lniked List

Linked List (1)

List

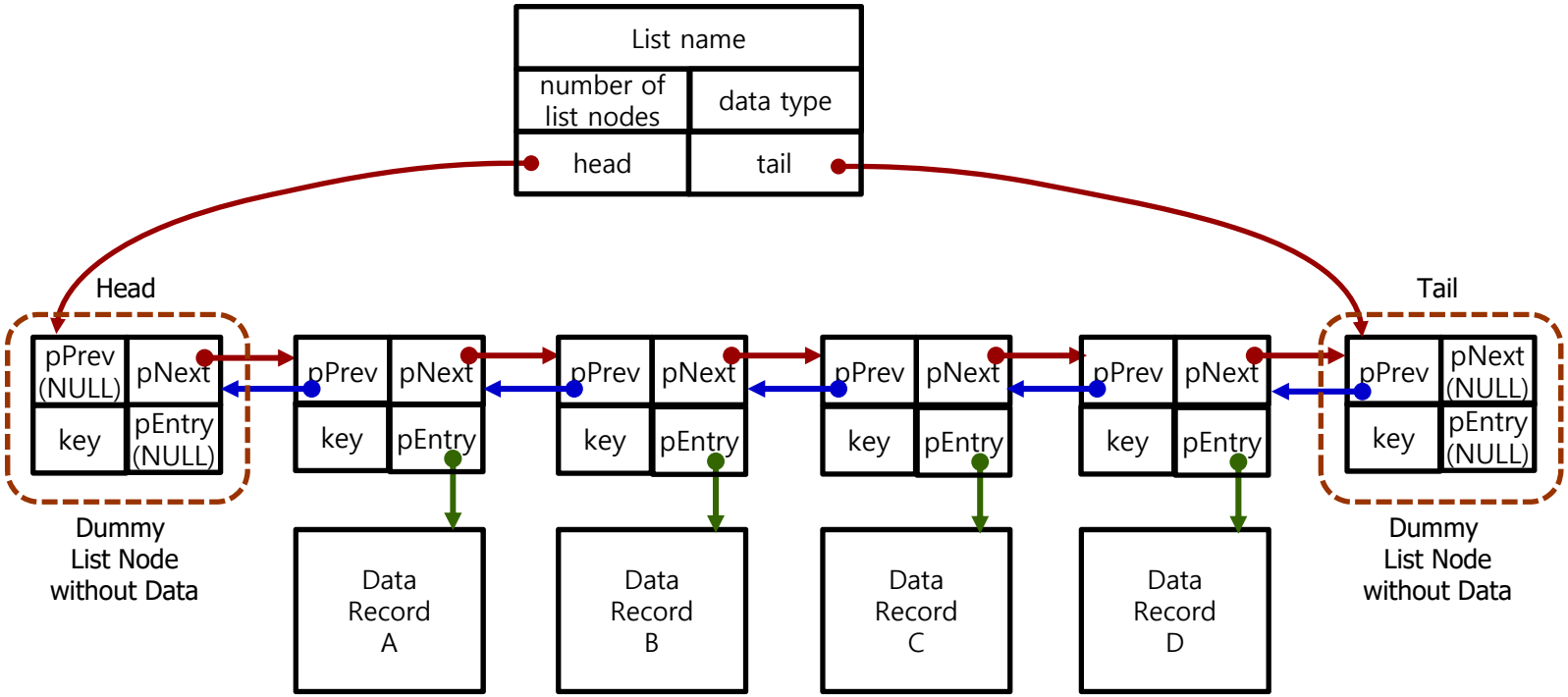
- linked list of nodes
- list abstract data type

Singly Linked List (SLL)



Linked List (2)

◆ Doubly Linked List (DLL)



Iterators in Data Structure

Containers and Positions/Iterators

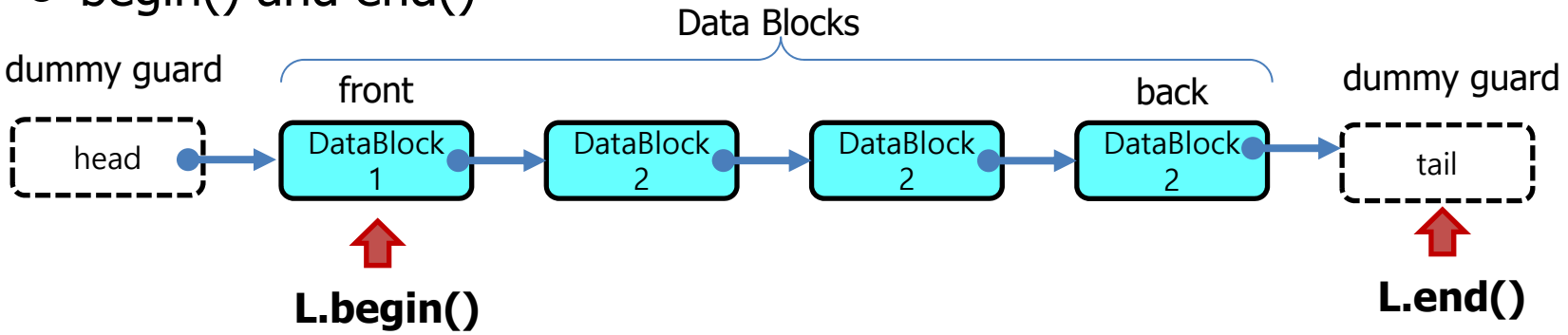
- **container** is a data structure that stores any collection of elements
  - elements in a container can be arranged in a linear order
- **position/iterator** is an abstract data type that is associated with a particular container
  - position supports element() that returns a reference to the element stored at this position

Iterator

- The Iterator abstract data type (ADT) models the notion of place within a data structure where a single object is stored
- It gives a unified view of diverse ways of storing data, such as
  - a cell of an array
  - a node of a linked list
- Just one method:
  - object p.element(): returns the element at position
  - In C++, it is convenient to implement this as ***p**

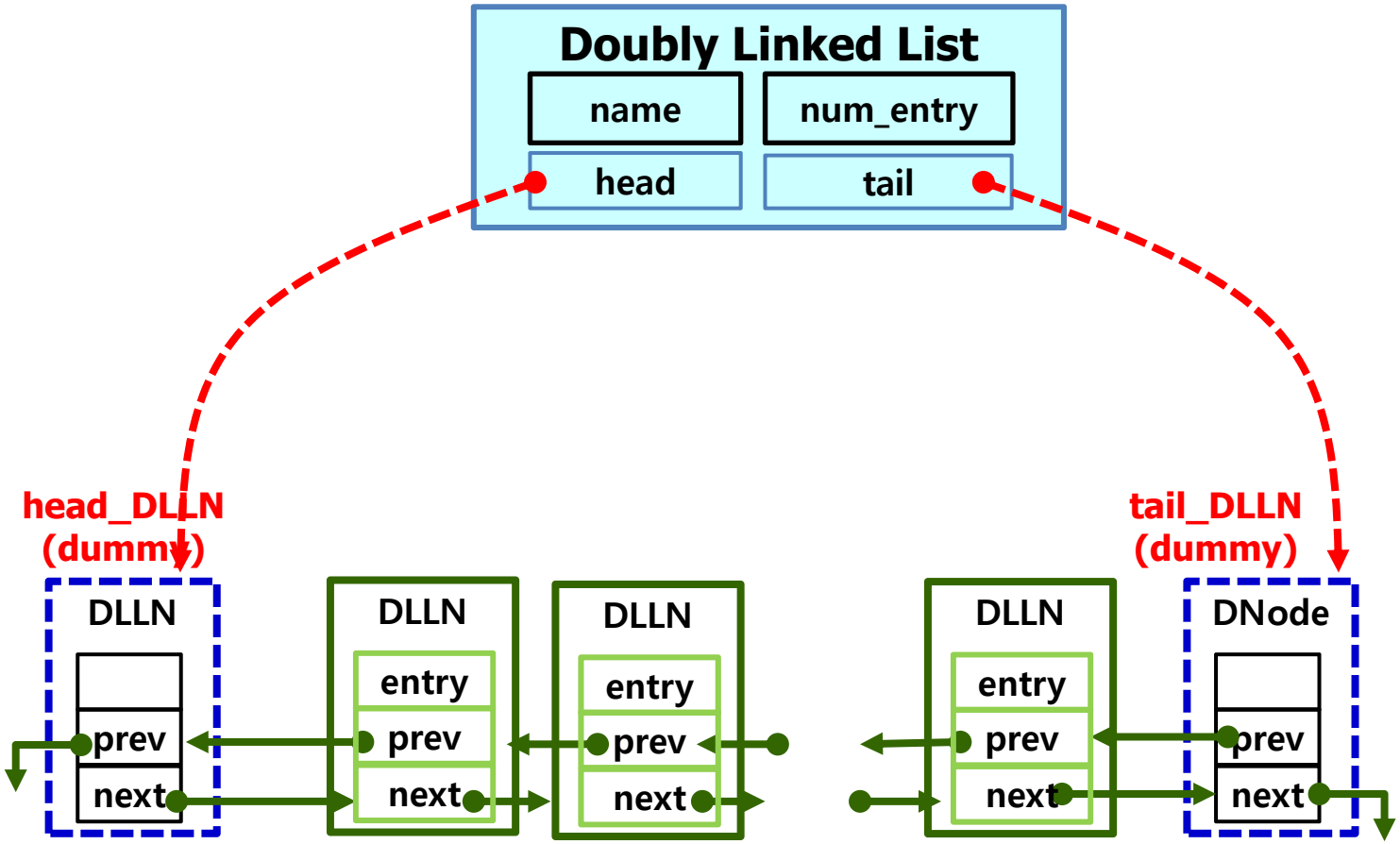
◆ Iterators as an extension of a pointer/position

- **iterator** is an extension of a position
  - iterator supports the ability to access node's element
  - iterator also supports the ability to ***navigate*** forwards (and possibly backwards) through the container
  - additional operation with iterator
    - **p.next()**
    - **++p, --p**
    - ***p**
    - **p = L.begin()**
    - **p = L.end()**
- begin() and end()



Implementation of Doubly Linked List

◆ Doubly Linked List 구현 구조



class T_DLLN<E>

```
/** T_DLLN.h  (1) */
#ifndef GENERICDNODE_H
#define GENERICDNODE_H
template<typename E>
class T_DLLN
{
public:
   T_DLLN(); // default constructor
   ~T_DLLN(); // destructor
   E* getElem() { return pE; }
   void setElem(E * pElem) { pE = pElem; }
   T_DLLN<E>* getPrev() { return prev; }
   T_DLLN<E>* getNext() { return next; }
   void setPrev(T_DLLN<E>* p) { prev = p; }
   void setNext(T_DLLN<E>* n) { next = n; }
private:
   E *pE;
   T_DLLN<E>* prev;
   T_DLLN<E>* next;
};
```

```
/** T_DLLN.h  (2) */

template<typename E>
T_DLLN<E>::T_DLLN()
{
   next = prev = NULL;
   pE = NULL;
}

template<typename E>
T_DLLN<E>::~T_DLLN()
{
   //cout << "Destruct of Generic Doubly
   //    Linked List Node" << endl;
}
#endif
```

class T_DLL<E>

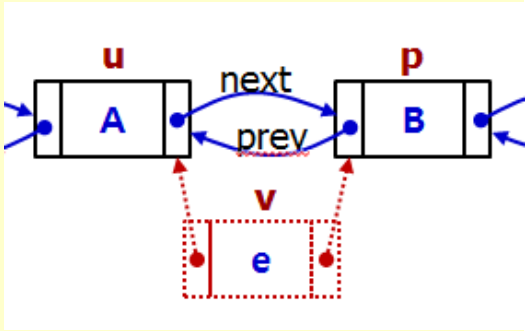
```
/** T_DLL.h (1)*/
#ifndef T_DLL_H
#define T_DLL_H
#include <mutex>
#include <iomanip>
#include "T_DLLN.h"
using namespace std;

template<typename E>
class T_DLL
{
public:
    T_DLL(string n);
    ~T_DLL();
    string getName() { return name; }
    void setName(string n) { name = n;}
    bool empty() const
       {return (head->getNext() == tail);}
    E* front() const;
    E* back() const;
    E* insertFront(E* pE);
    E* insertBack(E* pE);
    void removeFront();
    void removeBack();
    T_DLLN<E>* insertInOrder(E* pE);
    T_DLLN<E>* searchSequential(E* pE_Key);
    T_DLLN<E>* searchInOrder(E* pE_Key);
    int getNumEntry() { return num_entry; }
    string getElementType() { return element_type; }
    void fprint(ostream& fout, int elements_per_line);
```

```
/** T_DLL.h (2)*/
protected:
    T_DLLN<E>* _insert(T_DLLN<E>* v, E* pE);
    void _remove(T_DLLN<E>* v);
private:
    string name;
    int num_entry;
    string element_type;
    T_DLLN<E>* head;
    T_DLLN<E>* tail;
    mutex  cs_dll; // semaphore
};
template<typename E>
T_DLL<E>::T_DLL(string n)
{
    name = n;
    head = new T_DLLN<E>;
    tail = new T_DLLN<E>;
    head->setNext(tail);
    tail->setPrev(head);
    num_entry = 0;
    element_type = typeid(E).name();
}
template<typename E>
T_DLL<E>::~T_DLL()
{
    while (!empty())
       removeFront();
    delete head;
    delete tail;
}
```

```
/** T_DLL.h (3)*/

template<typename E>
E* T_DLL<E>::front() const
{
    E *pE;

    pE = (head->getNext())->getElem();
    return pE;
}

template<typename E>
E* T_DLL<E>::back() const
{
    E *pE;

    pE = (tail->getPrev())->getElem();
    return pE;
}


template<typename E>
T_DLLN<E>* T_DLL<E>::searchInOrder(E* pE_Key)
{
   T_DLLN<E>* p;

   p = head->getNext();
   while (p != tail)
   {
      if (*(p->getElem()) > *pE_Key)
         break;
      else
         p = p->getNext();
   }
   return p;
}
```

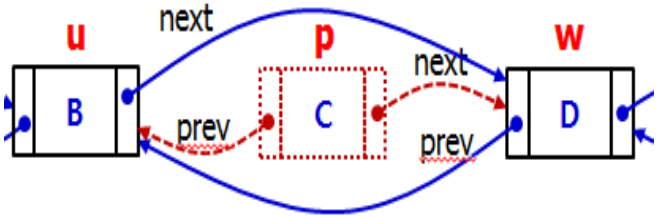
```
/** T_DLL.h (4)*/

template<typename E>
T_DLLN<E>* T_DLL<E>::
    searchSequential(E* pE)
{
    T_DLLN<E>* p;

    p = head->getNext();
    while (p != tail)
    {
        if (*(p->getElem()) == *pE)
           break;
        else
           p = p->getNext();
    }
    if (p == tail)
        return NULL;
        // for outside, just return NULL
    else
        return p;
}
```

```
/** T_DLL.h (5)*/

template<typename E>
T_DLLN<E>* T_DLL<E>::
    _insert(T_DLLN<E>*p, E* pE)
{
    cs_dll.lock();
    T_DLLN<E>* v = new T_DLLN<E>;
    T_DLLN<E>* u = p->getPrev();
    v->setElem(pE);
    v->setNext(p);
    p->setPrev(v);
    v->setPrev(u);
    u->setNext(v);

    num_entry++;
    cs_dll.unlock();
    return v;
}

template<typename E>
E* T_DLL<E>::insertFront(E* pE)
{
    T_DLLN<E>* u;
    E* pEntry;
    u = _insert(head->getNext(), pE);
    pEntry = u->getElem();
    return pEntry;
}
```




```
/** T_DLL.h (6)*/

template<typename E>
E* T_DLL<E>::insertBack(E* pE)
{
    T_DLLN<E>* v;
    E* pEntry;
    v = _insert(tail, pE);
    pEntry = v->getElem();
    return pEntry;
}

template<typename E>
T_DLLN<E>* T_DLL<E>::insertInOrder(E* pE)
{
    T_DLLN<E>* p;
    T_DLLN<E>* v;

    p = searchInOrder(pE);
    v = _insert(p, pE);

    return v;
}
```

```
/** T_DLL.h (7)*/

template<typename E>
void T_DLL<E>::_remove(T_DLLN<E>* p)
{
    cs_dll.lock();
    T_DLLN<E>* u = p->getPrev();
    T_DLLN<E>* w = p->getNext();
    u->setNext(w);
    w->setPrev(u);
    delete p;

    num_entry--;
    cs_dll.unlock();
}

template<typename E>
void T_DLL<E>::removeFront()
{
    _remove(head->getNext());
}

template<typename E>
void T_DLL<E>::removeBack()
{
    _remove(tail->getPrev());
}
```



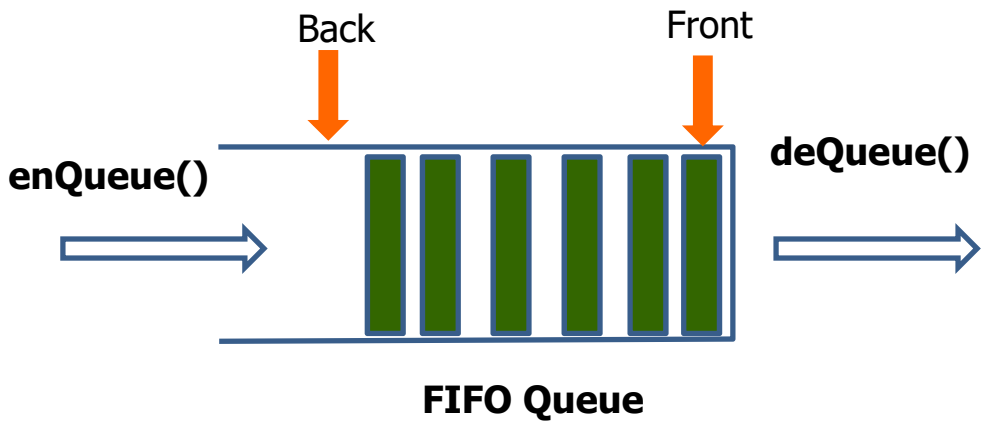
```
/** T_DLL.h (8)*/

template<typename E>
void T_DLL<E>::fprint(ostream& fout,
    int elements_per_line)
{
    E data;
    if (empty())
    {
        fout << "DLList is Empty now !!" << endl;
        return;
    }

    int count = 0;
    T_DLLN<E>* p = head->getNext();
    while (p != tail)
    {
        count++;
        data = *(p->getElem());
        fout << data  << "  ";
        p = p->getNext();
        if (elements_per_line == 1)
        {
            fout << endl;
            continue;
        }
        if ((count % elements_per_line) == 0)
              fout << endl;
    }
    fout << endl;
}
#endif
```

연결형 리스트 (Linked List) 템플릿 기반 응용 자료 구조 - Queue

First In First Out (FIFO) Queues

◆ The Queue stores arbitrary objects

- Insertions and deletions follow the first-in first-out (FIFO) scheme
- Insertions are at the rear of the queue and removals are at the front of the queue



FIFO Queues

- **The Queue abstract data type (ADT) stores arbitrary objects**
- **Insertions and deletions follow the first-in first-out scheme**
- **Insertions are at the rear of the queue and removals are at the front of the queue**
- **Main queue operations:**
  - enqueue(object): inserts an element at the end of the queue
  - dequeue(): removes the element at the front of the queue
- **Auxiliary queue operations:**
  - object front(): returns the element at the front without removing it
  - integer size(): returns the number of elements stored
  - boolean empty(): indicates whether no elements are stored
- **Exceptions**
  - Attempting the execution of dequeue or front on an empty queue throws an QueueEmpty

class T_DLL_Queue<T>

```
/* T_DLL_Queue.h  (1) */

#ifndef T_DLL_QUEUE_H
#define T_DLL_QUEUE_H
#include <Windows.h>
#include "T_DLL.h"

template<typename T>
class T_DLL_Queue
{
public:
    T_DLL_Queue(string nm); // constructor
    ~T_DLL_Queue(); // destructor
    T* dequeue(); // dequeue the first block
        // at the front of queue
    T* enqueue(T* pE); // enqueue the given
        // block at the end of queue
    string getName() { return name; }
    bool isEmpty() { return (dLList->empty()); }
    int size() { return dLList->getNumEntry(); }
    void fprint(ostream& fout,
         int elements_per_line);
private:
    T_DLL<T> *dLList;
    string name;
};
```

```
/* T_DLL_Queue.h  (2) */

template<typename T>
T_DLL_Queue<T>::T_DLL_Queue(string nm) //
constructor
{
    dLList = (T_DLL<T> *) new T_DLL<T>(nm);
    name = nm;
}

template<typename T>
T_DLL_Queue<T>::~T_DLL_Queue() // constructor
{

}
```

```
/* T_DLL_Queue.h  (3) */

template<typename T>
T* T_DLL_Queue<T>::enqueue(T* pE)
   // insert new element at the back of queue
{
    T* pEntry;
    pEntry = dLList->insertBack(pE);
    return pEntry;
}

template<typename T>
T* T_DLL_Queue<T>::dequeue()
    // pop the element at the front of the queue
{
    if (isEmpty())
    {
        return NULL;
    }
    else
    {
        T* pE;
        pE = dLList->front();
        dLList->removeFront();
        return pE;
    }
}
```

```
/* T_DLL_Queue.h  (3) */

template<typename T>
void T_DLL_Queue<T>::fprint(ostream& fout, int
    elements_per_line)
{
    int count = 0;
    if (isEmpty())
    {
        fout << this->getName()
            << " is Empty now !" << endl;
        return;
    }
    dLList->fprint(fout, elements_per_line);
}

#endif
```

main()

```
/** main.cpp (1) */
#include <iostream>
#include <fstream>
#include <iomanip>
#include <string>
#include "T_DLL.h"
#include "T_DLL_Queue.h"

using namespace std;
#define NUM_DATA 10

void main()
{
   ofstream fout;

   T_DLL_Queue<int> DLL_Queue_INT
     (string("DLL_Queue_INT"));
   int data[NUM_DATA] =
     { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
   string DLL_Queue_name;
   int d, *pD;
   int num_data = 5;
```

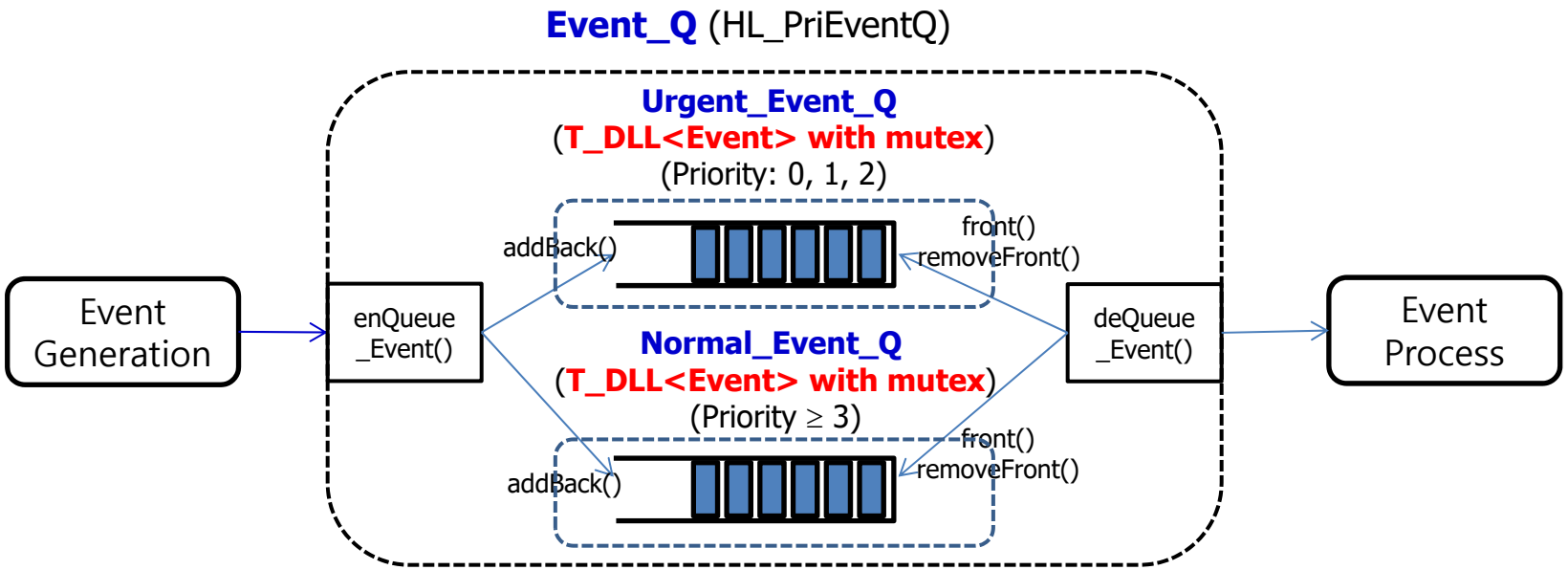
```
/** main.cpp (2) */

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

   DLL_Queue_name =
     DLL_Queue_INT.getName();
   fout << "\nTesting "
     << DLL_Queue_name << endl;
   for (int i = 0; i < NUM_DATA; i++)
   {
      d = data[i];
      DLL_Queue_INT.enqueue(&data[i]);
      fout << "After enqueue(" << setw(3)
        << d << ") : ";
      DLL_Queue_INT.fprint(fout, 10);
   }
   fout << endl;
```

```
/** main.cpp (3) */

    for (int i = 0; i < NUM_DATA; i++)
    {
        pD = DLL_Queue_INT.dequeue();
        fout << "After dequeue("
             << setw(3) << *pD << ") : ";
        /* fout << " ==> "
           << DLL_Queue_INT.size() << " entries in
           " << DLL_Stack_name << endl; */
        DLL_Queue_INT.fprint(fout, 10);
    }
    fout << endl;

    fout.close();

} // tail of main
```

```
Testing DLL_Queue_INT
After enqueue(  0) : 0
After enqueue(  1) : 0  1
After enqueue(  2) : 0  1  2
After enqueue(  3) : 0  1  2  3
After enqueue(  4) : 0  1  2  3  4
After enqueue(  5) : 0  1  2  3  4  5
After enqueue(  6) : 0  1  2  3  4  5  6
After enqueue(  7) : 0  1  2  3  4  5  6  7
After enqueue(  8) : 0  1  2  3  4  5  6  7  8
After enqueue(  9) : 0  1  2  3  4  5  6  7  8  9


After dequeue(  0) : 1  2  3  4  5  6  7  8  9
After dequeue(  1) : 2  3  4  5  6  7  8  9
After dequeue(  2) : 3  4  5  6  7  8  9
After dequeue(  3) : 4  5  6  7  8  9
After dequeue(  4) : 5  6  7  8  9
After dequeue(  5) : 6  7  8  9
After dequeue(  6) : 7  8  9
After dequeue(  7) : 8  9
After dequeue(  8) : 9
After dequeue(  9) : DLL_Queue_INT is Empty now !
```

Application of FIFO Queue – FIFO Event Processing with Priority Group

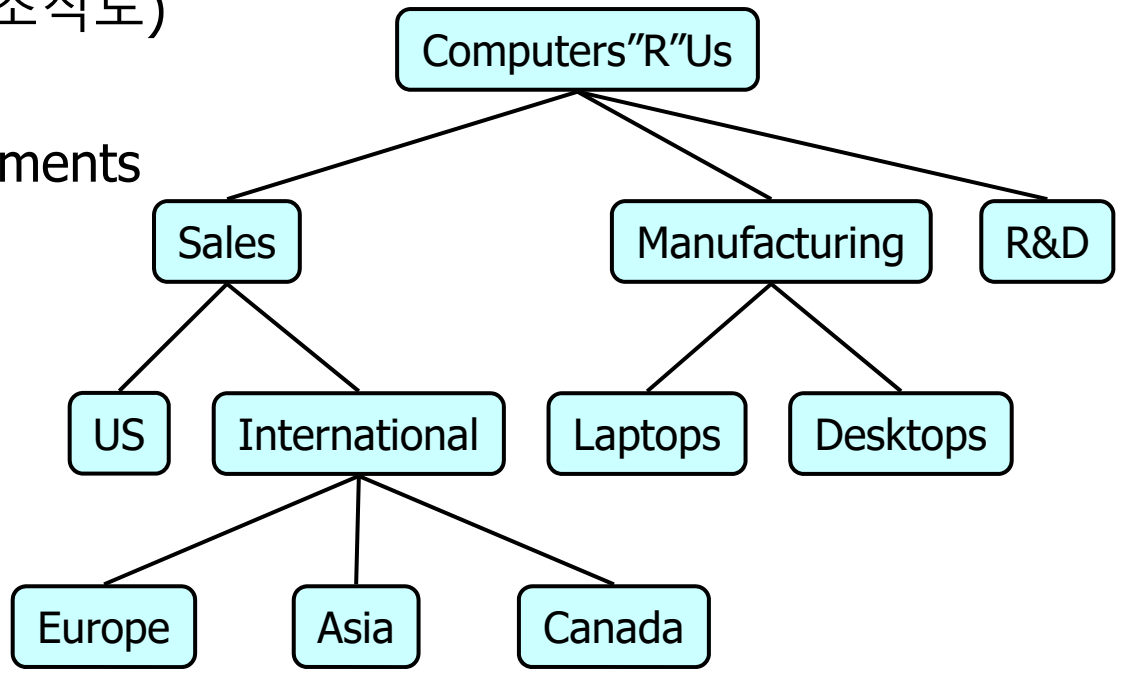
◆ Task_Q based on DoublyLinkedList (T_DLL<Event>)



이진탐색트리
(Binary Search Tree)

General Tree

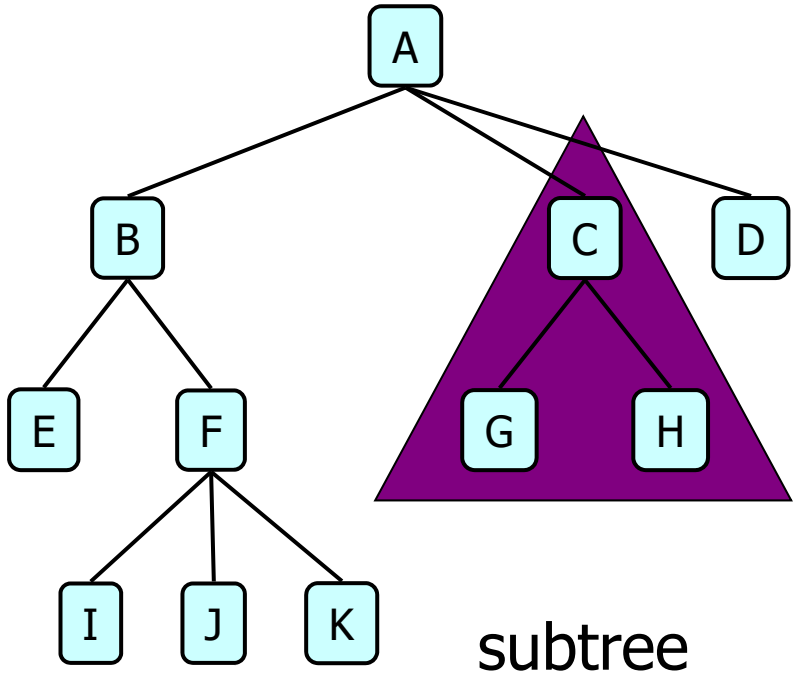
- **In computer science, a tree is an abstract model of a hierarchical structure**
- **A tree consists of nodes with a parent-child relation**
- **Applications:**
  - Organization charts (조직도)
  - File systems
  - Programming environments



트리자료구조의 용어

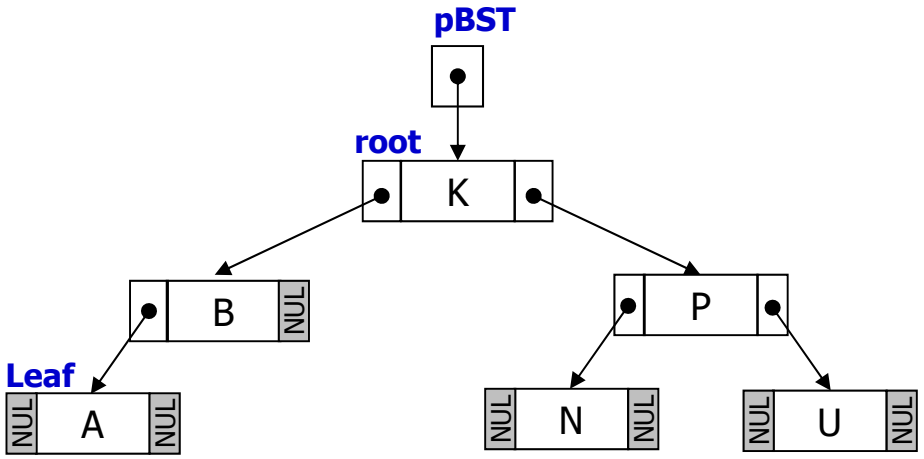
◆ 트리 자료구조의 용어

- Root: node without parent (A)
- Internal node: node with at least one child (A, B, C, F)
- External node (a.k.a. leaf): node without children (E, I, J, K, G, H, D)
- Ancestors of a node: parent, grandparent, grand-grandparent, etc.
- B is parent of child E and F
- E and F are children of B
- E and F are sibling
- Depth of a node: number of ancestors
- Height of a tree: maximum depth of any node (3)
- Descendant of a node: child, grandchild, grand-grandchild, etc.
- Subtree: tree consisting of a node and its descendants



이진 탐색 트리 (Binary Search Tree)

- **Binary Search Tree (이진 탐색 트리)**
  - 비 선형 자료 구조이며, 각 노드는 0, 1, 또는 2개의 다른 노드를 가리킴
  - 부모와 자식 노드간에 크기 순서가 유지됨
    (왼쪽자식노드값 < 부모노드 값 ≤ 오른쪽 자식노드값)
- **Parent(부모) 노드, Children (자식) 노드**
  - (부모) 노드 N이 가리키는 하위 노드들은 그 (부모) 노드 N의 children (자식) 노드들임
- **Root(루트) 노드**
  - 이진 트리의 최상위 부모노드
- **Leaf(리프) 노드**
  - 이진 트리의 최하단 노드로서 자식이 없는 트리노드



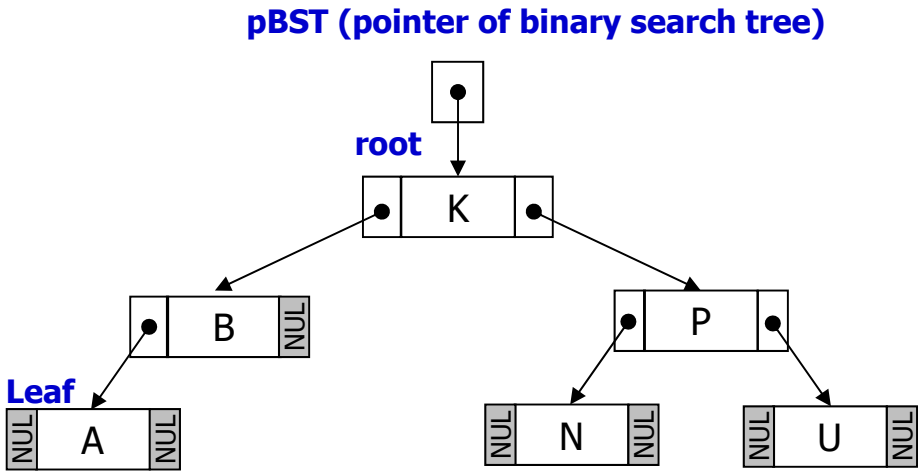
◆만약 노드 *N* 이 다른 노드 *P* 의 자식 (child) 노드이면,
노드 *P* 는 노드 *N* 의 부모 (parent) 노드임

◆자식이 하나도 없는 노드를 leaf 라고 함

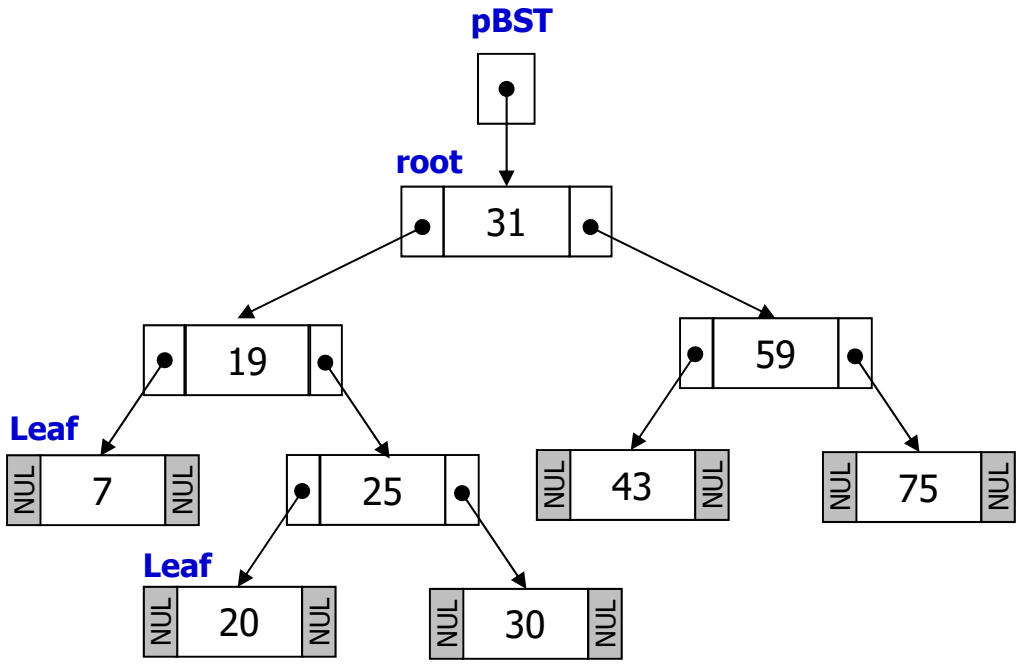
◆이진 트리에서는 부모 노드가 없는 단 하나의 특별한 노드가 있으며,
이를 그 트리의 *root*라고 함

◆트리 포인터 (tree pointer) pBST:
이진 트리의 첫번째 노드 (*root* 노드)를 가리킴

◆루트 노드 (root node):이진 트리에서 부모가 없는 최상위 노드



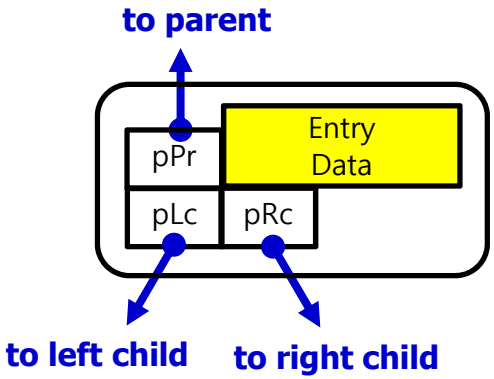
- 리프 노드(Leaf Node): 트리에서 자식이 하나도 없는 노드
- 아래 예에서, 데이터 7과 43을 가지는 노드는 leaf node
- 자식 노드: 데이터 31을 가지는 노드의 자식 노드들은 데이터 19와 59를 가지는 노드들임
- 데이터 43을 가지는 노드의 부모 (parent) 노드는 데이터 59을 가지는 노드임



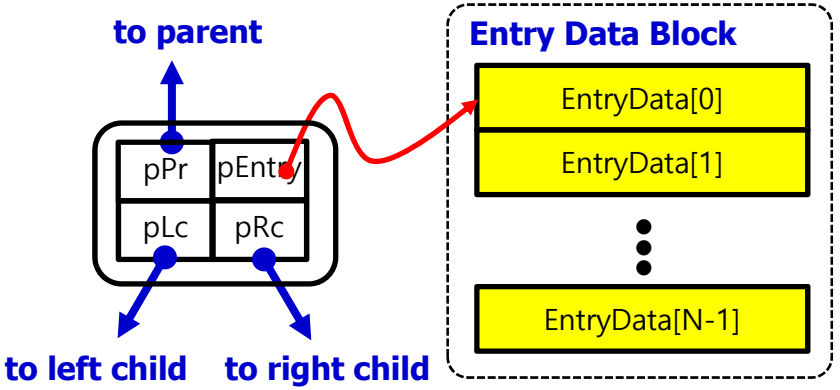
Binary Tree의 구현 (1)

◆ Binary Tree Node

- three pointers: to left child(pLc), to right child(pRc), to parent (pPr)
- data or pointer to data (pD)



(a) Binary Tree Node with Data



(b) Binary Tree Node with pointer to Entry Data

Binary Tree의 구현 (2)

◆ Structure BinarySearchTreeNode (BSTN)

- 연결 리스트의 노드와 유사하나, 2개의 자식을 가리키는 포인터 변수 (왼쪽 서브 트리 포인터, 오른쪽 서브 트리 포인터)와 부모 노드를 가리키는 포인터 를 가짐:

```
typedef struct BTN
{
    struct NodeData *pD; // to node data
    BTN *pLc;    //to left child
    BTN *pRc;    //to right child
    BTN *pPr;    //to parent
} BSTN;  // Binary Search Tree Node
```

Edge, Path, Ordered Tree

◆ Edges and paths

- **edge** (간선) of tree T is a pair of nodes (u, v) such that u is the parent of v, or vice versa
- **path** (경로) of T is a sequence of nodes such that any two consecutive nodes in the sequence form an edge

◆ Ordered Tree

- a **tree** is ordered if there is a linear ordering defined for the children of each node
- e.g.) structured document

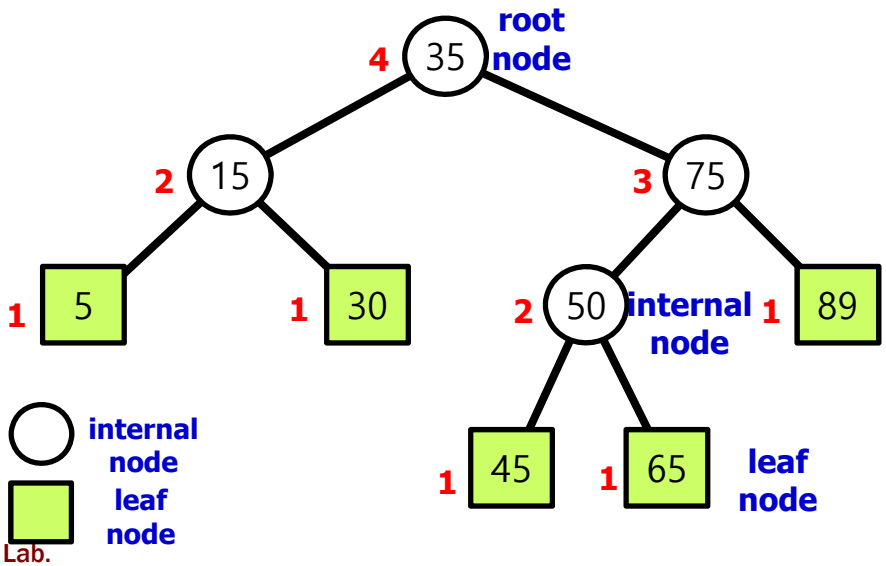
Internal vs. External, Height

◆ Internal vs. External

- internal node: binary tree node with at least one child
- external node: binary tree node without any child

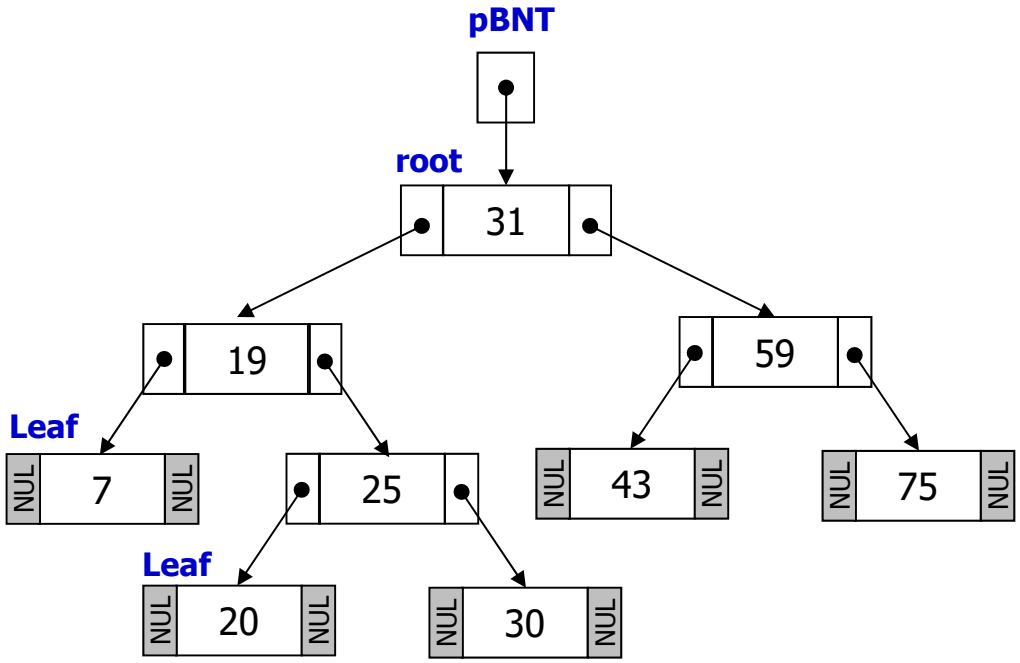
◆ Height

- height of leaf node = 1
- height =
  1 + max(getHeight(pBSTN->pLc), getHeight(pBSTN->pRc));



이진 탐색 트리 (Binary Search Tree)

- 이진 탐색 트리 (Binary search tree): 데이터 탐색을 쉽게 수행할 수 있도록 구성된 이진 트리
- 각 노드의 왼쪽 서브 트리는 그 노드의 값보다 작은 데이터를 가지는 노드들로만 구성
- 각 노드의 오른쪽 서브 트리는 그 노드의 데이터 값보다 같거나 큰 데이터 값을 가지는 노드들로만 구성



이진 탐색 트리 (Binary Search Tree)의 연산

- 이진 탐색 트리의 생성 (create)
- 이진 트리에 노드를 삽입 (insert): 이진 트리에서의 데이터 값들의 순서가 유지되도록 올바른 곳에 새로운 노드를 삽입
- 이진 트리에서의 노드 탐색 (search): 지정된 데이터 값을 가지는 노드를 탐색
- 이진 트리로 부터의 노드 삭제 (delete, erase): 지정된 노드를 삭제하고, 나머지 노드 들을 순서에 맞추어 재배열

T_Entry<K, V>

```
template<typename K, typename V>
class T_Entry
{
   friend ostream& operator<<(ostream& fout, T_Entry<K, V>& entry)
   {
         fout << "(" << entry.getKey() << ": " << *(entry.getValue()) << ")";
         return fout;
   }
public:
   T_Entry(K key, V value): _key(key), _value(value) {} // constructor
   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;
};
```

class T_BSTN<K, V>

```
/** Template Binary Search Tree Node. h */
#ifndef T_BSTN_H
#define T_BSTN_H
#include "T_Entry.h"
template<typename K, typename V>
class T_BSTN    //  binary search tree node
{
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
};
#endif
```

Class T_BST<K, V>

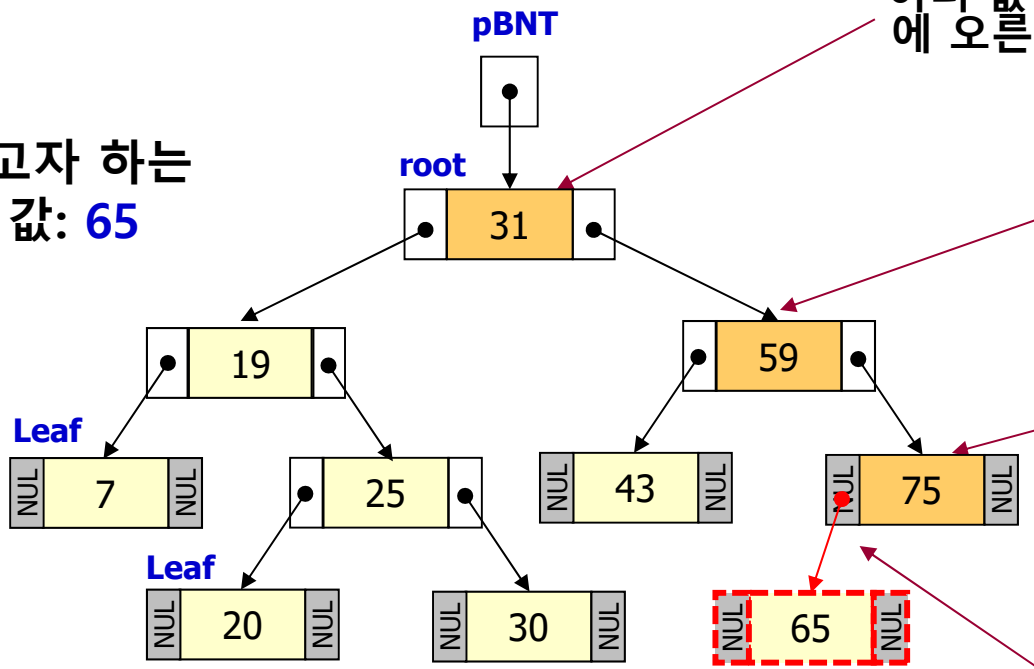
```
/** Template Binary Search Tree.h  (1) */
#ifndef T_BST_H
#define T_BST_H

#include <iostream>
#include <fstream>
#include "T_BSTN.h"
#include "T_Array.h"

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(); }
```

```
/** Template Binary Search Tree.h  (1) */

    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);
    V  searchBST(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);
```

```
/** Template Binary Search Tree.h  (1) */

protected:
    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
```

이진 탐색 트리 (Binary Search Tree)에 새로운 Entry (65 데이터 값)의 추가

입력하고자 하는 노드 값: 65



Step 1: 65가 root 노드의 데이터 값 (31) 보다 크기 때문에 오른쪽 서브 트리에 삽입해야 함.

Step 2: 65가 서브트리 root 노드의 데이터 값 (59) 보다 크기 때문에, 오른쪽 서브 트리에 삽입해야 함.

Step 3: 65가 서브트리 root 노드의 데이터 값 (75) 보다 작기 때문에, 왼쪽 서브 트리에 삽입해야 함.

Step 4: 왼쪽 서브 트리가 NULL 이므로 데이터 65의 이진 트리 노드를 이곳에 삽입

이진 탐색 트리 (Binary Search Tree)에서의 새로운 항목 삽입

◆ Binary Search Tree가 비어 있는 (empty) 경우

- 하나의 새로운 노드를 추가하여, **root** 노드로 만들고, 이 노드의 왼쪽 서브 트리와 오른쪽 서브 트리는 모두 비어 있는 상태로 구성

◆ Binary Search Tree가 비어 있지 않는 경우

- root 노드의 데이터 값과 새롭게 삽입되는 데이터를 비교함.
- 만약, 새로운 데이터가 root 노드 데이터 보다 작다면, 왼쪽 서브 트리에 추가하도록 하고,
- 새로운 데이터가 root 노드 데이터 보다 크다면, 오른쪽 서브 트리에 추가하도록 함.
- 각 서브 트리에서는 서브 트리 root와 비교하여 삽입될 위치를 결정하는 절차를 반복 수행하며, 정확한 위치를 결정한 후, 노드를 삽입함.

```
template<typename K, typename V>
T_BSTN<K, V>* T_BST<K, V>::_insertInOrder(T_BSTN<K, V>** pp,
      T_BSTN<K, V>* parenPos,  const T_Entry<K, V> entry)
{
    T_BSTN<K, V>* newPos, **pChildPos;
    T_BSTN<K, V>* pos;
    T_Entry<K, V> ent;

    if (pp == NULL)
    {
        cout << "Error in creation of BinarySearchTree :";
        cout << " address of the pointer to the Root Node is NULL !!₩n";
        exit;
    }
    pos = *pp;
    if (pos == NULL)
    {
        pos = new T_BSTN<K, V>(entry);
        if (parenPos == NULL)
        {
            _root = pos; // initialize the root node
        }
        pos->setpPr(parenPos);
        *pp = pos;
        num_entry++; // increment the number of elements
        return pos;
    }
```

```
    ent = pos->getEntry();
    if (entry < ent)
    {
        pChildPos = pos->getppLc();
        newPos = _insertInOrder(pChildPos, pos, entry);
        if (newPos != NULL)
            pos->setpLc(newPos);
        return NULL; // only the leaf child is set correctly, while the intermediate node is skipped
    }
    else if (entry >= ent)
    {
        pChildPos = pos->getppRc();
        newPos = _insertInOrder(pChildPos, pos, entry);
        if (newPos != NULL)
            pos->setpRc(newPos);
        return NULL; // only the leaf child is set correctly, while the intermediate node is skipped
    }
}

template<typename K, typename V>
void T_BST<K, V>::insertInOrder(const T_Entry<K, V> entry)
{
    _insertInOrder(&_root, NULL, entry);
}
```

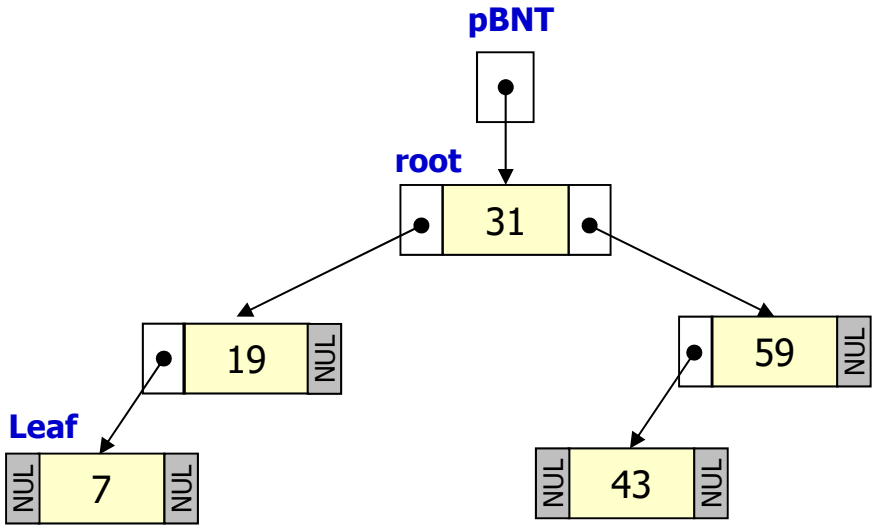
이진탐색트리의 순회, 탐색, 출력, 트리노드 삭제

이진 탐색 트리의 순회 (Traversal)

◆ 세가지 종류의 순회 방식:

- **Inorder**: 데이터의 크기에 따른 순차 순회
  - 서브루트 노드의 왼쪽 서브 트리를 먼저 순회
  - 서브루트 노드의 데이터 처리
  - 서브루트 노드의 오른쪽 서브 트리를 순회
- **Preorder**: 각 서브 트리의 서브 루트 노드 데이터를 먼저 처리
  - 서브루트 노드의 데이터 처리
  - 서브루트 노드의 왼쪽 서브 트리를 순회
  - 서브루트 노드의 오른쪽 서브 트리를 순회
- **Postorder**: 각 서브 트리의 서브 루트 노드 데이터를 나중에 처리
  - 서브루트 노드의 왼쪽 서브 트리를 순회
  - 서브루트 노드의 오른쪽 서브 트리를 순회
  - 서브루트 노드의 데이터 처리

이진 탐색 트리의 순회 (Traversal)



순회 방식	노드 방문 순서
Inorder	**7, 19, 31, 43, 59**
Preorder	**31, 19, 7, 59, 43**
Postorder	**7, 19, 43, 59, 31**

```
template<typename K, typename V>
void T_BST<K, V>::
   traversal_inOrder(T_BSTN<K, V>* pos,
   T_Array<V>& array_value)
{
    T_BSTN<K, V> *pLc, *pRc;
    T_Entry<K, V> entry;
    V value;
    if (pos == NULL)
        return;
    pLc = pos->getpLc();
    pRc = pos->getpRc();
    traversal_inOrder(pLc, array_value);
    entry = pos->getEntry();
    value = entry.getValue();
    array_value.insertBack(value);
    traversal_inOrder(pRc, array_value);
}

template<typename K, typename V>
void T_BST<K, V>::
   traversal_preOrder(T_BSTN<K, V>* pos,
   T_Array<V>& array_value)
{
    T_BSTN<K, V> *pLc, *pRc;
    T_Entry<K, V> entry;
    V value;
    if (pos == NULL)
        return;
    pLc = pos->getpLc();
    pRc = pos->getpRc();

    entry = pos->getEntry();
    value = entry.getValue();
    array_value.insertBack(value);

    traversal_preOrder(pLc, array_value);
    traversal_preOrder(pRc, array_value);
}

template<typename K, typename V>
void T_BST<K, V>::
   traversal_postOrder(T_BSTN<K, V>* pos,
   T_Array<V>& array_value)
{
    T_BSTN<K, V> *pLc, *pRc;
    T_Entry<K, V> entry;
    V value;
    if (pos == NULL)
        return;
    pLc = pos->getpLc();
    pRc = pos->getpRc();
    traversal_postOrder(pLc, array_value);
    traversal_postOrder(pRc, array_value);

    entry = pos->getEntry();
    value = entry.getValue();
    array_value.insertBack(value);
}
```

이진 탐색 트리에서의 탐색 (Searching)

1: Root 노드 부터 시작

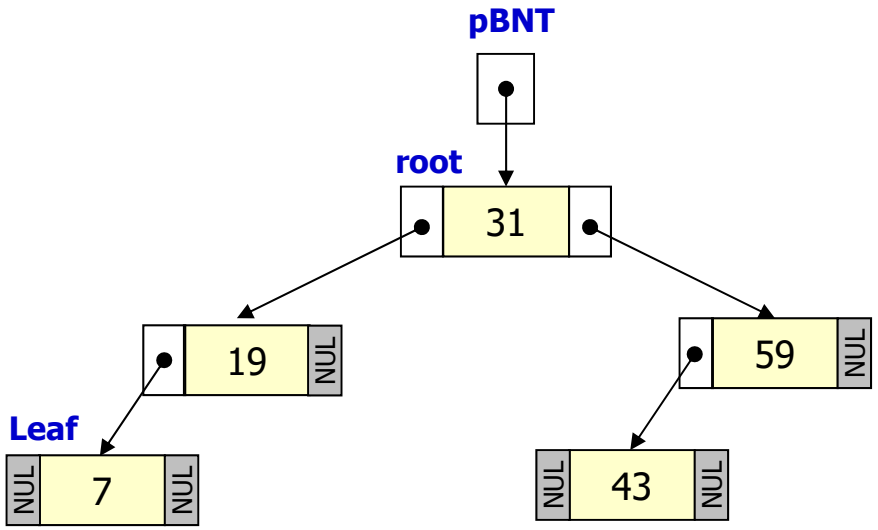
2: 노드 데이터를 검사 :

2.1: 만약 찾고자 하는 데이터라면 탐색 성공/종료

2.2: 만약 탐색 데이터가 노드 데이터 보다 작다면, 왼쪽 서브 트리에서 step 2를 수행

2.3: 만약 탐색 데이터가 노드 데이터 보다 크다면, 오른쪽 서브 트리에서 step 2를 수행

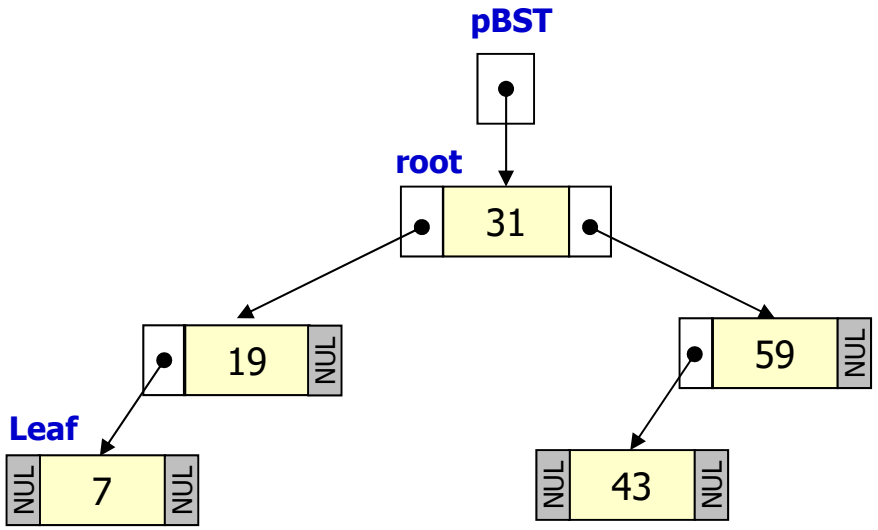
3: 탐색 데이터를 찾을 때 까지 계속 탐색하거나, NULL 포인터에 도달 할 때 까지 수행



이진 탐색 트리에서의 탐색 (Searching)

데이터 43을 가지는 노드를 탐색:

- Root 노드의 데이터 (31)과 비교
- 43이 31보다 크기 때문에 오른쪽 서브 트리의 루트 (데이터 59)와 비교
- 43이 59보다 작기 대문에 왼쪽 서브 트리의 루트 노드 (데이터 43)과 비교
- 그 노드의 데이터 값이 43이므로 탐색 성공. 탐색 종료



```
template<typename K, typename V>
T_BSTN<K, V>* T_BST<K,
V>::_searchBSTN(T_BSTN<K, V>* pos, K k)
{
      K ent_k;
      T_BSTN<K, V>* pos_result = NULL;

      if (pos == NULL)
          return NULL;

      ent_k = pos->getKey();
      if (ent_k == k)
          pos_result = pos;
            // given entry was found here !!
      else if (ent_k > k)
          pos_result =
            _searchBSTN(pos->getpLc(), k)
      else if (ent_k < k)
          pos_result =
              _searchBSTN(pos->getpRc(), k)

      return pos_result;
}
```

```
template<typename K, typename V>
T_BSTN<K, V>* T_BST<K, V>::searchBSTN(K key)
{
      T_BSTN<K, V>* pEntry;
      pEntry = _searchBSTN(_root, key);

      return pEntry;
}

template<typename K, typename V>
V  T_BST<K, V>::searchBST(K key)
{
    T_BSTN<K, V>* pEntry;
    V value;

    pEntry = _searchBSTN(_root, key);
    value = pEntry->getValue();
    return value;
}
```

Find Min, Max

```
template<typename K, typename V>
T_BSTN<K, V>* T_BST<K, V>::
    _minBSTN(T_BSTN<K, V>* subRoot)
{
        T_BSTN<K, V> *pos, *pLc;

        if ((subRoot == NULL) ||
            (NULL == subRoot->getpLc()))
                return subRoot;
        pos = subRoot;
        while ((pos->getpLc()) != NULL)
                pos = pos->getpLc();
        return pos;
}

template<typename K, typename V>
T_BSTN<K, V>* T_BST<K, V>::
    _maxBSTN(T_BSTN<K, V>* subRoot)
{
        T_BSTN<K, V> *pos, *pLc;

        if ((subRoot == NULL) ||
            (NULL == subRoot->getpRc()))
                return subRoot;
        pos = subRoot;
        while ((pos->getpRc()) != NULL)
                pos = pos->getpRc();
        return pos;
}
```

```
template<typename K, typename V>
T_Entry<K, V>& T_BST<K, V>::minEntry()
{
        T_BSTN<K, V>* pMin;

        pMin = _minBSTN(_root);
        return pMin->getEntry();
}

template<typename K, typename V>
T_Entry<K, V>& T_BST<K, V>::maxEntry()
{
        T_BSTN<K, V>* pMax;

        pMax = _maxBSTN(_root);
        return pMax->getEntry();
}
```

fprint_inOrder()

```
template<typename K, typename V>
void T_BST<K, V>::_fprint_inOrder(T_BSTN<K, V>* pTN, ostream& fout)
{
    T_BSTN<K, V> *pRc, *pLc;
    if ((pLc = pTN->getpLc()) != NULL)
        _fprint_inOrder(pLc, fout);
    fout << pTN->getEntry() << endl;
    if ((pRc = pTN->getpRc()) != NULL)
        _fprint_inOrder(pRc, fout);
}

template<typename K, typename V>
void T_BST<K, V>::fprint_inOrder(ostream& fout)
{
    T_BSTN<K, V>* root = getRoot();
    if (num_entry == 0)
    {
        fout << getName() << " is empty now !!" << endl;
        return;
    }
    _fprint_inOrder(root, fout);
}
```

fprint_with_Depth()

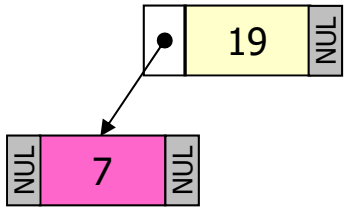
```
template<typename K, typename V>
void T_BST<K, V>::_fprint_with_Depth(T_BSTN<K, V>* pTN, ostream& fout, int depth)
{
    T_BSTN<K, V> *pRc, *pLc;
    T_Entry<K, V>* pEntry;

    if ((pRc = pTN->getpRc()) != NULL)
        _fprint_with_Depth(pRc, fout, depth + 1);
    for (int i = 0; i < depth; i++)
    {
        fout << "    ";
    }
    fout << pTN->getEntry() << endl;
    if ((pLc = pTN->getpLc()) != NULL)
        _fprint_with_Depth(pLc, fout, depth + 1);
}

template<typename K, typename V>
void T_BST<K, V>::fprint_with_Depth(ostream& fout)
{
    T_BSTN<K, V>* root = getRoot();
    if (num_entry == 0)
    {
        fout << getName() << " is empty now !!" << endl;
        return;
    }
    _fprint_with_Depth(root, fout, 0);
}
```

```
BBST_BK_keyAuthor : current size (15)
            (Zulu    : [Book_13 , Zulu     , (2018.1 .1 ) ])
        (Yankee  : [Book_12 , Yankee   , (1977.1 .1 ) ])
    (Turk    : [Book_11 , Turk     , (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 ) ])
            (Dail    : [Book_15 , Dail     , (2000.1 .1 ) ])
    (Choi    : [Book_06 , Choi     , (2003.1 .1 ) ])
            (Chalie  : [Book_10 , Chalie   , (1970.1 .1 ) ])
        (Brown   : [Book_08 , Brown    , (2012.1 .1 ) ])
            (Abraham : [Book_09 , Abraham , (1980.1 .1 ) ])
                (Abc     : [Book_14 , Abc      , (2015.1 .1 ) ])
```

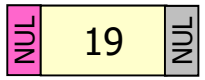
Leaf Node 삭제 (Erase)

◆ 삭제하고자 하는 노드가 리프 노드 (leaf node)일때

- 삭제 대상 리프 노드의 부모 노드에서 그 (삭제 대상) 리프 노드를 가리키는 포인터 값을 NULL로 설정하고, 그 리프 노드를 삭제

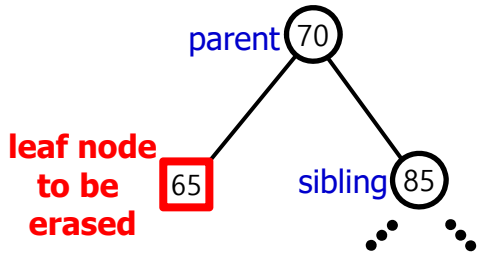


(a) before erasing node with data 7

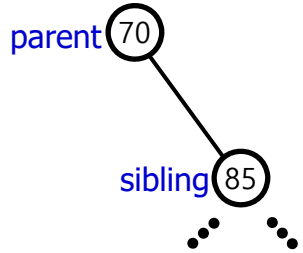


(b) after erasing node with data 7

◆ 삭제 대상 노드의 sibling 이 있는 경우



(a) before erase of a leaf node

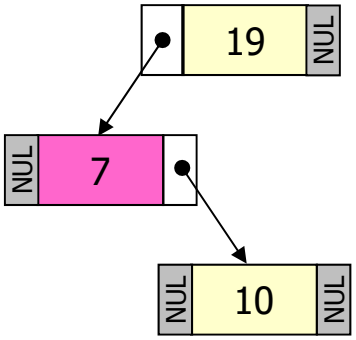


(b) after erase of a leaf node

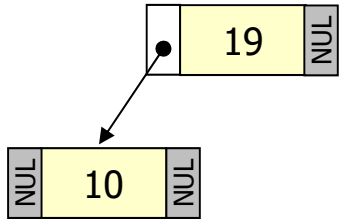
하나의 자식 노드를 가진 중간 노드 삭제 (Erase)

◆ 삭제 대상 노드가 하나의 자식 노드를 가지고 있을 때

- 삭제 대상 노드의 부모 노드 포인터가 삭제 대상 노드의 자식 노드를 직접 가리키도록 조정한 후, 그 삭제 대상 노드를 삭제



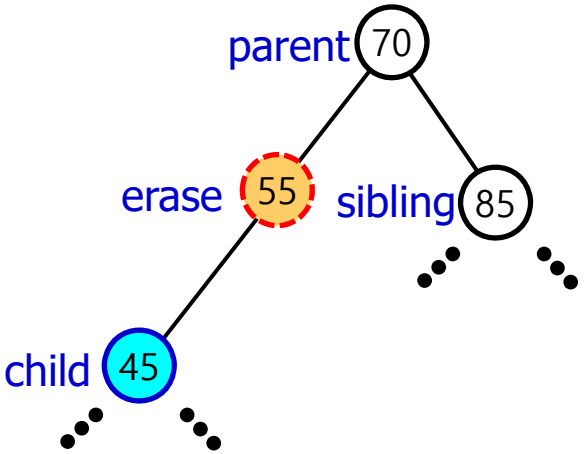
(a) before erasing node with data 7



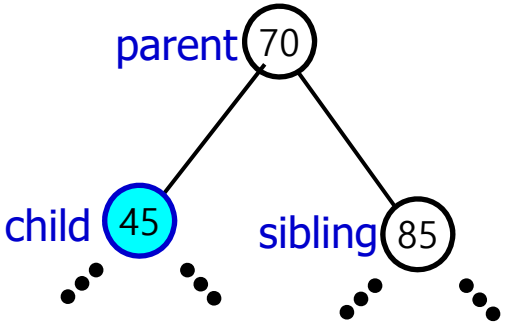
(b) after erasing node with data 7

Erase of a Tree Node with Single Child

◆ Erase of a node with single child



(a) before erase of a node with single child



(b) after erase of a node with single child

두개의 자식노드를 가진 트리 노드 삭제 (Erase)

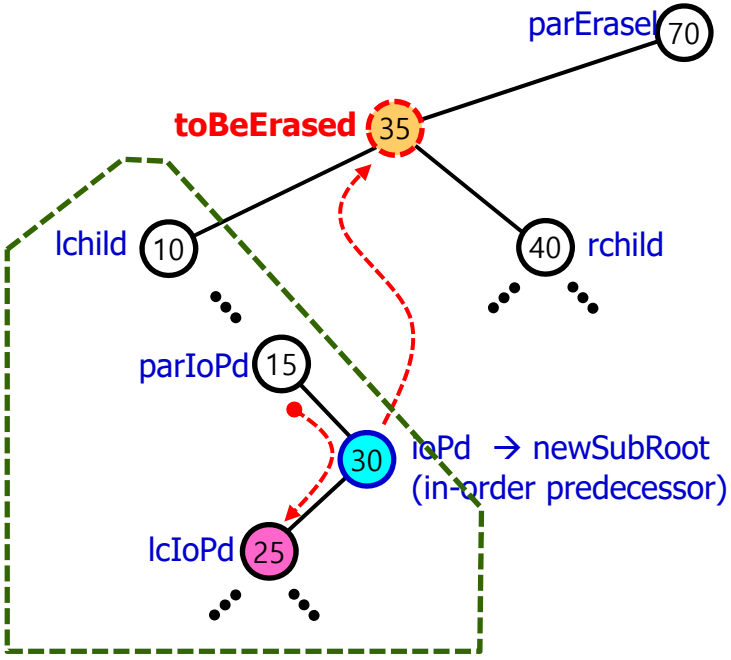
◆ 이진탐색트리의 균형을 고려하는 경우

- 삭제 대상 트리노드의 두 자식이 포함된 서브트리의 높이를 고려하여, 더 높이가 큰 서브트리로 부터 위치 조정 대상 트리노드를 선정
- 만약 왼쪽 서브트리의 높이가 더 높은 경우, 삭제 대상 노드의 값보다 작으면서 그 서브트리의 노드 들 중 값이 가장 큰 노드를 선택 (in-order predecessor)
- 만약 오른쪽 서브 트리의 높이가 더 높은 경우, 삭제 대상 노드의 값보다 크면서 그 서브트리의 노드 들 중 값이 가장 작은 노드를 선택 (in-order successor)

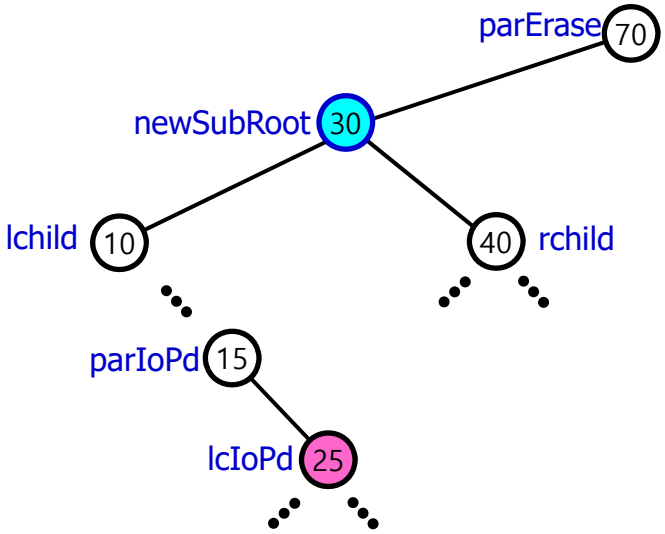
Balance를 고려한 eraseBSTN() (1)

◆ 왼쪽 서브 트리의 height가 더 높은 경우

- 왼쪽 서브 트리에서 가장 큰 값을 가진 in-order predecessor를 찾아 삭제 대상 노드 위치에 배치



(a) Before erase of a tree node
(delete 35)

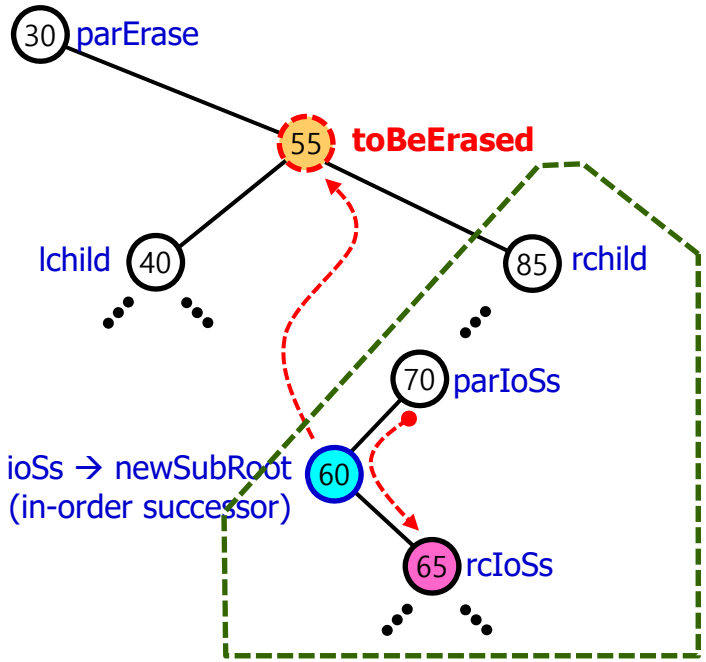


(b) After erase and substitution with
in-order predecessor

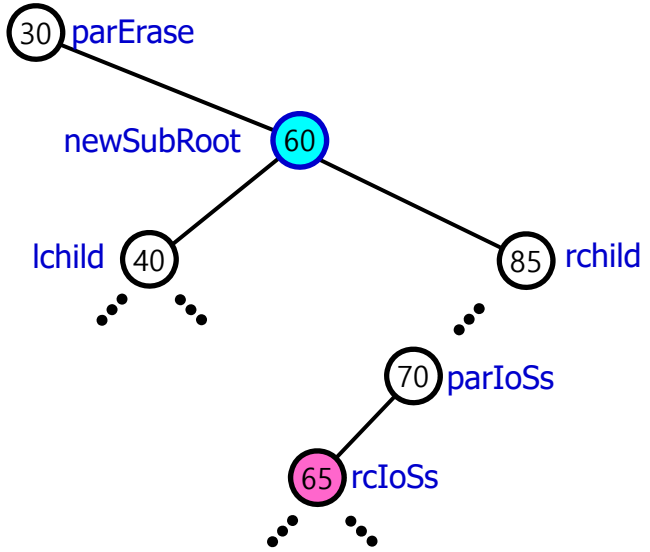
Balance를 고려한 eraseBSTN() (2)

◆ 오른쪽 서브 트리의 height가 더 높은 경우

- 오른쪽 서브 트리에서 가장 작은 값을 가진 in-order successor를 찾아 삭제 대상 노드 위치에 배치



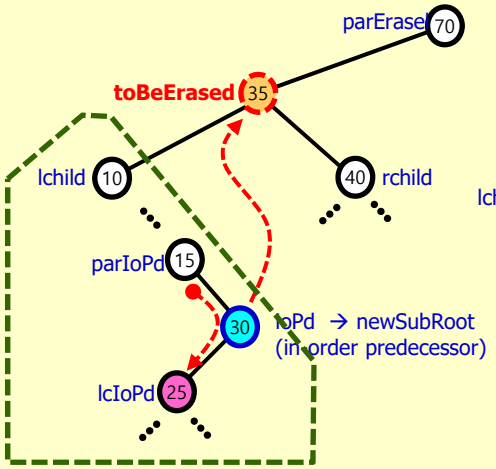
(c) Before erase of a tree node
(delete 55)



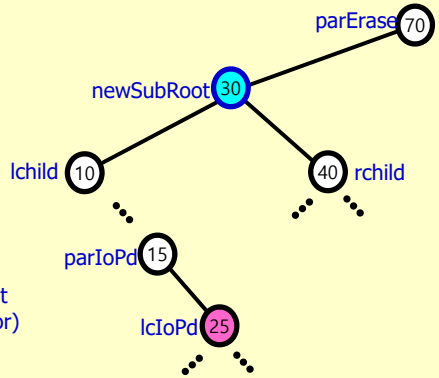
(d) After erase and substitution with
in-order successor

```
template<typename K, typename V>
T_BSTN<K, V>* T_BST<K, V>::eraseBSTN(T_BSTN<K, V>** pptoBeErased)
    // remove BSTN considering balance of the BST
{
    T_BSTN<K, V> *newSubRoot, *temp, *w, *wlc;
    T_BSTN<K, V>* toBeErased;

    toBeErased = *pptoBeErased;
    if (toBeErased == NULL)
    return NULL;
    if ((toBeErased->getpLc() == NULL) && (toBeErased->getpRc() == NULL))
        // no child
    {
        newSubRoot = NULL;
    }
    else if ((toBeErased->getpLc() != NULL) && (toBeErased->getpRc() == NULL))
        // only left child
    {
        newSubRoot = toBeErased->getpLc();
        newSubRoot->setpPr(toBeErased->getpPr());
    }
    else if ((toBeErased->getpLc() == NULL) && (toBeErased->getpRc() != NULL))
        // only right child
    {
        newSubRoot = toBeErased->getpRc();
        newSubRoot->setpPr(toBeErased->getpPr());
    }
```

```
else
{    /* tree node to be deleted has both left child and right child */
     int heightDiff = _getHeightDiff(toBeErased);
     T_BSTN<K, V> *parDel = toBeErased->getpPr();
     T_BSTN<K, V> *lChild = toBeErased->getpLc();
     T_BSTN<K, V> *rChild = toBeErased->getpRc();
     T_BSTN<K, V> *ioSs = NULL, *rcIoSs, *parIoSs;
     T_BSTN<K, V> *ioPd = NULL, *lcIoPd, *parIoPd;

     if (heightDiff > 0)
     // left subtree is higher, so put the ioPd in the place of the erased node
     {
          ioPd = _maxBSTN(lChild); // in-order predecessor (ioPd)
          lcIoPd = ioPd->getpLc();
          parIoPd = ioPd->getpPr();
          newSubRoot = ioPd;
          if (ioPd->getpPr() != toBeErased)
          {
              newSubRoot ->setpLc(lChild);
              parIoPd->setpRc(lcIoPd);
              if (lcIoPd != NULL)
                  lcIoPd->setpPr(parIoPd);
          }
          newSubRoot ->setpRc(rChild);
          newSubRoot ->setpPr(toBeErased->getpPr());
      }
```

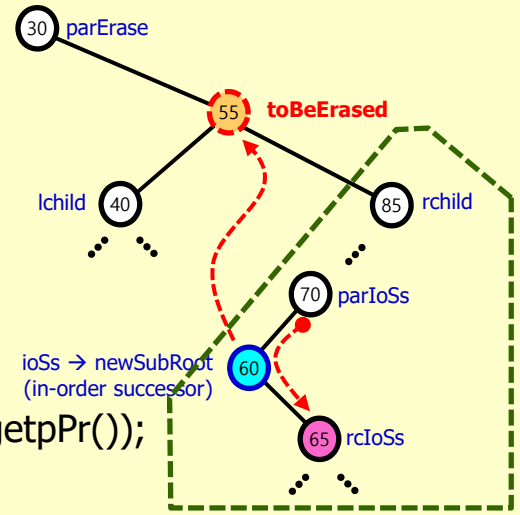


(a) Before erase of a tree node (delete 35)

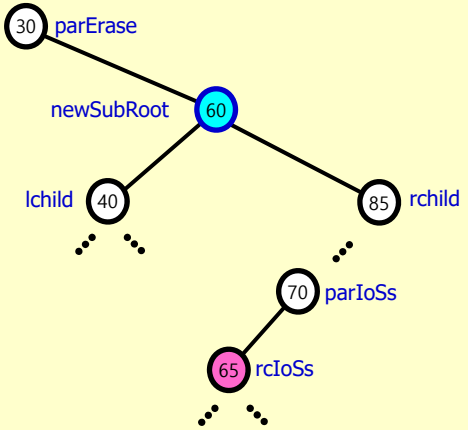


(b) After erase and substitution with in-order predecessor

```
        else
        // right subtree is higher, so put the ioSs in the place of the erased node
        {
                ioSs = _minBSTN(rChild); // in-order successor (ioSs)
                rcIoSs = ioSs->getpRc();
                parIoSs = ioSs->getpPr();
                newSubRoot = ioSs;
                if (ioSs->getpPr() != toBeErased)
                {
                    newSubRoot ->setpRc(rChild);
                    parIoSs->setpLc(rcIoSs);
                    if (rcIoSs != NULL)
                       rcIoSs->setpPr(parIoSs);
                }
                newSubRoot ->setpLc(lChild);
                newSubRoot ->setpPr(toBeErased->getpPr());
        }

        if (lChild != ioPd)
              lChild->setpPr(newSubRoot);
         if (rChild != ioSs)
              rChild->setpPr(newSubRoot);
    }
    if (toBeErased == _root)
        _root = newSubRoot;
    num_entry--; // decrement the number of entries in the BST
    free(toBeErased);
    *pptoBeErased = newSubRoot;
    return newSubRoot;
}
```



(c) Before erase of a tree node (delete 55)



(d) After erase and substitution with in-order successor

이진탐색트리의 응용

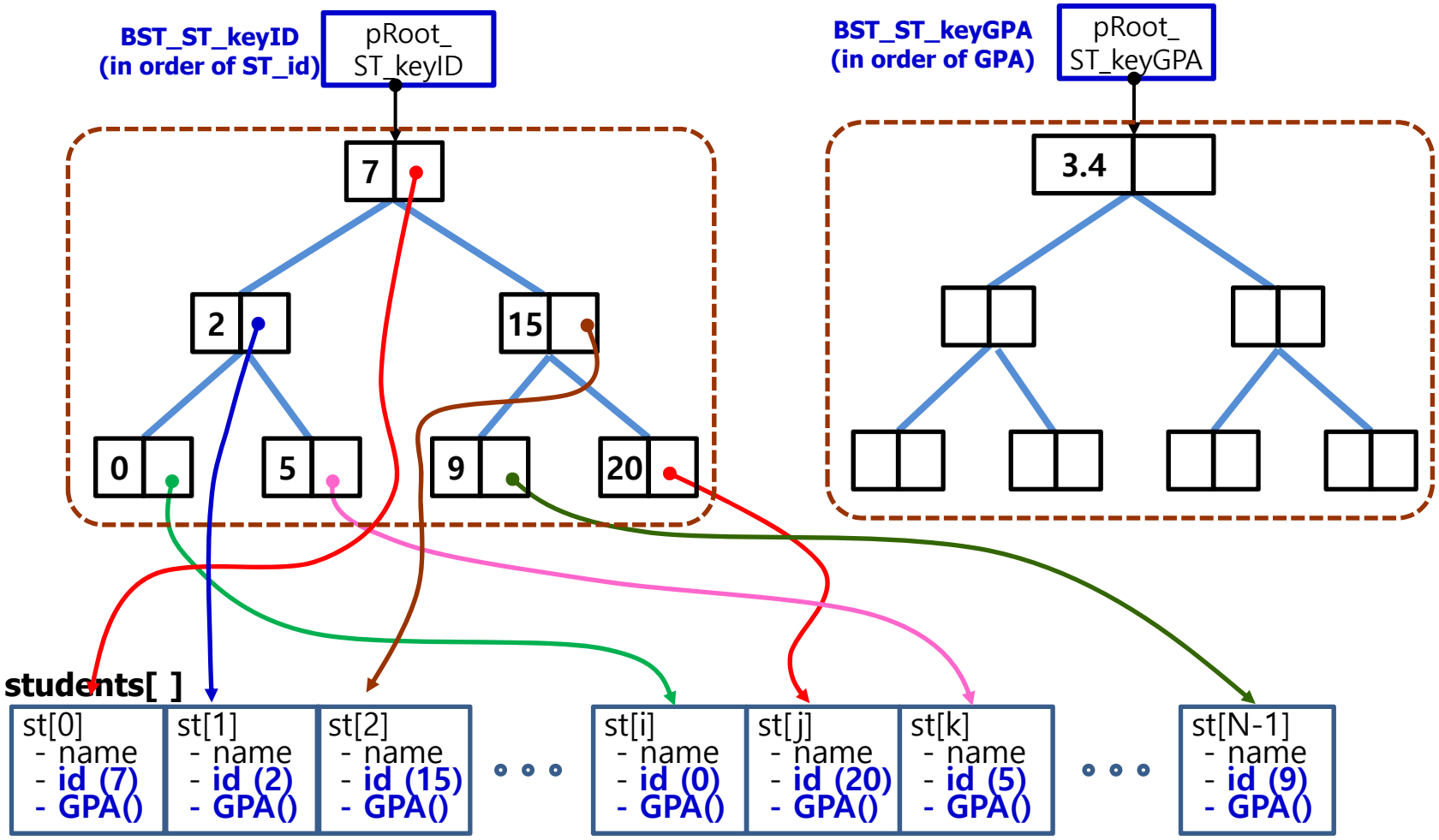
Binary Tree기반의 학생 정보 관리

◆ class Student

```
class Student
{
      friend ostream & operator<< (ostream &, const Student &);
public:
      Student(int id, string n, double gpa);
      int getID() const { return st_id; }
      double getGPA() const { return gpa; }
      string getName() const { return name; }
      void setST_id(int id) { st_id = id; }
      void setGPA(double g) { gpa = g; }
      const Student& operator=(const Student& right);
      bool operator>(const Student& right);
      bool operator==(const Student& right);
private:
      int st_id;
      string name;
      double gpa;
};
```

Binary Search Tree for School with Students

◆ Total configuration



```
/** main.cpp  (1) */
#include <iostream>
#include <fstream>
#include <iomanip>
#include <string>
#include "T_BST.h"
#include "T_Array.h"
#include "T_Entry.h"
#include "Student.h"

using namespace std;
#define NUM_STUDENTS 15

void main()
{
	Student students[NUM_STUDENTS] =
	{
		Student(10, string("Kim, G-M"), 4.5),
		Student(11, string("Lee, S-M"), 4.4),
		Student(12, string("Park, S-S"), 4.3),
		Student(13, string("Lee, K-M"), 4.2),
		Student(14, string("Hong, G-M"), 4.1),
		Student(15, string("Jang, S-M"), 4.0),
		Student(16, string("Hwang, S-T"), 3.9),
		Student(17, string("Choi, Y-H"), 3.8),
		Student(18, string("Shin, D-J"), 3.7),
		Student(19, string("Kwak, S-B"), 3.6),
		Student(20, string("Kang, S-M"), 3.5),
		Student(21, string("Jong, S-T"), 3.4),
		Student(22, string("Lee, Y-H"), 3.3),
		Student(23, string("Sung, D-J"), 3.2),
		Student(24, string("Kong, S-B"), 3.1)
	};
```

```
/** main.cpp  (2) */

      ofstream fout("output.txt");
      if (fout.fail())
      {
            cout << "Fail to create output.txt for results !!" << endl;
            exit;
      }
      T_Entry<int, Student*> entry_ID_pST;
      T_BST<int, Student*> BST_ST("BST_Student");
      T_BSTN<int, Student*> **ppBST_ST_root;

      fout << "Input student[] array : " << endl;
      for (int i = 0; i < NUM_STUDENTS; i++)
      {
             fout << setw(3) << students[i] << endl;
      }
      fout << endl;

      fout << endl << "Testing Binary Search Tree without Rebalancing" << endl;
      ppBST_ST_root = BST_ST.getRootAddr();
      for (int i = 0; i < NUM_STUDENTS; i++)
      {
             entry_ID_pST.setKey(students[i].getID());
             entry_ID_pST.setValue(&students[i]);
             //fout << "Insert inOrder (" << entry_ID_pST << ") into " << bst_name << endl;
             BST_ST.insertInOrder(entry_ID_pST);
      }
      fout << "₩nElements in " << BST_ST.getName() << " (in order) : " << endl;
      //BST_ST.fprint_inOrder(fout);
      BST_ST.fprint_with_Depth(fout);
```

```
/** main.cpp  (3) */

      fout << endl << "Testing BST_ST_keyGPA without Rebalancing" << endl;
      T_Entry<double, Student*> entry_GPA_pST;
      T_BST<double, Student*> BST_ST_keyGPA("BST_Student_keyGPA");
      T_BSTN<double, Student*> *pRoot_ST_keyGPA, **ppRoot_BST_ST_keyGPA;
      ppBST_ST_root = BST_ST.getRootAddr();
      for (int i = 0; i < NUM_STUDENTS; i++)
      {
          entry_GPA_pST.setKey(students[i].getGPA());
          entry_GPA_pST.setValue(&students[i]);
          //fout << "Insert inOrder (" << setw(3) << data[i] << ") into " << bst_name << endl;
          BST_ST_keyGPA.insertInOrder(entry_GPA_pST);
      }
      fout << "₩nElements in " << BST_ST_keyGPA.getName() << " (in order of GPA) : " << endl;
      //BST_ST.fprint_inOrder(fout);
      BST_ST_keyGPA.fprint_with_Depth(fout);

      fout.close();
}
```

```
Input student[] array :
Student[ 10,   Kim, G-M,  4.50]
Student[ 11,   Lee, S-M,  4.40]
Student[ 12,  Park, S-S,  4.40]
Student[ 13,   Lee, K-M,  4.40]
Student[ 14,  Hong, G-M,  4.10]
Student[ 15,  Jang, S-M,  4.00]
Student[ 16, Hwang, S-T,  3.90]
Student[ 17,  Choi, Y-H,  3.80]
Student[ 18,  Shin, D-J,  3.70]
Student[ 19,  Kwak, S-B,  3.60]
Student[ 20,  Kang, S-M,  3.50]
Student[ 21,  Jong, S-T,  3.40]
Student[ 22,   Lee, Y-H,  3.30]
Student[ 23,  Sung, D-J,  3.20]
Student[ 24,  Kong, S-B,  3.10]


Testing Binary Search Tree without Rebalancing

Elements in BST_Student (in order of ID) :
BST_Student : current size (15)
                                                    (24: Student[ 24,  Kong, S-B,  3.10] )
                                                (23: Student[ 23,  Sung, D-J,  3.20] )
                                            (22: Student[ 22,   Lee, Y-H,  3.30] )
                                        (21: Student[ 21,  Jong, S-T,  3.40] )
                                    (20: Student[ 20,  Kang, S-M,  3.50] )
                                (19: Student[ 19,  Kwak, S-B,  3.60] )
                            (18: Student[ 18,  Shin, D-J,  3.70] )
                        (17: Student[ 17,  Choi, Y-H,  3.80] )
                    (16: Student[ 16, Hwang, S-T,  3.90] )
                (15: Student[ 15,  Jang, S-M,  4.00] )
            (14: Student[ 14,  Hong, G-M,  4.10] )
        (13: Student[ 13,   Lee, K-M,  4.40] )
    (12: Student[ 12,  Park, S-S,  4.40] )
(11: Student[ 11,   Lee, S-M,  4.40] )
(10: Student[ 10,   Kim, G-M,  4.50] )
```

```
Elements in BST_Student_keyGPA (in order of GPA) :
BST_Student_keyGPA : current size (15)
(4.50: Student[ 10,   Kim, G-M,  4.50] )
            (4.40: Student[ 13,   Lee, K-M,  4.40] )
        (4.40: Student[ 12,  Park, S-S,  4.40] )
    (4.40: Student[ 11,   Lee, S-M,  4.40] )
        (4.10: Student[ 14,  Hong, G-M,  4.10] )
            (4.00: Student[ 15,  Jang, S-M,  4.00] )
                (3.90: Student[ 16, Hwang, S-T,  3.90] )
                    (3.80: Student[ 17,  Choi, Y-H,  3.80] )
                        (3.70: Student[ 18,  Shin, D-J,  3.70] )
                            (3.60: Student[ 19,  Kwak, S-B,  3.60] )
                                (3.50: Student[ 20,  Kang, S-M,  3.50] )
                                    (3.40: Student[ 21,  Jong, S-T,  3.40] )
                                        (3.30: Student[ 22,   Lee, Y-H,  3.30] )
                                            (3.20: Student[ 23,  Sung, D-J,  3.20] )
                                                (3.10: Student[ 24,  Kong, S-B,  3.10] )
```

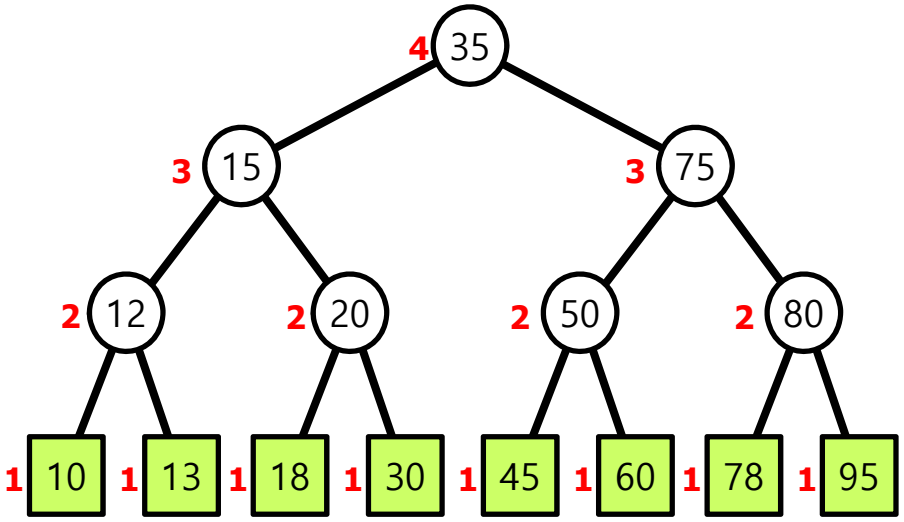
Binary Tree의 문제점

◆ Binary Tree의 편중/편향

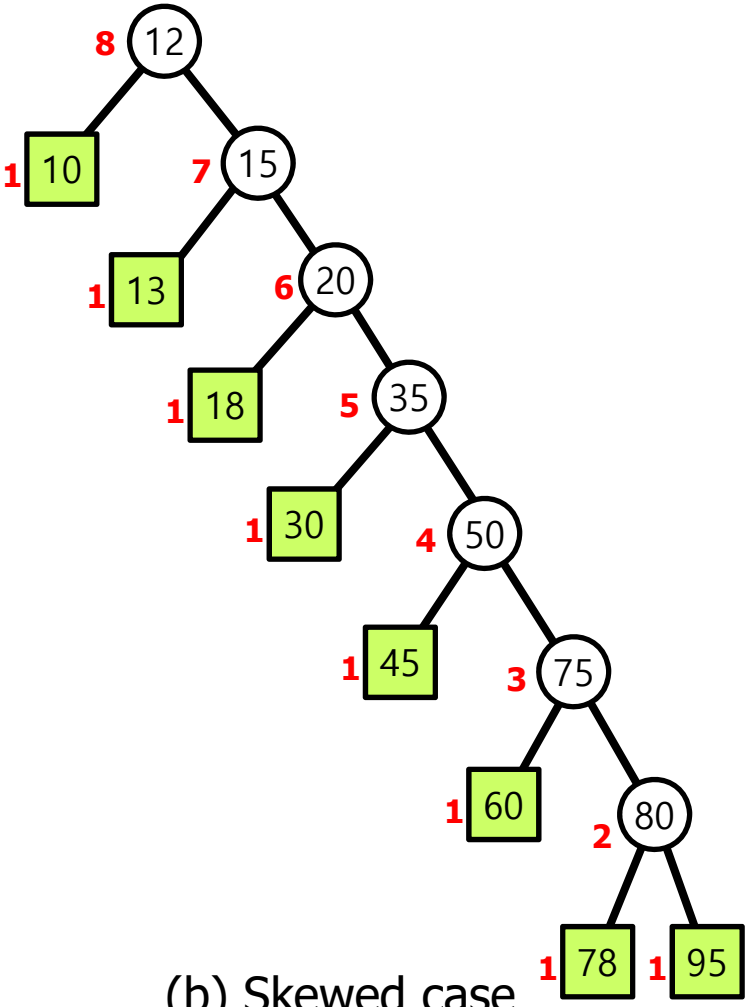
- Binary Tree의 좌우 균형이 잘 잡혀있는 경우, 탐색 (search) 시간은 $O(\log_2 N)$
- Root 노드의 데이터 값이 전체 데이터의 중간 값을 가지지 못하는 경우, 이진 트리가 한쪽으로 편중/편향될 수 있으며, 이 경우 binary search tree 기반의 search 성능이 저하됨
- 최악의 경우로 root 노드가 전체 데이터 값 중 최소값 또는 최대값을 가지는 경우, linked list와 같은 형태가 발생할 수 도 있음: 탐색 시간이 평균 N/2임 (i.e., O(N))
- Binary Tree에 새로운 데이터 노드를 추가할 때, 전체 binary tree의 균형을 잡을 수 있도록 재 조정하여야 함

이진탐색트리의 균형화
(Balancing of Binary Search Tree)

Balancing of Binary Search Tree



(a) Well-balanced case

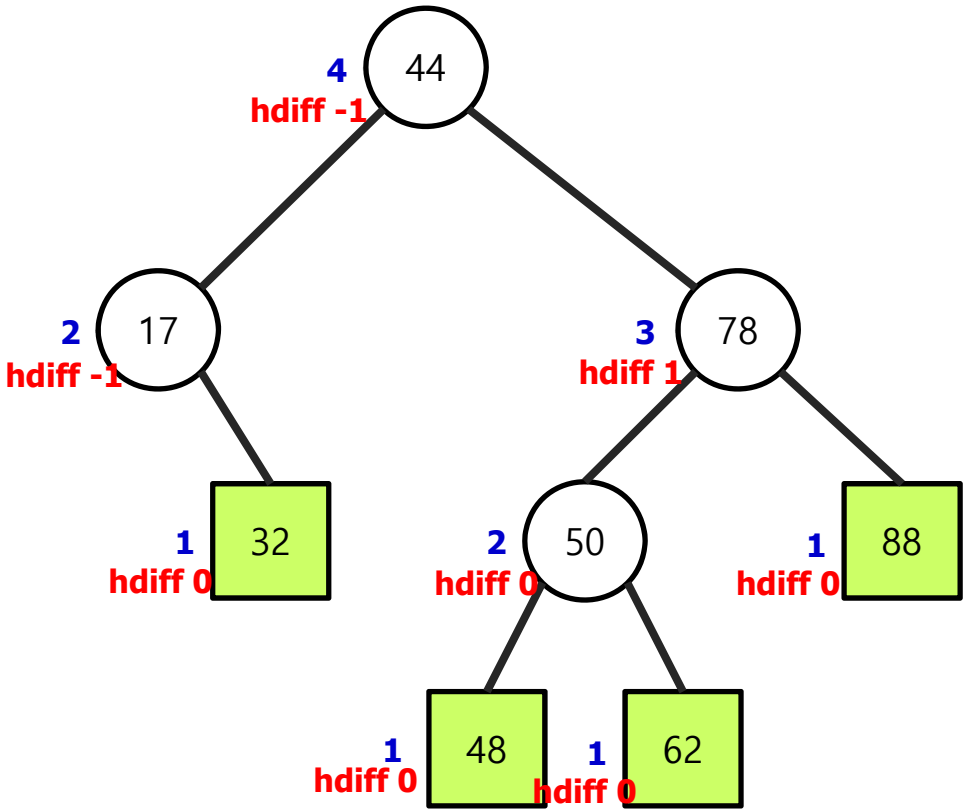


(b) Skewed case
(without re-balancing)

균형 이진 탐색 트리 - AVL Tree

◆AVL Tree

- Proposed by Adelson, Velskii, Landis in 1962
- AVL trees are balanced
- An AVL Tree is a binary search tree such that for every internal node v of T, the heights of the children of v can differ by at most 1



An example of an AVL tree
(with **height and height difference**)

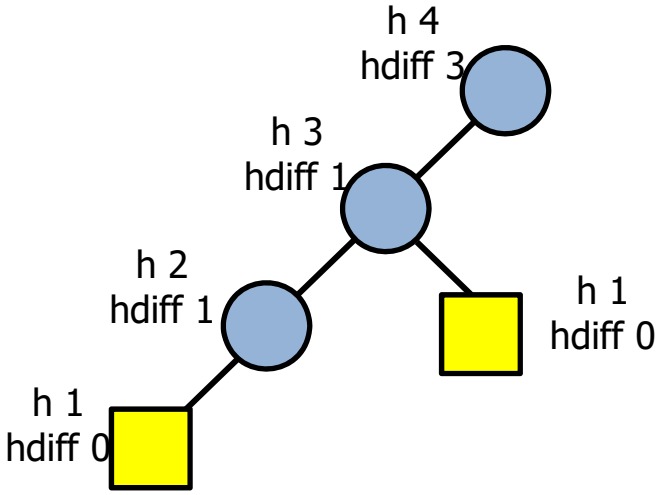
◆ getHeight(), getHeightDiff()

```
template<typename K, typename V>
int T_BST<K, V>::_getHeight(T_BSTN<K, V>* pTN)
{
    int height = 0;
    int height_Lc, height_Rc;

    if (pTN != NULL)
    {
        height_Lc = _getHeight(pTN->getpLc());
        height_Rc = _getHeight(pTN->getpRc());
        if (height_Lc > height_Rc)
            height = 1 + height_Lc;
        else
            height = 1 + height_Rc;
    }
    return height;
}
```

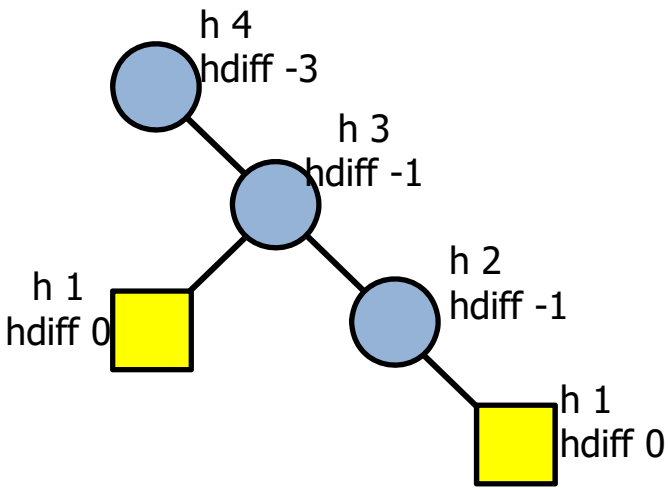
```
template<typename K, typename V>
int T_BST<K, V>::
    _getHeightDiff(T_BSTN<K, V>* pTN)
{
    int heightDiff = 0;

    if (pTN == NULL)
        return 0;
    heightDiff = _getHeight(pTN->getpLc())
                - _getHeight(pTN->getpRc());

    return heightDiff;
}
```

Tri-node Restructuring

- **height**
  - height of leaf node : 1
  - height of inner node: 1 + MAX (height of left child, height of right child)
- **height difference**
  - height difference = height of left child – height of right child
- **tri-node restructuring**
  - re-arrange three node so that the re-arranged subtree has height difference less than or equal to 1



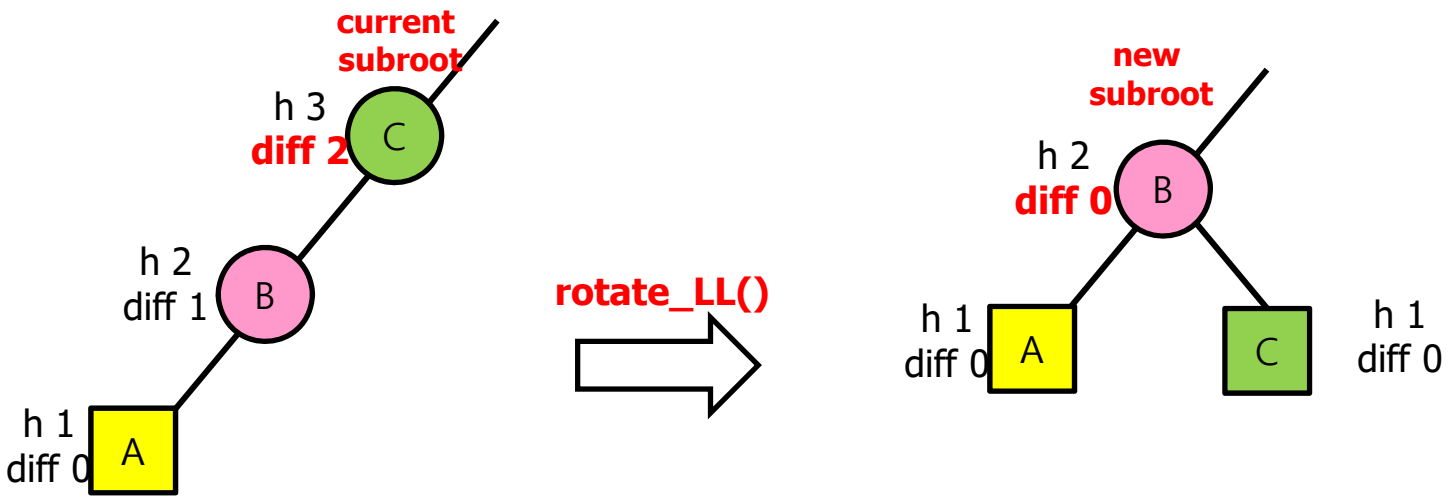
(a) Left - Skewed Binary Tree



(b) Right - Skewed Binary Tree

Tri-node Restructuring – rotate_LL

◆ Single rotation LL of subtree

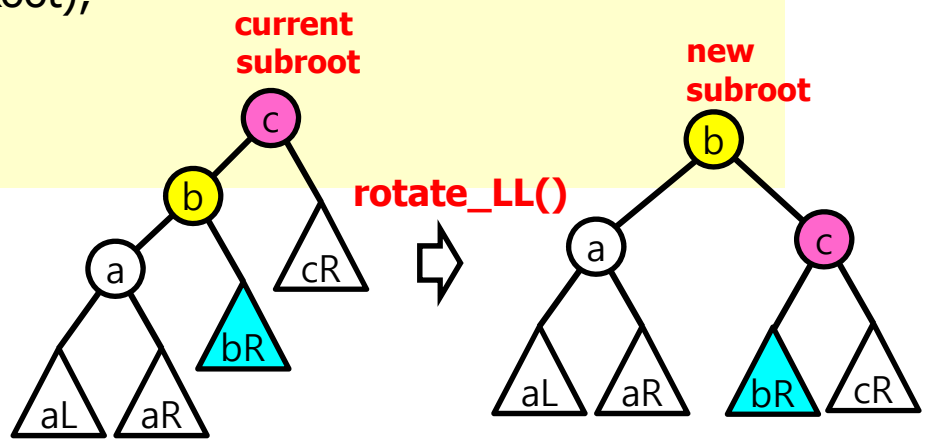


(a) Left-left skewed

(b) After rotate_LL()

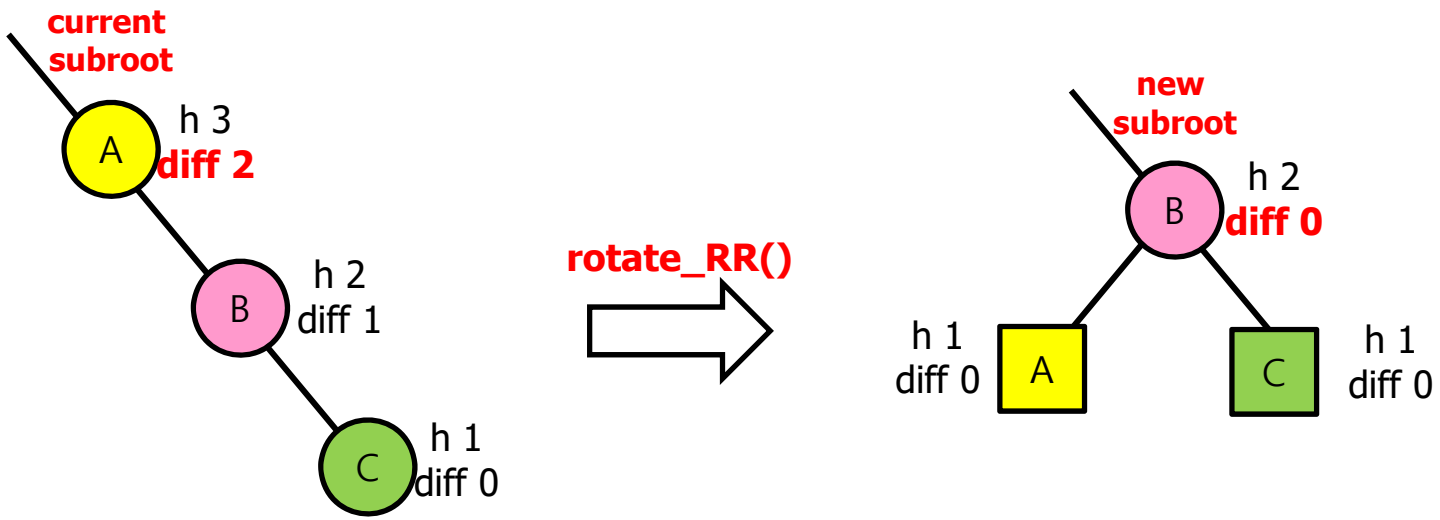
◆ rotate_LL()

```
template<typename K, typename V>
T_BSTN<K, V>* T_BST<K, V>::_rotate_LL(T_BSTN<K, V> *pCurSubRoot)
{
    T_BSTN<K, V> *pNewSubRoot, *pBR, *pCurParent;

    pCurParent = pCurSubRoot->getpPr();
    pNewSubRoot = pCurSubRoot->getpLc();
    pBR = pNewSubRoot->getpRc();
    pCurSubRoot->setpLc(pBR);
    if (pBR != NULL)
        pBR->setpPr(pCurSubRoot);
    pNewSubRoot->setpRc(pCurSubRoot);
    pNewSubRoot->setpPr(pCurParent);
    pCurSubRoot->setpPr(pNewSubRoot);

    return pNewSubRoot;
}
```



Tri-node Restructuring – rotate_RR (1)

◆ Single rotation RR of subtree

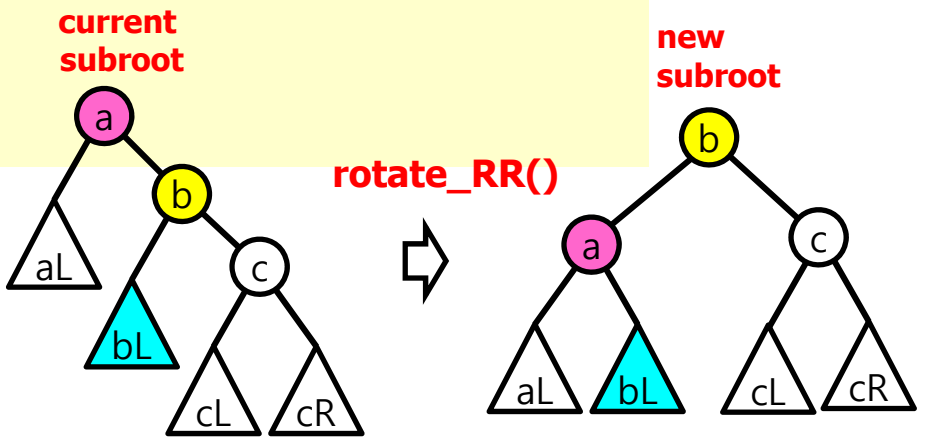


(a) Right-right skewed

(b) After rotate_RR()

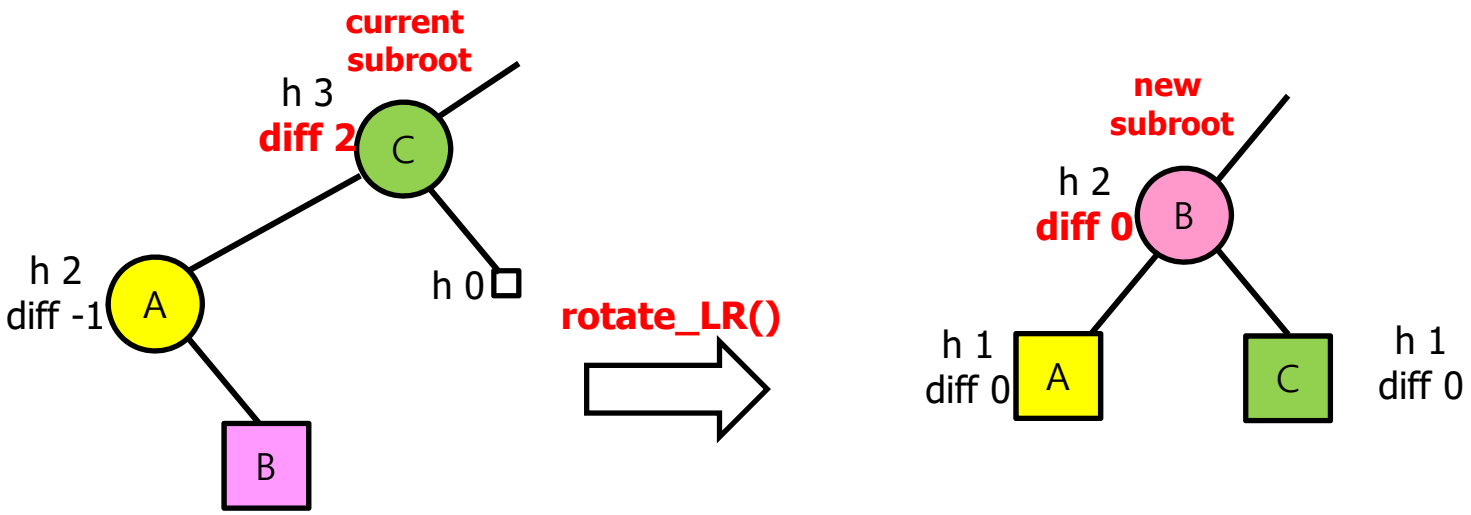
◆ rotate_RR(),

```
template<typename K, typename V>
T_BSTN<K, V>* T_BST<K, V>::_rotate_RR(T_BSTN<K, V> *pCurSubRoot)
{
    T_BSTN<K, V> *pNewSubRoot, *pBL, *pCurParent;

    pCurParent = pCurSubRoot->getpPr();
    pNewSubRoot = pCurSubRoot->getpRc();
    pBL = pNewSubRoot->getpLc();
    pCurSubRoot->setpRc(pBL);
    if (pBL != NULL)
        pBL->setpPr(pCurSubRoot);
    pNewSubRoot->setpLc(pCurSubRoot);
    pNewSubRoot->setpPr(pCurParent);
    pCurSubRoot->setpPr(pNewSubRoot);

    return pNewSubRoot;
}
```



Tri-node Restructuring – rotate_LR (1)

◆ Double rotation LR of subtree

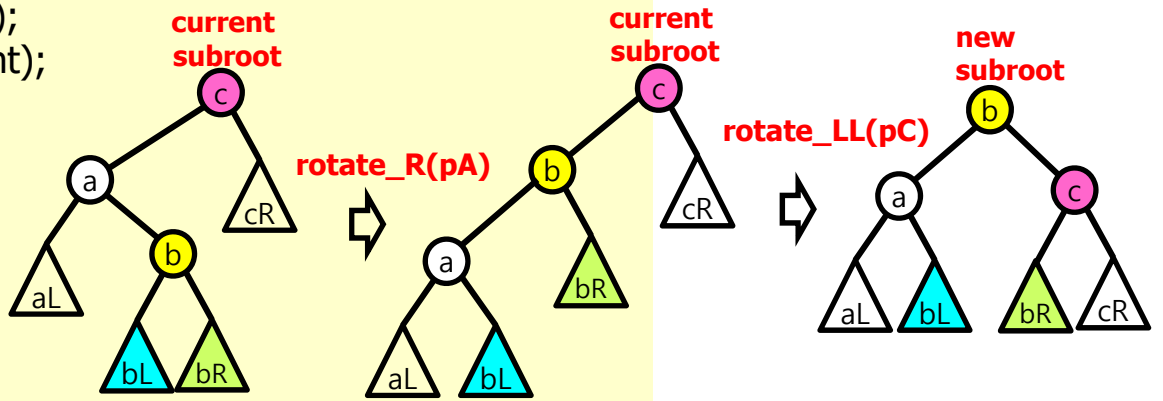


(a) Left-right skewed

(b) After rotate_LR()

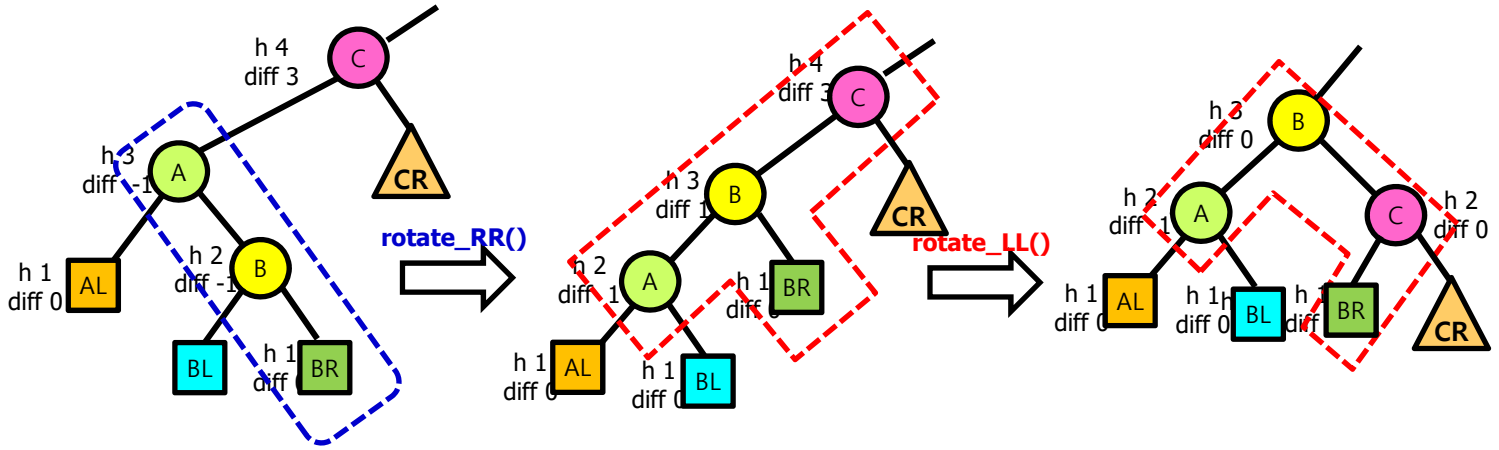
◆ Algorithm rotate_LR()

```
template<typename K, typename V>
T_BSTN<K, V>* T_BST<K, V>::_rotate_LR(T_BSTN<K, V> *pCurSubRoot)
{
    T_BSTN<K, V> *pSubRoot, *pNewSubRoot, *pCurParent;
    T_BSTN<K, V> *pA, *pB, *pC, *pBL, *pBR;

    pC = pCurSubRoot;
    pCurParent = pCurSubRoot->getpPr();
    pA = pC->getpLc();
    pB = pA->getpRc();
    pBL = pB->getpLc();
    pBR = pB->getpRc();
    pSubRoot = _rotate_RR(pA);
    pCurSubRoot->setpLc(pSubRoot);
    pNewSubRoot = _rotate_LL(pC);
    pNewSubRoot->setpPr(pCurParent);
    pA->setpPr(pNewSubRoot);
    pC->setpPr(pNewSubRoot);
    if (pBL != NULL)
        pBL->setpPr(pA);
    if (pBR != NULL)
        pBR->setpPr(pC);

    return pNewSubRoot;
}
```



(a) LR- skewed (b) after rotate_RR(pA) (c) after rotate_LR()

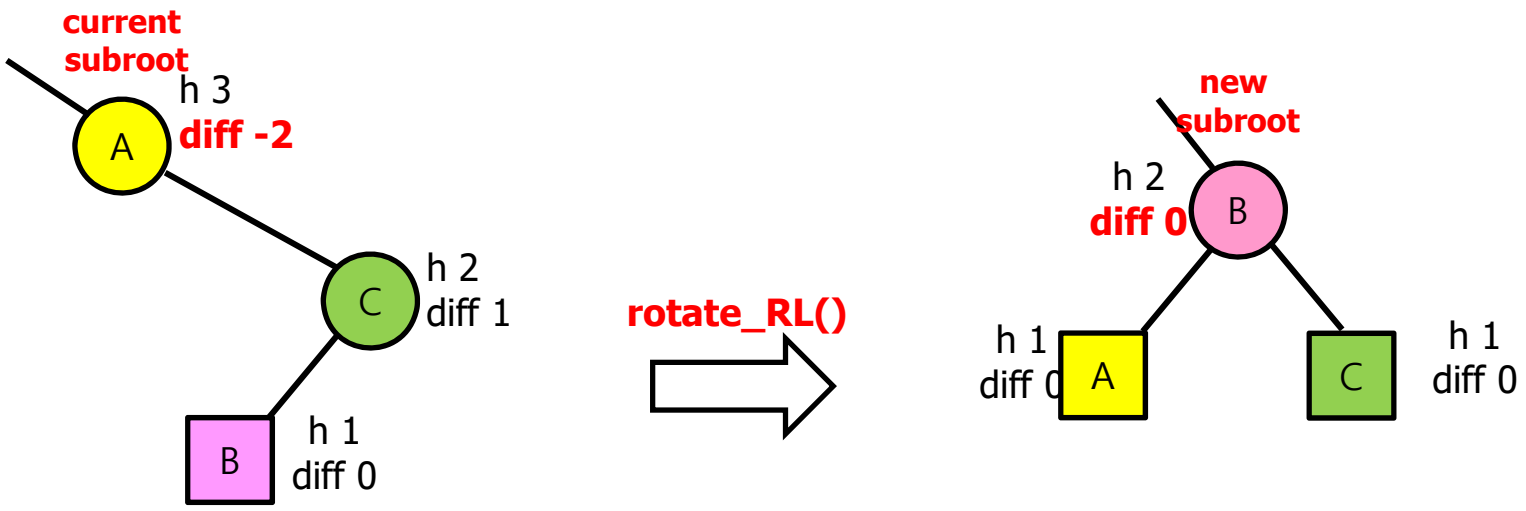
◆ Detailed Operations in rotateLR()



(a) Initial Status (b) after rotate_RR() (c) after rotate_LL()

Tri-node Restructuring – rotate_RL (1)

◆ Double rotation RL of subtree

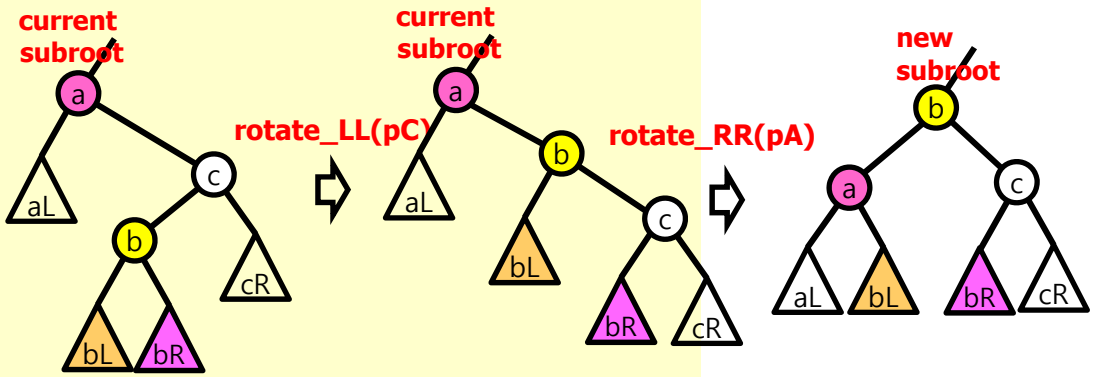


(a) Right-Left skewed

(b) After rotate_RL()

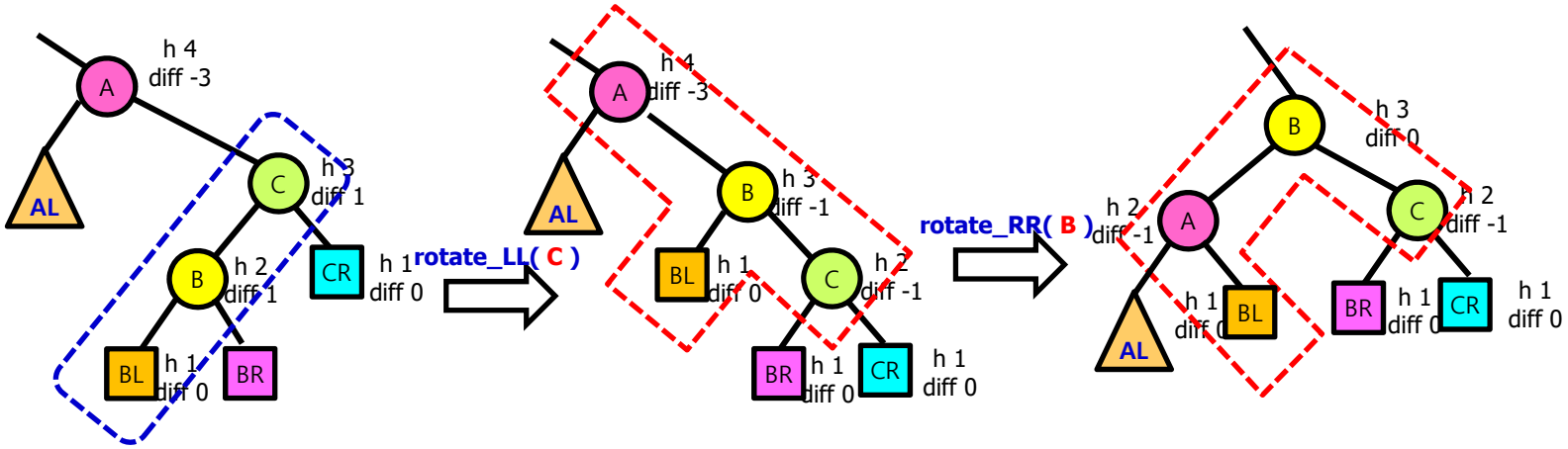
◆ rotate_RL()

```
template<typename K, typename V>
T_BSTN<K, V>* T_BST<K, V>::_rotate_RL(T_BSTN<K, V> *pCurSubRoot)
{
    T_BSTN<K, V> *pSubRoot, *pNewSubRoot, *pCurParent;
    T_BSTN<K, V> *pA, *pB, *pC, *pBL, *pBR;

    pA = pCurSubRoot;
    pCurParent = pCurSubRoot->getpPr();
    pC = pA->getpRc();
    pB = pC->getpLc();
    pBL = pB->getpLc();
    pBR = pB->getpRc();
    pSubRoot = _rotate_LL(pC);
    pCurSubRoot->setpRc(pSubRoot);
    pNewSubRoot = _rotate_RR(pA);

    pNewSubRoot->setpPr(pCurParent);
    pA->setpPr(pNewSubRoot);
    pC->setpPr(pNewSubRoot);
    if (pBL != NULL)
        pBL->setpPr(pA);
    if (pBR != NULL)
        pBR->setpPr(pC);

    return pNewSubRoot;

}
```



(a) Right-Left skewed (b) Right-Left skewed (c) after rotate_RL()

◆ Detailed Operations in rotateRL()



(a) Initial status (b) After rotate_LL (c) After rotate_RR

_reBalance()

```
template<typename K, typename V>
T_BSTN<K, V>* T_BST<K, V>::_reBalance(T_BSTN<K, V>** ppTN)
{
	int heightDiff = 0;

	heightDiff = _getHeightDiff(*ppTN);
	if (heightDiff > 1) // left subtree is higher
	{
		if (_getHeightDiff((*ppTN)->getpLc()) > 0)
			*ppTN = _rotate_LL(*ppTN);
		else
			*ppTN = _rotate_LR(*ppTN);
	}
	else if (heightDiff < -1)  // right subtree is higher
	{
		if (_getHeightDiff((*ppTN)->getpRc()) < 0)
			*ppTN = _rotate_RR(*ppTN);
		else
			*ppTN = _rotate_RL(*ppTN);
	}

	return *ppTN;
}
```

insertAndRebalance()

```
template<typename K, typename V>
void T_BST<K, V>::insertAndRebalance(T_Entry<K, V> entry)
{
    _insertAndRebalance(&_root, NULL, entry);
}

template<typename K, typename V>
T_BSTN<K, V>* T_BST<K, V>::_insertAndRebalance(T_BSTN<K, V>** ppTN,
    T_BSTN<K, V>* pPr, T_Entry<K, V> entry)
{
     T_BSTN<K, V> *pTN, **ppLc, **ppRc;

     if (*ppTN == NULL) // attach a new tree node at the currently external node
     {
         pTN = new T_BSTN<K, V>(entry);
         *ppTN = pTN;
         if (pPr != NULL) // if not root
             pTN->setpPr(pPr);
         (*ppTN)->setpLc(NULL);
         (*ppTN)->setpRc(NULL);
         num_entry++;
         return *ppTN;
     }
```

```
    T_Entry<K, V> bstn_entry;
    bstn_entry = (*ppTN)->getEntry();
    if (entry < bstn_entry)  // T_Entry<K, V> must provide '<' operator overloading !!
    {
        ppLc = (*ppTN)->getppLc();
        pTN = _insertAndRebalance(ppLc, *ppTN, entry);
        if (ppTN != NULL)
        {
            (*ppTN)->setpLc(pTN);
            *ppTN = _reBalance(ppTN);
        }
    }
    else // entry >= bstn_entry
    {
        ppRc = (*ppTN)->getppRc();
        pTN = _insertAndRebalance(ppRc, *ppTN, entry);
        if (ppTN != NULL)
        {
            (*ppTN)->setpRc(pTN);
            *ppTN = _reBalance(ppTN);
        }
    }
    return *ppTN;
}
```

```
/** main.cpp  (1) */
. . . . .

void main()
{
    Student students[NUM_STUDENTS] = { ..... } ;

    T_BST<double, Student*>
       BST_ST_keyGPA_Balanced("BST_Student_KeyGPA_Balanced");
    T_Entry<double, Student*> entry_GPA_pST;
    T_BSTN<double, Student*> **ppBST_ST_kGPA_root, *pBST_ST_kGPA_root;
    fout << endl << "Testing Binary Search Tree of Student_KeyGPA with Rebalancing“
         << endl;
    ppBST_ST_kGPA_root = BST_ST_keyGPA_Balanced.getRootAddr();
    for (int i = 0; i < NUM_STUDENTS; i++)
    {
        entry_GPA_pST.setKey(students[i].getGPA());
        entry_GPA_pST.setValue(&students[i]);
        fout << "Insert inOrder (" << entry_GPA_pST << ") into "
             << BST_ST_keyGPA_Balanced.getName() << endl;
        //BST_ST.insertInOrder(entry_ID_pST);
        BST_ST_keyGPA_Balanced.insertAndRebalance(entry_GPA_pST);
    }
    fout << "\nElements in " << BST_ST_keyGPA_Balanced.getName() << " (in order of
    GPA) : " << endl;
    //BST_ST_keyGPA_Balanced.fprint_inOrder(fout);
    BST_ST_keyGPA_Balanced.fprint_with_Depth(fout);
```

```
/** main.cpp  (2) */


      fout << "\nRemoving root entry in sequence ..." << endl;
      for (int i = 0; i < NUM_STUDENTS; i++)
      {
            pBST_ST_kGPA_root = BST_ST_keyGPA_Balanced.getRoot();
            entry_GPA_pST = pBST_ST_kGPA_root->getEntry();
            fout << "\nat "<< i << "-th erase " << entry_GPA_pST << endl;
            BST_ST_keyGPA_Balanced.eraseBSTN(&pBST_ST_kGPA_root);
            BST_ST_keyGPA_Balanced.fprint_with_Depth(fout);
            if (!BST_ST_keyGPA_Balanced.empty())
            {
                  entry_GPA_pST = BST_ST_keyGPA_Balanced.maxEntry();
                  fout << "Current max is " << entry_GPA_pST << endl;
            }
      }

      fout.close();
}
```

```
Testing Binary Search Tree of Student_KeyGPA with Rebalancing
Insert inOrder ((4.50: Student[ 10,   Kim, G-M,  4.50] )) into BST_Student_KeyGPA_Balanced
Insert inOrder ((4.40: Student[ 10,   Lee, S-M,  4.40] )) into BST_Student_KeyGPA_Balanced
Insert inOrder ((4.30: Student[ 10,  Park, S-S,  4.30] )) into BST_Student_KeyGPA_Balanced
Insert inOrder ((4.20: Student[ 13,   Lee, K-M,  4.20] )) into BST_Student_KeyGPA_Balanced
Insert inOrder ((4.10: Student[ 14,  Hong, G-M,  4.10] )) into BST_Student_KeyGPA_Balanced
Insert inOrder ((4.00: Student[ 15,  Jang, S-M,  4.00] )) into BST_Student_KeyGPA_Balanced
Insert inOrder ((3.90: Student[ 16, Hwang, S-T,  3.90] )) into BST_Student_KeyGPA_Balanced
Insert inOrder ((3.80: Student[ 17,  Choi, Y-H,  3.80] )) into BST_Student_KeyGPA_Balanced
Insert inOrder ((3.70: Student[ 18,  Shin, D-J,  3.70] )) into BST_Student_KeyGPA_Balanced
Insert inOrder ((3.60: Student[ 19,  Kwak, S-B,  3.60] )) into BST_Student_KeyGPA_Balanced
Insert inOrder ((3.50: Student[ 20,  Kang, S-M,  3.50] )) into BST_Student_KeyGPA_Balanced
Insert inOrder ((3.40: Student[ 21,  Jong, S-T,  3.40] )) into BST_Student_KeyGPA_Balanced
Insert inOrder ((3.30: Student[ 22,   Lee, Y-H,  3.30] )) into BST_Student_KeyGPA_Balanced
Insert inOrder ((3.20: Student[ 23,  Sung, D-J,  3.20] )) into BST_Student_KeyGPA_Balanced
Insert inOrder ((3.10: Student[ 24,  Kong, S-B,  3.10] )) into BST_Student_KeyGPA_Balanced

Elements in BST_Student_KeyGPA_Balanced (in order of GPA) :
BST_Student_KeyGPA_Balanced : current size (15)
            (4.50: Student[ 10,   Kim, G-M,  4.50] )
        (4.40: Student[ 10,   Lee, S-M,  4.40] )
            (4.30: Student[ 10,  Park, S-S,  4.30] )
    (4.20: Student[ 13,   Lee, K-M,  4.20] )
            (4.10: Student[ 14,  Hong, G-M,  4.10] )
        (4.00: Student[ 15,  Jang, S-M,  4.00] )
            (3.90: Student[ 16, Hwang, S-T,  3.90] )
(3.80: Student[ 17,  Choi, Y-H,  3.80] )
            (3.70: Student[ 18,  Shin, D-J,  3.70] )
        (3.60: Student[ 19,  Kwak, S-B,  3.60] )
            (3.50: Student[ 20,  Kang, S-M,  3.50] )
    (3.40: Student[ 21,  Jong, S-T,  3.40] )
            (3.30: Student[ 22,   Lee, Y-H,  3.30] )
        (3.20: Student[ 23,  Sung, D-J,  3.20] )
            (3.10: Student[ 24,  Kong, S-B,  3.10] )
```

```
Removing root entry in sequence ...

at 0-th erase (3.80: Student[ 17,  Choi, Y-H,  3.80] )
BST_Student_KeyGPA_Balanced : current size (14)
            (4.50: Student[ 10,   Kim, G-M,  4.50] )
        (4.40: Student[ 10,   Lee, S-M,  4.40] )
            (4.30: Student[ 10,  Park, S-S,  4.30] )
    (4.20: Student[ 13,   Lee, K-M,  4.20] )
            (4.10: Student[ 14,  Hong, G-M,  4.10] )
        (4.00: Student[ 15,  Jang, S-M,  4.00] )
(3.90: Student[ 16, Hwang, S-T,  3.90] )
            (3.70: Student[ 18,  Shin, D-J,  3.70] )
        (3.60: Student[ 19,  Kwak, S-B,  3.60] )
            (3.50: Student[ 20,  Kang, S-M,  3.50] )
    (3.40: Student[ 21,  Jong, S-T,  3.40] )
            (3.30: Student[ 22,   Lee, Y-H,  3.30] )
        (3.20: Student[ 23,  Sung, D-J,  3.20] )
            (3.10: Student[ 24,  Kong, S-B,  3.10] )
Current max is (4.50: Student[ 10,   Kim, G-M,  4.50] )

at 1-th erase (3.90: Student[ 16, Hwang, S-T,  3.90] )
BST_Student_KeyGPA_Balanced : current size (13)
            (4.50: Student[ 10,   Kim, G-M,  4.50] )
        (4.40: Student[ 10,   Lee, S-M,  4.40] )
            (4.30: Student[ 10,  Park, S-S,  4.30] )
    (4.20: Student[ 13,   Lee, K-M,  4.20] )
        (4.10: Student[ 14,  Hong, G-M,  4.10] )
(4.00: Student[ 15,  Jang, S-M,  4.00] )
            (3.70: Student[ 18,  Shin, D-J,  3.70] )
        (3.60: Student[ 19,  Kwak, S-B,  3.60] )
            (3.50: Student[ 20,  Kang, S-M,  3.50] )
    (3.40: Student[ 21,  Jong, S-T,  3.40] )
            (3.30: Student[ 22,   Lee, Y-H,  3.30] )
        (3.20: Student[ 23,  Sung, D-J,  3.20] )
            (3.10: Student[ 24,  Kong, S-B,  3.10] )
Current max is (4.50: Student[ 10,   Kim, G-M,  4.50] )

at 9-th erase (4.30: Student[ 10,  Park, S-S,  4.30] )
BST_Student_KeyGPA_Balanced : current size (5)
    (4.50: Student[ 10,   Kim, G-M,  4.50] )
(4.40: Student[ 10,   Lee, S-M,  4.40] )
        (3.30: Student[ 22,   Lee, Y-H,  3.30] )
    (3.20: Student[ 23,  Sung, D-J,  3.20] )
        (3.10: Student[ 24,  Kong, S-B,  3.10] )
Current max is (4.50: Student[ 10,   Kim, G-M,  4.50] )

at 10-th erase (4.40: Student[ 10,   Lee, S-M,  4.40] )
BST_Student_KeyGPA_Balanced : current size (4)
    (4.50: Student[ 10,   Kim, G-M,  4.50] )
(3.30: Student[ 22,   Lee, Y-H,  3.30] )
    (3.20: Student[ 23,  Sung, D-J,  3.20] )
        (3.10: Student[ 24,  Kong, S-B,  3.10] )
Current max is (4.50: Student[ 10,   Kim, G-M,  4.50] )

at 11-th erase (3.30: Student[ 22,   Lee, Y-H,  3.30] )
BST_Student_KeyGPA_Balanced : current size (3)
    (4.50: Student[ 10,   Kim, G-M,  4.50] )
(3.20: Student[ 23,  Sung, D-J,  3.20] )
    (3.10: Student[ 24,  Kong, S-B,  3.10] )
Current max is (4.50: Student[ 10,   Kim, G-M,  4.50] )

at 12-th erase (3.20: Student[ 23,  Sung, D-J,  3.20] )
BST_Student_KeyGPA_Balanced : current size (2)
(4.50: Student[ 10,   Kim, G-M,  4.50] )
    (3.10: Student[ 24,  Kong, S-B,  3.10] )
Current max is (4.50: Student[ 10,   Kim, G-M,  4.50] )

at 13-th erase (4.50: Student[ 10,   Kim, G-M,  4.50] )
BST_Student_KeyGPA_Balanced : current size (1)
(3.10: Student[ 24,  Kong, S-B,  3.10] )
Current max is (3.10: Student[ 24,  Kong, S-B,  3.10] )

at 14-th erase (3.10: Student[ 24,  Kong, S-B,  3.10] )
BST_Student_KeyGPA_Balanced is empty now !!
```

이진트리와 주요 자료구조들의 비교

자료구조	특성
단순 배열	컴파일 단계에서 크기가 지정되어 변경되지 않는 크기의 배열
동적 배열	프로그램 실행 단계에서 크기가 지정되며, 프로그램 실행 중에 크기를 변경할 수 있는 배열
구조체 배열	구조체로 배열원소가 지정되는 배열
연결형 리스트	확장성이 있으며, 단일연결형 또는 이중 연결형으로 구성 가능. 다양한 컨테이너 자료구조의 내부적인 자료구조로도 활용됨
스택	Last In First Out (LIFO) 특성을 가짐. 배열 또는 연결형 리스트로 구현할 수 있음.
큐	First In First Out (FIFO) 특성을 가짐. 배열 또는 연결형 리스트로 구현할 수 있음.
우선 순위 큐	컨테이너 내부에 가장 우선 순위가 높은 데이터 항목을 추출할 수 있도록 관리하며, 배열 또는 자기 참조 구조체로 구현할 수 있음
이진 탐색 트리	컨테이너 내부에 포함된 데이터 항목들을 정렬된 상태로 관리하여야 할 때 매우 효율적임. 단순 이진 탐색 트리의 경우 편중될 수 있으며, 편중된 경우 검색 성능이 저하되기 때문에, 밸런싱이 필요함.
해시 테이블 (Hash Table)	컨테이너 자료구조에 포함된 항목들을 문자열 (string) 또는 긴 숫자를 키 (key)로 사용하여 관리하여야 하는 경우, key로부터 해시 값을 구하고, 이 해시 값을 배열의 인덱스로 사용함.
Map	key와 항목 간에 1:1 관계가 유지되어야 하는 경우에 사용되며, 해시 테이블을 기반으로 구현할 수 있음
Dictionary	key와 항목 간에 1:N 관계가 유지되어야 하는 경우에 사용되며, 해시 테이블을 기반으로 구현할 수 있음
trie	텍스트 (문자열)의 탐색에 사용되는 구조. 예측구문 (predictive test) 제시, 인터넷 라우터의 최장 접두어 매칭 (longest prefix matching) 등에 사용됨
그래프	정점 (vertex)/노드 (node)로 개체 (object)가 표현되고, 간선 (edge)/링크(link)들을 사용하여 개체 간의 관계를 표현하는 경우에 적합함. 그래프를 기반으로 경로 탐색, 최단 거리 경로 탐색, 신장 트리 (spanning tree) 탐색 등에 활용됨.

Homework 10

Homework 10

10.1 도서관에 소장된 서적 (book) 정보를 관리하기 위한 자료구조 구현.

(1) 서적의 정보를 표현하기 위하여 다음과 같은 항목이 포함 되는 class Book을 구현하라: 도서명 (string), ISBN (international standard book number) (int), 저자명 (string), 출판연월일(class Date).

(2) class Book을 사용하여 15권의 서적 정보를 담는 배열 books[]를 준비하라.

(3) 서적의 모든 정보를 출력하는 operator<<() 연산자 오버로딩 함수로 구현하라.

(4) class T_Entry<K, V>를 저장하는 균형화 이진 탐색 트리 (Balanced Binaray Search Tree, BBST)를 구현하라. 이 균형화 이진 탐색 트리는 지정된 key 값에 따라 정렬이 되도록 하며, 자동적으로 균형을 잡을 수 있도록 구현되어야 한다.

(5) 위에서 만든 도서관 서적 정보에서 도서명을 기준으로 정렬이 되도록 이진 탐색 트리 (BST_BK_title)를 구현하고, 이진 탐색 트리의 각 노드가 class Book 배열 books[]의 해당 서적 레코드를 가리키도록 하라. 서적명을 입력 받아 이진 탐색 트리를 검색하고, 그 서적의 모든 정보를 출력하는 프로그램을 작성하라.

(6) 위에서 만든 도서관 서적 정보에서 저자명을 기준으로 정렬이 되도록 이진 탐색 트리 (BST_BK_author)를 구현하고, 이진 탐색 트리의 각 노드가 book 객체 배열의 해당 서적 레코드를 가리키도록 하라. 이진 탐색 트리에서 동일 저자가 저술한 다수의 서적들이 함께 관리될 수 있도록 이진 탐색 트리를 구현하라.
동일 저자가 저술한 서적들인 경우, 먼저 입력된 순서로 저장될 수 있게 하라.
저자명을 입력 받아 이진 탐색 트리를 검색하고, 그 저자가 저술한 모든 서적들의 모든 정보를 출력하는 프로그램을 작성하라.

(7) 위에서 만든 도서관 서적 정보에서 출판연월일을 기준으로 정렬이 되도록 이진 탐색 트리 (BST_BK_publishDate)를 구현하고, 이진 탐색 트리의 각 노드가 객체 배열의 해당 서적 레코드를 가리키도록 하라. 출판연월일은 두개 입력받아, 지정된 두 날짜 사이에 출판되었던 서적들을 모두 출력하는 프로그램을 작성하라.

(8) 저자명과 출판연월일을 입력받아, 그 저자가 지정된 출판연월일 이후에 저술한 모든 서적을 출력하는 프로그램을 작성하라.

```
class Book
{
    friend ostream& operator<<(ostream& fout, Book& bk)
    {
            fout << "[" ;
            fout.setf(ios::left);
            fout << setw(8) << bk.title;
            fout << ", " << setw(8) << bk.author;
            fout << ", " << bk.pubDate;
            fout << "]" ;
            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;
};
```

```
Input student[] 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 , Abraham , (1980.1 .1 ) ]
[Book_10 , Chalie  , (1970.1 .1 ) ]
[Book_11 , Turk    , (1985.1 .1 ) ]
[Book_12 , Yankee  , (1977.1 .1 ) ]
[Book_13 , Zulu    , (2018.1 .1 ) ]
[Book_14 , Abc     , (2015.1 .1 ) ]
[Book_15 , Dail    , (2000.1 .1 ) ]


Testing Binary Search Tree with Rebalancing

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

Entries in BBST_BK_keyAuthor (in order of Book Author) :
(Abc     : [Book_14 , Abc     , (2015.1 .1 ) ])
(Abraham : [Book_09 , Abraham , (1980.1 .1 ) ])
(Brown   : [Book_08 , Brown   , (2012.1 .1 ) ])
(Chalie  : [Book_10 , Chalie  , (1970.1 .1 ) ])
(Choi    : [Book_06 , Choi    , (2003.1 .1 ) ])
(Dail    : [Book_15 , Dail    , (2000.1 .1 ) ])
(Hwang   : [Book_05 , Hwang   , (2001.1 .1 ) ])
(Kim     : [Book_01 , Kim     , (2020.1 .1 ) ])
(Kim     : [Book_02 , Kim     , (2010.1 .1 ) ])
(Kim     : [Book_03 , Kim     , (2013.1 .1 ) ])
(Lee     : [Book_04 , Lee     , (2011.1 .1 ) ])
(Park    : [Book_07 , Park    , (2009.1 .1 ) ])
(Turk    : [Book_11 , Turk    , (1985.1 .1 ) ])
(Yankee  : [Book_12 , Yankee  , (1977.1 .1 ) ])
(Zulu    : [Book_13 , Zulu    , (2018.1 .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 ) ]

Entries in BBST_BK_keyPubDate (in order of Book Publication Date) :
((       1970.1 .1 ) : [Book_10 , Chalie  , (1970.1 .1 ) ])
((       1977.1 .1 ) : [Book_12 , Yankee  , (1977.1 .1 ) ])
((       1980.1 .1 ) : [Book_09 , Abraham , (1980.1 .1 ) ])
((       1985.1 .1 ) : [Book_11 , Turk    , (1985.1 .1 ) ])
((       2000.1 .1 ) : [Book_15 , Dail    , (2000.1 .1 ) ])
((       2001.1 .1 ) : [Book_05 , Hwang   , (2001.1 .1 ) ])
((       2003.1 .1 ) : [Book_06 , Choi    , (2003.1 .1 ) ])
((       2009.1 .1 ) : [Book_07 , Park    , (2009.1 .1 ) ])
((       2010.1 .1 ) : [Book_02 , Kim     , (2010.1 .1 ) ])
((       2011.1 .1 ) : [Book_04 , Lee     , (2011.1 .1 ) ])
((       2012.1 .1 ) : [Book_08 , Brown   , (2012.1 .1 ) ])
((       2013.1 .1 ) : [Book_03 , Kim     , (2013.1 .1 ) ])
((       2015.1 .1 ) : [Book_14 , Abc     , (2015.1 .1 ) ])
((       2018.1 .1 ) : [Book_13 , Zulu    , (2018.1 .1 ) ])
((       2020.1 .1 ) : [Book_01 , Kim     , (2020.1 .1 ) ])
```