DATA
STRUCTURES
USING C++

09 CHAPTER

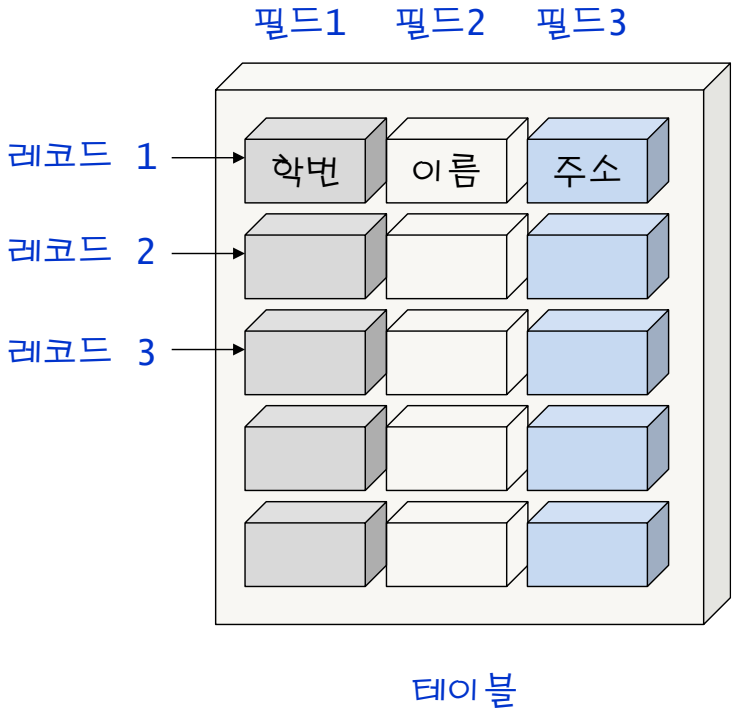
이진 탐색 트리

이진탐색트리

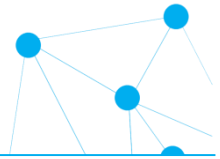
- **탐색(search)**은 가장 중요한 컴퓨터 응용의 하나
- **이진 탐색 트리(BST, Binary Search Tree)**
  - 이진트리 기반의 탐색을 위한 자료 구조
  - 효율적인 탐색 작업을 위한 자료 구조

탐색 관련 용어

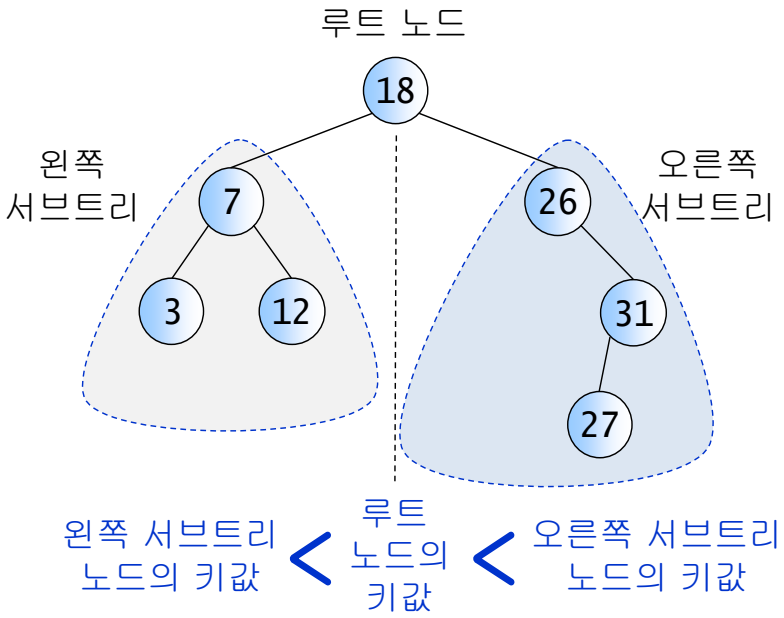
- 레코드(record)
- 필드(field)
- 테이블(table)
- 키(key)
- 주요키(primary key)



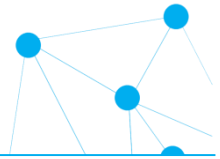
이진탐색트리의 정의



- 탐색작업을 효율적으로 하기 위한 자료구조
  - key(왼쪽서브트리)≤key(루트노드)≤key(오른쪽서브트리)
  - 이진탐색를 중위순회하면 오름차순으로 정렬된 값을 얻을 수 있다.



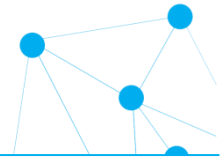
이진탐색트리 ADT

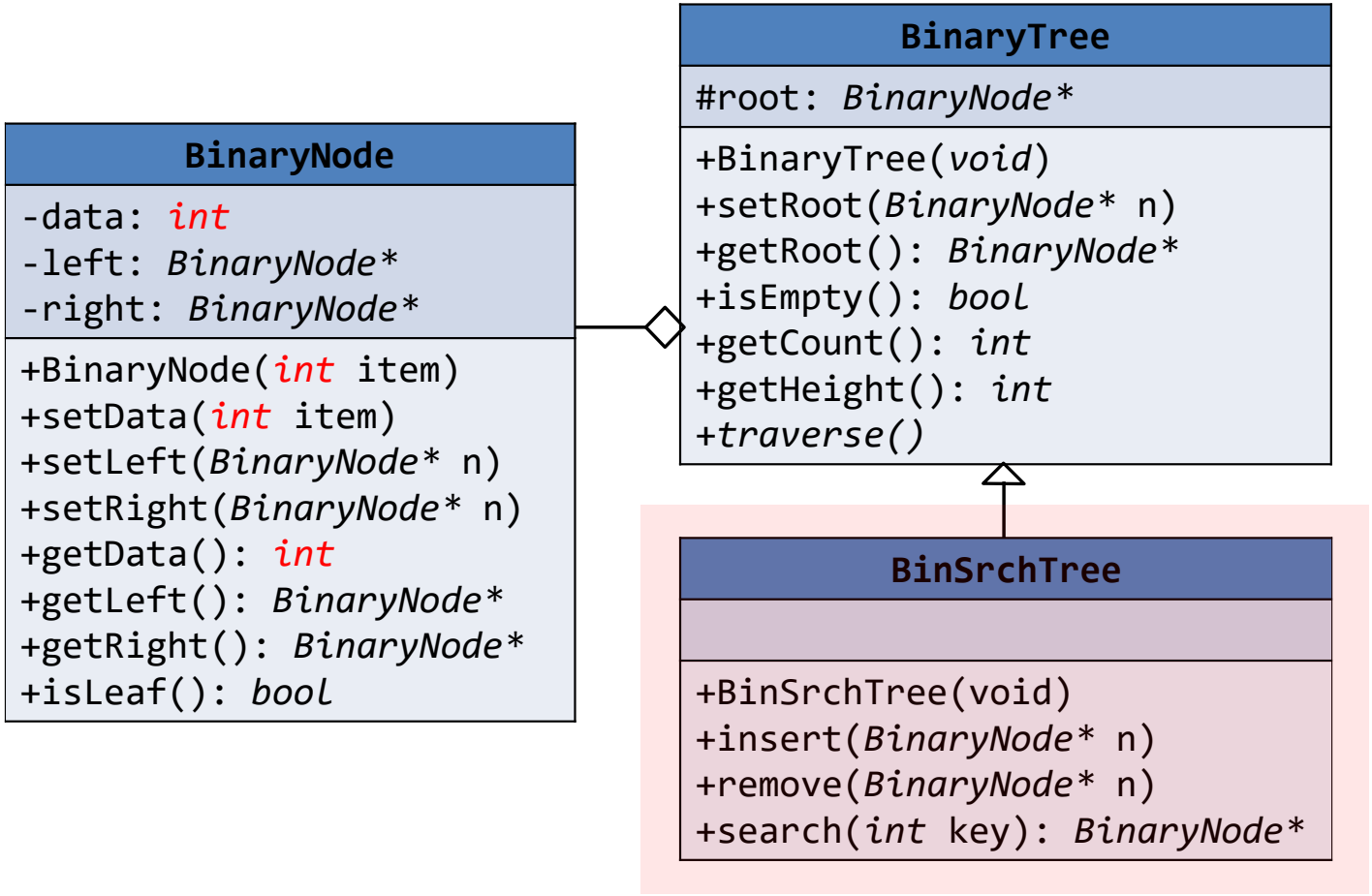


데이터:
이진 탐색 트리(BST)의 특성을 만족하는 이진트리: 어떤 노드 x의 왼쪽 서브트리의 키들은 x의 키보다 작고, 오른쪽 서브트리의 키들은 x의 키보다 크다. 이때 왼쪽과 오른쪽 서브트리도 모두 이진 탐색 트리이다.

연산:
- insert(n): 이진 탐색 트리의 특성을 유지하면서 새로운 노드 n을 이진 탐색 트리에 삽입한다.
- remove(n): 이진 탐색 트리의 특성을 유지하면서 노드 n을 트리에서 삭제한다.
- search(key): 키 값이 key인 노드를 찾아 반환한다.

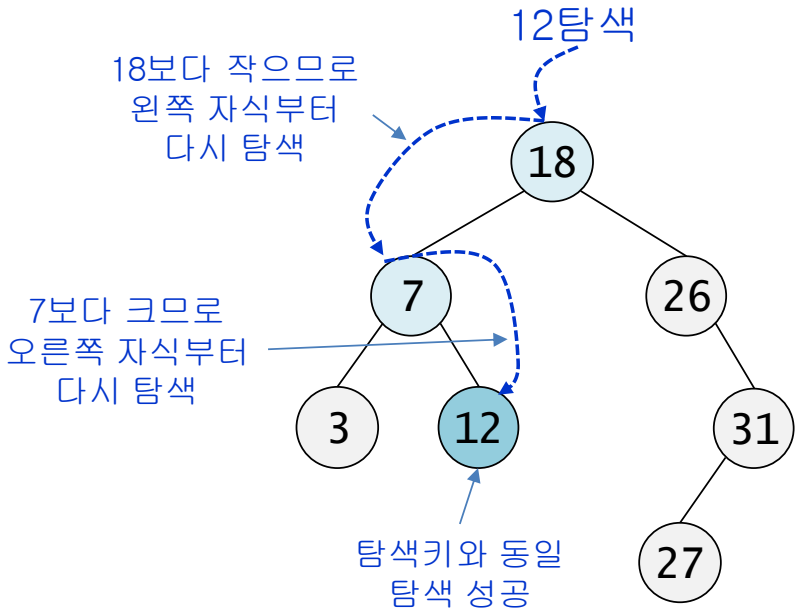
이진탐색트리 클래스 다이어그램



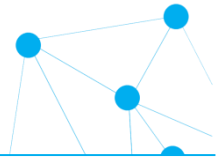


이진탐색트리::탐색연산

- 탐색 연산
  - 비교한 결과가 같으면 탐색이 성공적으로 끝난다.
  - 키 값이 루트보다 작으면 ➔ 왼쪽 자식을 기준으로 다시 탐색
  - 키 값이 루트보다 크면 ➔ 오른쪽 자식을 기준으로 다시 탐색



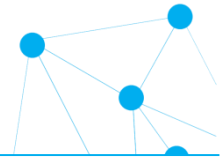
탐색연산의 구현



- 다양한 방법으로 구현할 수 있음
  - 순환적인 구현, 반복적인 구현
  - 일반 함수 구현, 트리의 멤버함수 구현, 노드의 멤버함수 구현

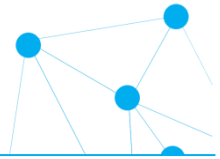
- 예: 순환적인 일반함수 구현

```
// 키 값으로 노드를 탐색하는 함수
BinaryNode* searchRecur( BinaryNode *n, int key )
{
     if( n == NULL ) return NULL;       // n이 NULL
     if( key == n->getData() )          // (1) key == 현재노드의 data
          return n;
     else if (key < n->getData() )      // (2) key < 현재노드의 data
          return searchRecur( n->getLeft(), key );
     else                               // (3) key > 현재노드의 data
          return searchRecur( n->getRight(), key );
}
```



- 예: 반복적인 일반함수 구현

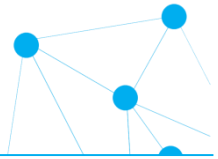
```
// 키 값으로 노드를 탐색하는 함수
BinaryNode* searchIter( BinaryNode *n, int key )
{
        while(n != NULL){
                if( key == n->getData() ) return n;
                else if( key < n-> getData() )
                        n = node->getLeft();
                else n = node->getRight();
        }
        return NULL;
}
```



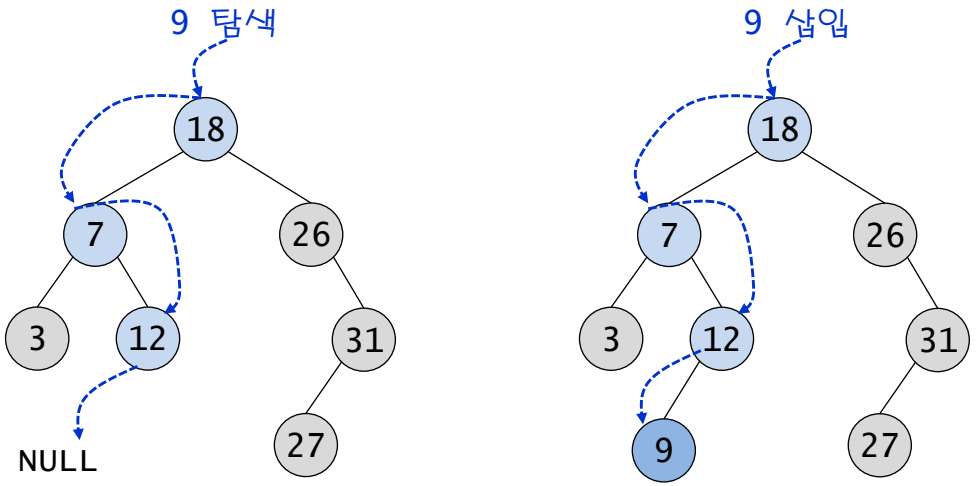
- 예: 노드의 멤버함수로 구현(순환적)

```
// 키 값으로 노드를 탐색하는 함수
BinaryNode* BinaryNode::search( int key )
{
      if( key == data )                          // (1) key == 현재노드의 data
          return this;
      else if (key < data && left!=NULL )        // (2) key < 현재노드의 data
            return left->search( key );
      else if (key > data && right!=NULL )       // (3) key > 현재노드의 data
            return right->search( key );
      else                                       // (4) 찾는 노드 없음
            return NULL;
}
```

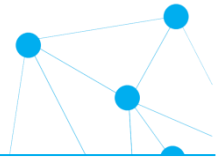
이진탐색트리::삽입연산



- 삽입 연산
  - 이진 탐색 트리에 원소를 삽입하기 위해서는 먼저 탐색을 수행하는 것이 필요
  - 탐색에 실패한 위치가 바로 새로운 노드를 삽입하는 위치



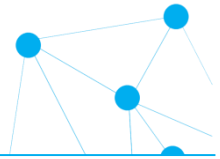
삽입연산 알고리즘



```
insert (root, n)

if KEY(n) = KEY(root)                    // root와 키가 같으면
    then return;                         // return
else if KEY(n) < KEY(root) then          // root보다 키가 작으면
    if LEFT(root) = NULL                 // root의 왼쪽 자식이
         then LEFT(root) ← n;            // 없으면 n이 왼쪽 자식
         else insert(LEFT(root),n);      // 있으면 순환 호출
else                                     // root보다 키가 크면
    if RIGHT(root) = NULL
    then RIGHT(root) ← n;
    else insert(RIGHT(root),n);
```

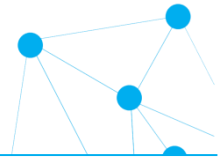
삽입연산 구현



- 순환적으로 구현한 삽입 연산(일반함수 또는 트리의 멤버함수)

```
// 이진 탐색 트리의 삽입 함수
void insertRecur( BinaryNode* r, BinaryNode* n ) {
     if( n->getData() == r->getData() )
          return;
     else if( n->getData() < r->getData() ) {
          if( r->getLeft() == NULL ) r->setLeft(n);
          else insertRecur( r->getLeft(), n );
     }
     else {
          if( r->getRight() == NULL ) r->setRight(n);
          else insertRecur( r->getRight(), n );
     }
}
```

이진탐색트리 :: 삭제연산



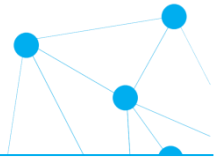
- 노드 삭제의 3가지 경우

  1. 삭제하려는 노드가 단말 노드일 경우

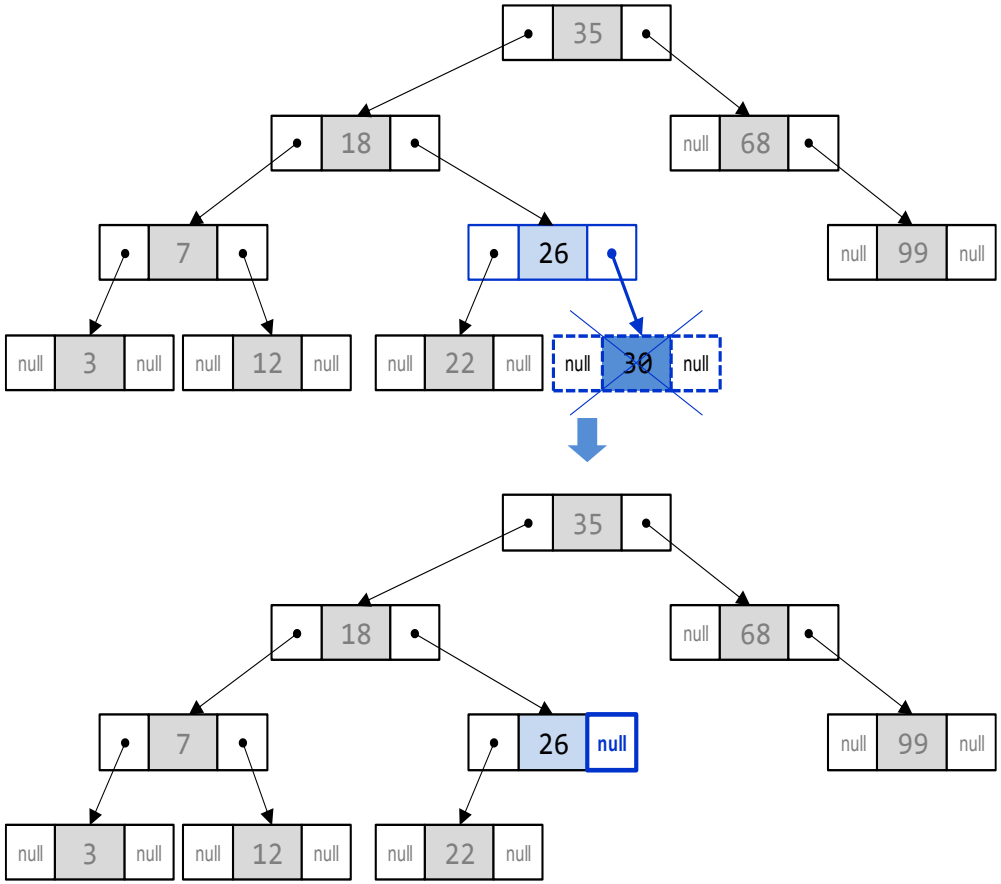
  2. 삭제하려는 노드가 하나의 왼쪽이나 오른쪽 서브 트리중 하나만 가지고 있는 경우

  3. 삭제하려는 노드가 두개의 서브 트리 모두 가지고 있는 경우

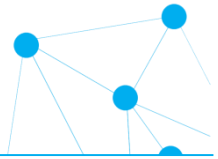
Case 1: 단말 노드 삭제



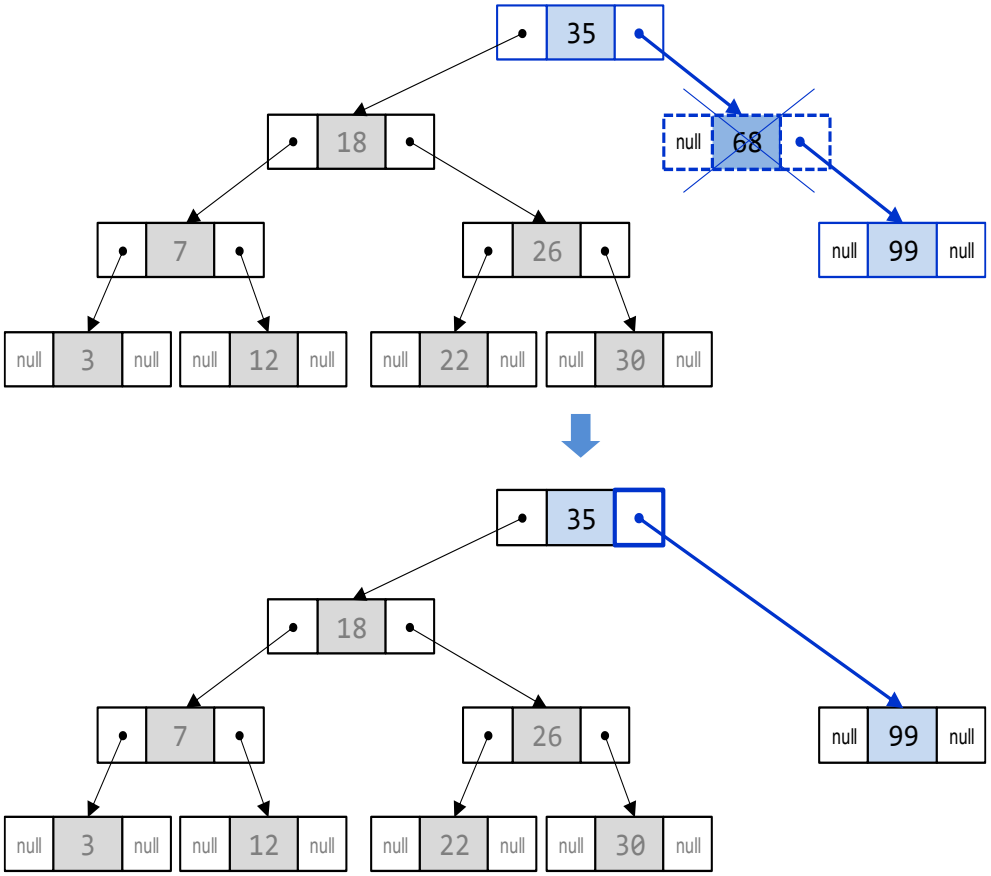
- 단말노드의 부모노드를 찾아서 연결을 끊으면 된다.



Case 2: 자식이 하나인 노드 삭제

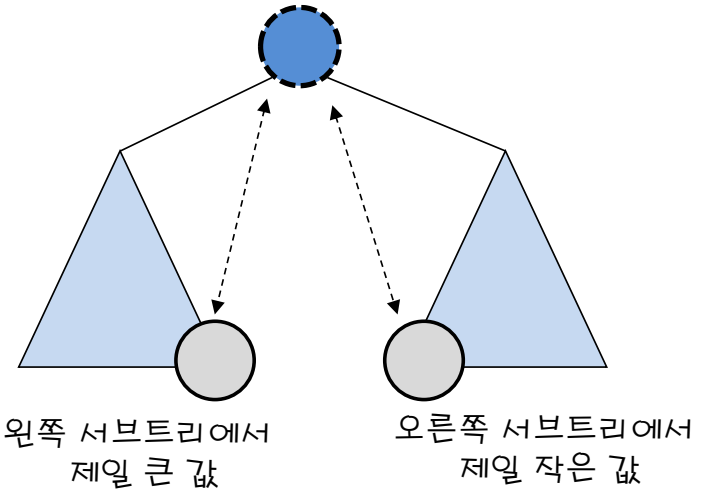


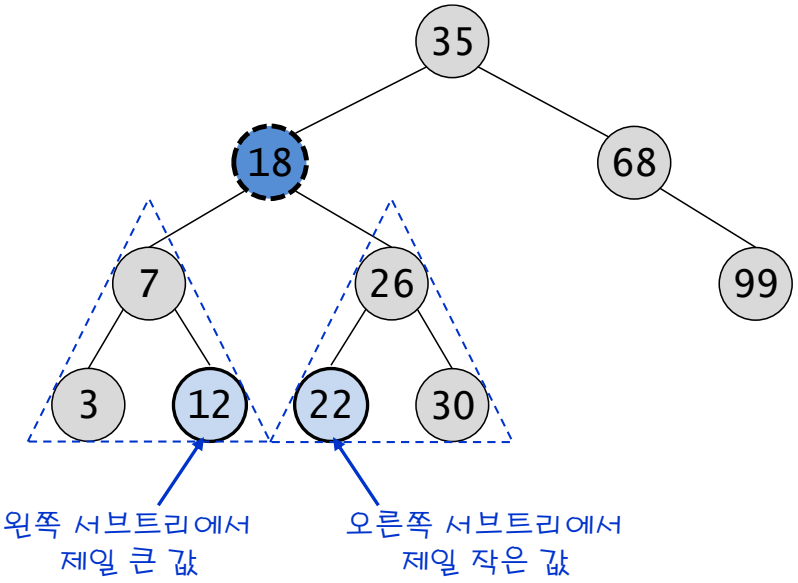
- 노드는 삭제하고 서브 트리는 부모 노드에 붙여줌



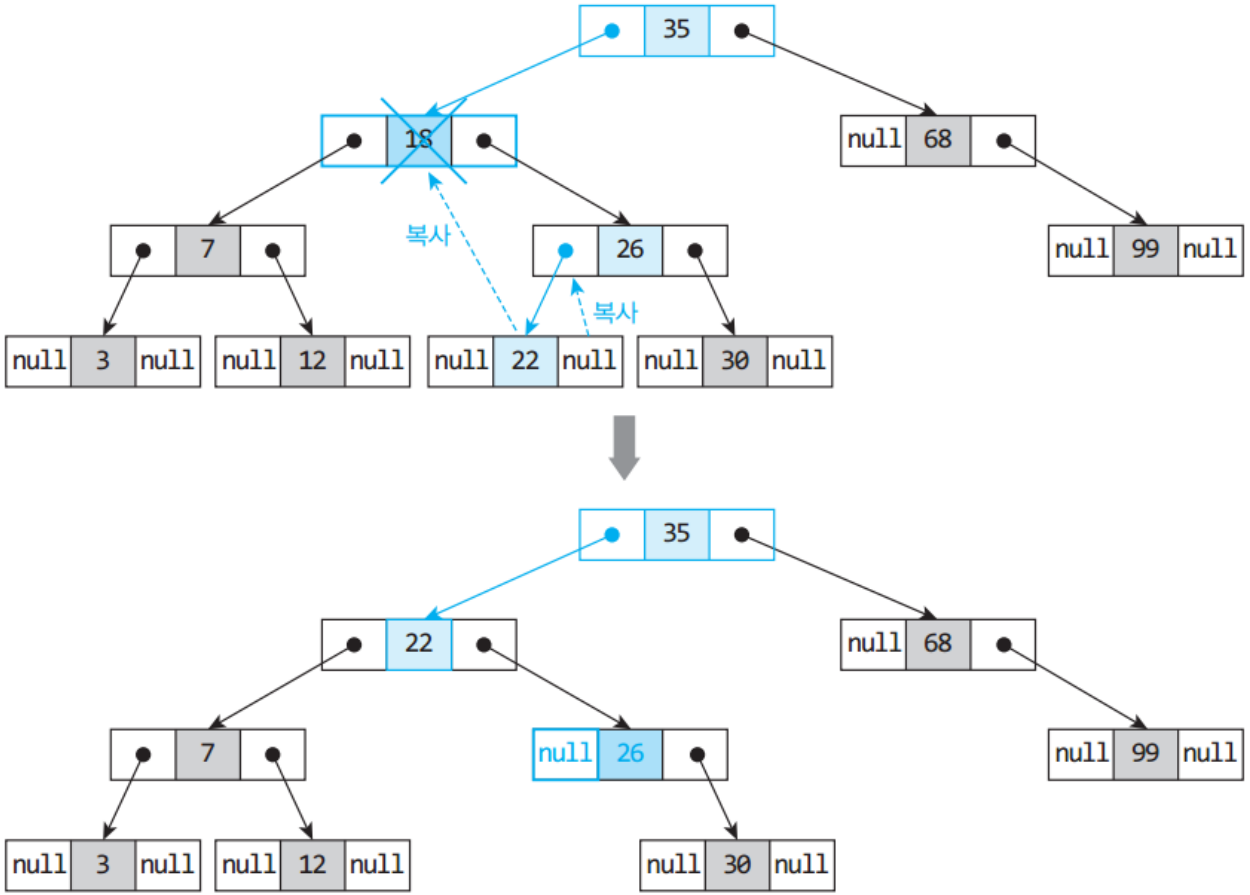
Case 3: 두 개의 자식을 가진 노드 삭제

- 가장 비슷한 값을 가진 노드를 삭제 노드 위치로 가져옴
- 후계 노드의 선택

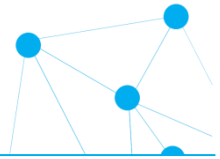




- Case3: 노드 18 삭제 과정



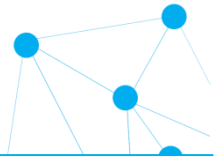
삭제연산 구현



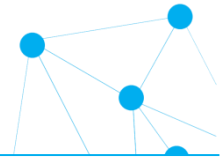
- 트리의 멤버함수로 구현
  - 노드이 멤버 함수로는 구현이 불가능 함. Why???
  - 함수 중복 사용

```
void remove (int data) {
    if( isEmpty() ) return;

    BinaryNode *parent = NULL;
    BinaryNode *node = root;
    while( node != NULL && node->getData() != data ) {
        parent = node;
        node = (data<node->getData()) ? node->getLeft():node->getRight();
    }
    if( node == NULL ) {
        printf(" Error: key is not in the tree!\n");
        return;
    }
    else remove ( parent, node );
}
```



```
void remove (BinaryNode *parent, BinaryNode *node) {
    // case 1
    if( node->isLeaf() ) {
        if( parent == NULL ) root = NULL;
        else {
            if( parent->getLeft() == node )
                parent->setLeft(NULL);
            else
                parent->setRight(NULL);
        }
    }
    // case 2
    else if( node->getLeft()== NULL|| node->getRight()==NULL ) {
        BinaryNode *child = (node->getLeft() != NULL )
                          ? node->getLeft() : node->getRight();
        if( node == root )
            root = child;
        else {
            if( parent->getLeft() == node )
                parent->setLeft(child);
            else
                parent->setRight(child);
        }
    }
```

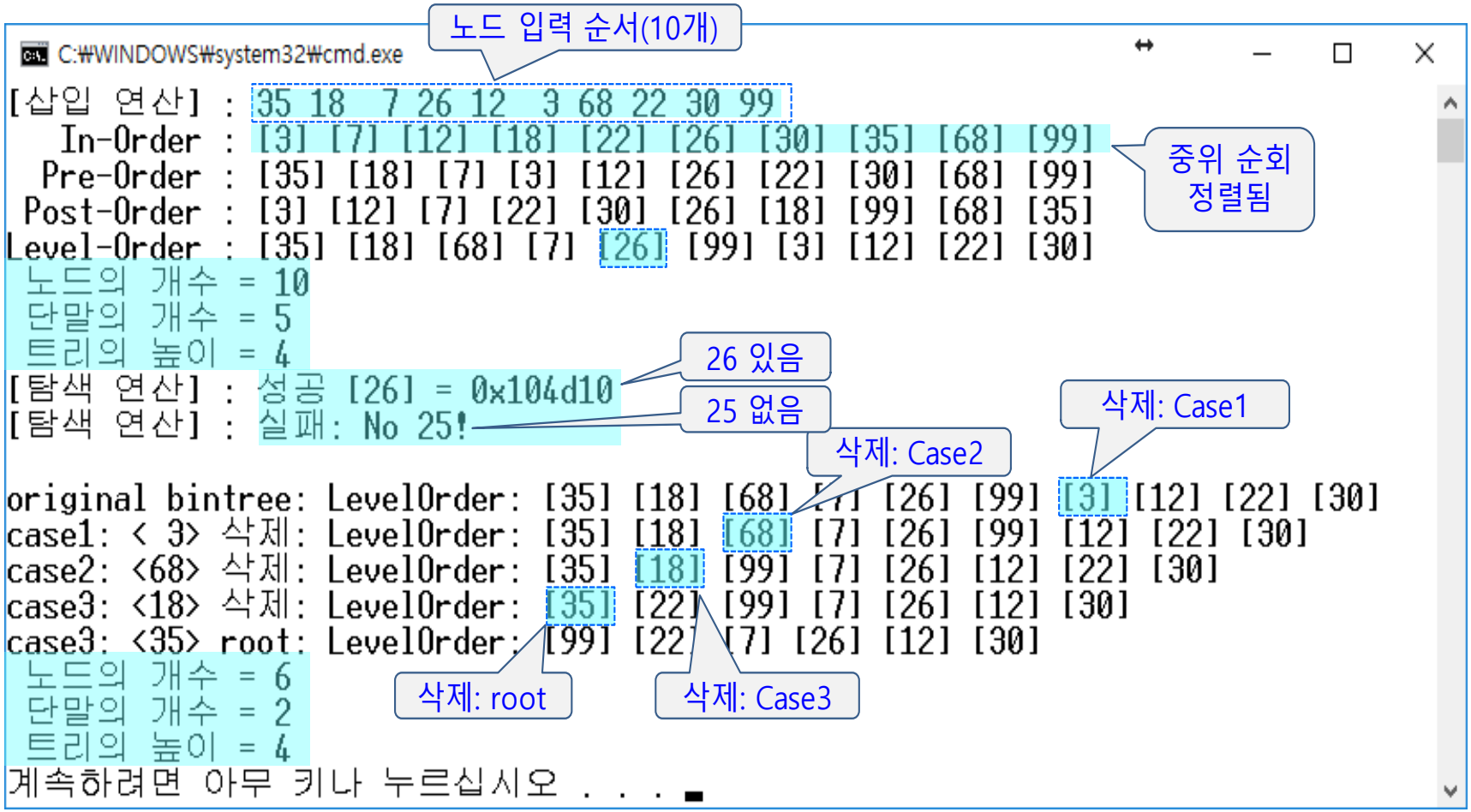


```
    // case 3
    else {
        BinaryNode* succp = node;
        BinaryNode* succ = node->getRight();
        while (succ->getLeft() != NULL) {
            succp = succ;
            succ = succ->getLeft();
        }

        if( succp->getLeft() == succ )
            succp->setLeft(succ->getRight());
        else
            succp->setRight(succ->getRight());

        node->setData(succ->getData());
        node = succ;
    }
    delete node;
}
```

전체 프로그램 실행결과


노드 입력 순서(10개)
C:₩WINDOWS₩system32₩cmd.exe
[삽입 연산] : 35 18 7 26 12 3 68 22 30 99
In-Order : [3] [7] [12] [18] [22] [26] [30] [35] [68] [99]
Pre-Order : [35] [18] [7] [3] [12] [26] [22] [30] [68] [99]
Post-Order : [3] [12] [7] [22] [30] [26] [18] [99] [68] [35]
Level-Order : [35] [18] [68] [7] [26] [99] [3] [12] [22] [30]
중위 순회 정렬됨
노드의 개수 = 10
단말의 개수 = 5
트리의 높이 = 4
[탐색 연산] : 성공 [26] = 0x104d10
26 있음
[탐색 연산] : 실패: No 25!
25 없음
삭제: Case1
삭제: Case2
original bintree: LevelOrder: [35] [18] [68] [7] [26] [99] [3] [12] [22] [30]
case1: < 3> 삭제: LevelOrder: [35] [18] [68] [7] [26] [99] [12] [22] [30]
case2: <68> 삭제: LevelOrder: [35] [18] [99] [7] [26] [12] [22] [30]
case3: <18> 삭제: LevelOrder: [35] [22] [99] [7] [26] [12] [30]
case3: <35> root: LevelOrder: [99] [22] [7] [26] [12] [30]
삭제: root
삭제: Case3
노드의 개수 = 6
단말의 개수 = 2
트리의 높이 = 4
계속하려면 아무 키나 누르십시오 . . .

이진탐색트리의 성능

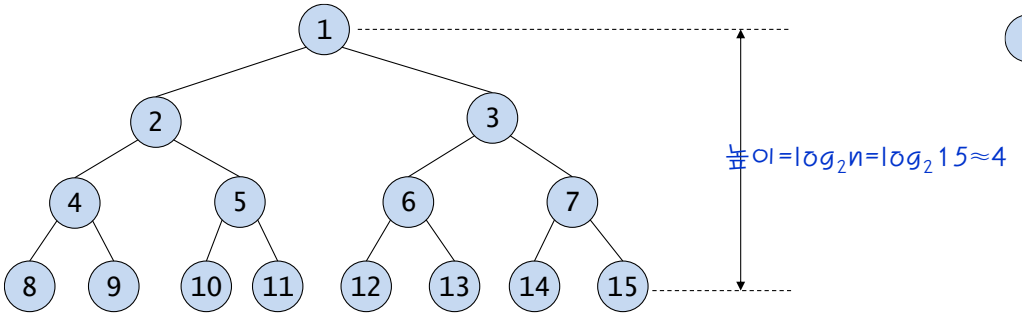
- 이진 탐색 트리에서의 탐색, 삽입, 삭제 연산의 시간 복잡도는 트리의 높이를 h라고 했을때 h에 비례한다

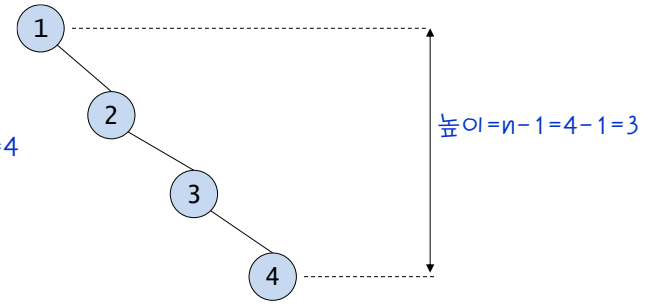
□ 최선의 경우

- 이진 트리가 균형적으로 생성되어 있는 경우: h=log2n
- 시간복잡도: **O(logn)**

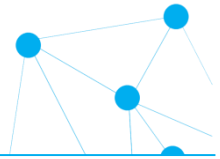
□ 최악의 경우

- 경사이진트리: h=n
- 시간복잡도: **O(n)**



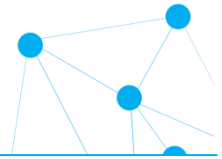


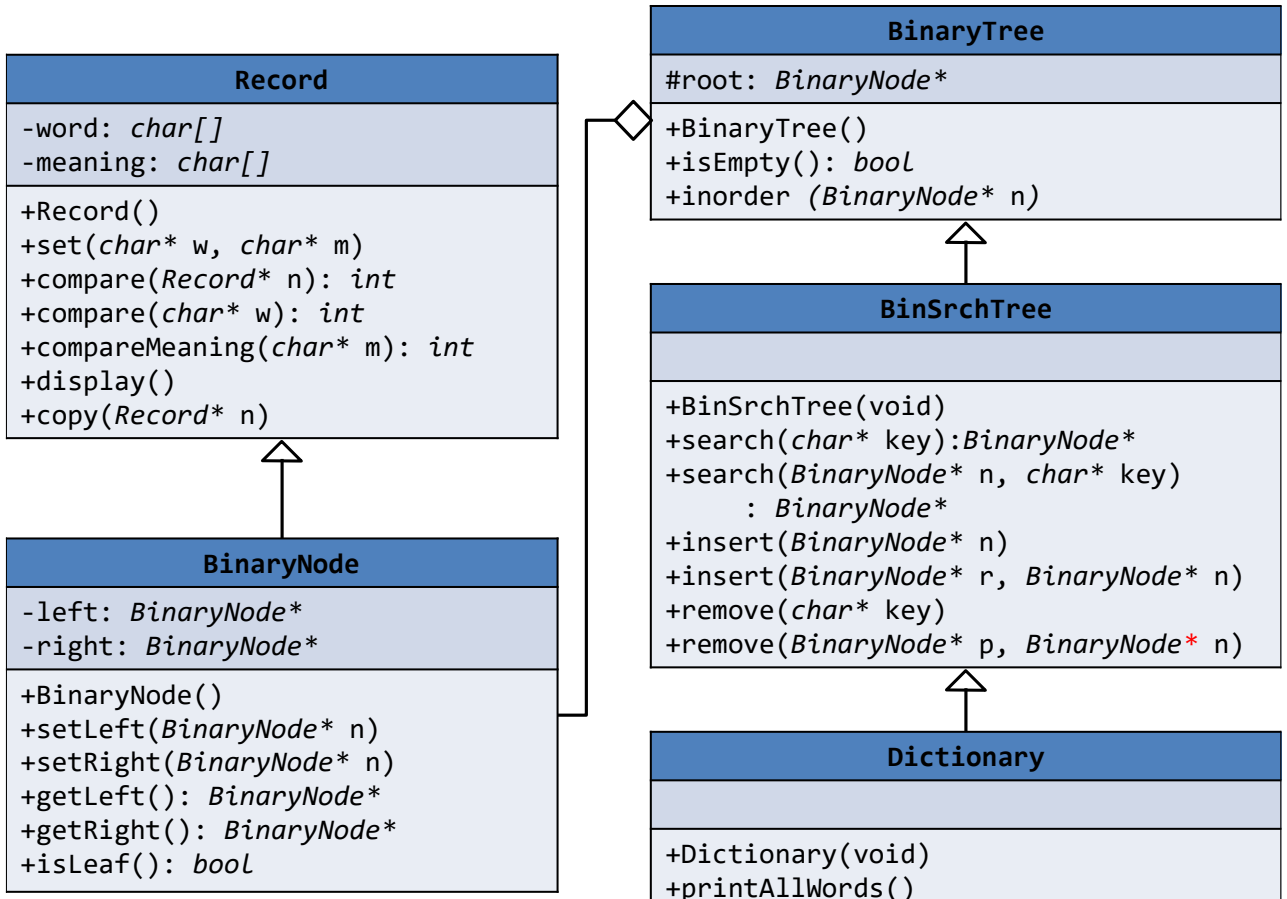
이진탐색트리의 응용 : 영어 사전



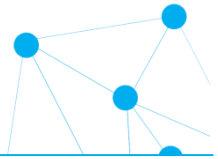
- 영어 사전 기능
  - 입력 (i) : 단어와 의미를 입력하여 하나의 노드 추가
  - 삭제 (d) : 단어를 입력하면 해당 노드를 찾아 트리에서 제거
  - 단어 탐색 (w) : 단어를 입력하면 해당 단어의 노드를 찾아 화면에 출력
  - 의미 탐색 (m) : 의미를 입력하면 해당 의미의 노드를 찾아 화면에 출력
  - 사전 출력 (p) : 사전의 모든 단어를 알파벳 순서대로(inorder) 화면에 출력
  - 종료 (q) : 프로그램을 종료

클래스 다이어그램



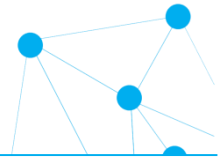

Record
-word: char[]
-meaning: char[]
+Record()
+set(char* w, char* m)
+compare(Record* n): int
+compare(char* w): int
+compareMeaning(char* m): int
+display()
+copy(Record* n)
BinaryNode
-left: BinaryNode*
-right: BinaryNode*
+BinaryNode()
+setLeft(BinaryNode* n)
+setRight(BinaryNode* n)
+getLeft(): BinaryNode*
+getRight(): BinaryNode*
+isLeaf(): bool
BinaryTree
#root: BinaryNode*
+BinaryTree()
+isEmpty(): bool
+inorder (BinaryNode* n)
BinSrchTree
+BinSrchTree(void)
+search(char* key):BinaryNode*
+search(BinaryNode* n, char* key)
: BinaryNode*
+insert(BinaryNode* n)
+insert(BinaryNode* r, BinaryNode* n)
+remove(char* key)
+remove(BinaryNode* p, BinaryNode* n)
Dictionary
+Dictionary(void)
+printAllWords()
+searchWord(char* w): BinaryNode*
+searchMeaning(char* m): BinaryNode*
+searchMeaning(BinaryNode* n, char* m)
: BinaryNode*

영어사전 프로그램



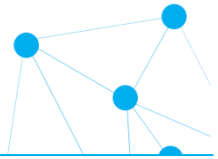
```
#include "Dictionary.h"
#include <conio.h>
void help() {printf("[사용법] i-추가 d-삭제 w-단어검색 m-의미검색 p-출력 q-종료 =>"); }
void main() {
     char command;
     char word[80];
     char meaning[200];
     Dictionary tree;
     do{
          help();
          command = getche();
          printf("\n");
          switch(command){
              case 'i': printf(" > 삽입 단어: "); gets(word);
                        printf(" > 단어 설명: "); gets(meaning);
                        tree.insert( new BinaryNode(word, meaning) );
                        break;
```

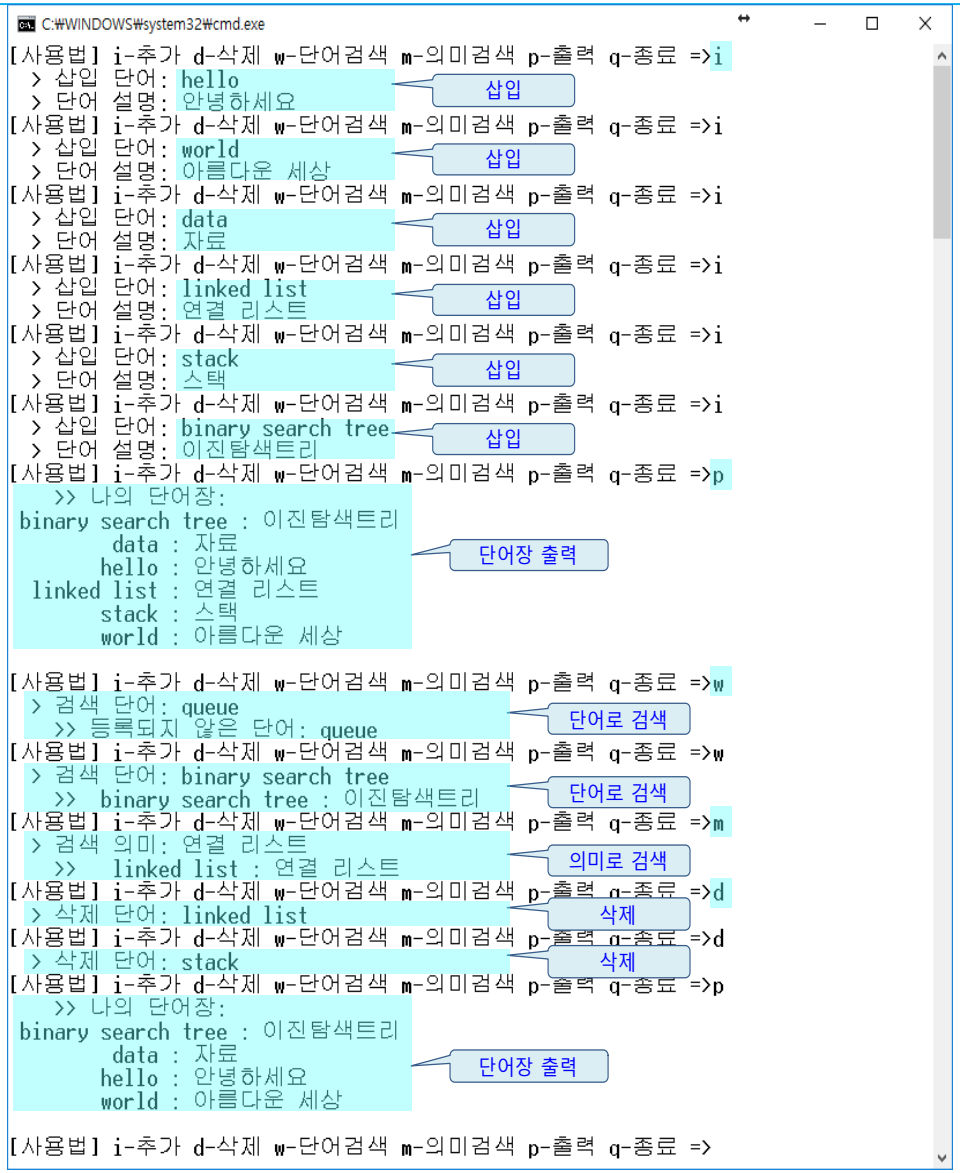
영어사전 프로그램



```
    case 'd': printf(" > 삭제 단어: "); gets(word);
              tree.remove( word );
              break;
    case 'p': tree.printAllWords( );
              printf("\n");
              break;
    case 'w': printf(" > 검색 단어: "); gets(word);
              tree.searchWord( word );
              break;
    case 'm': printf(" > 검색 의미: "); gets(word);
              tree.searchMeaning( word );
              break;
   }
  } while(command != 'q');
}
```

이진탐색트리 실행결과




C:₩WINDOWS₩system32₩cmd.exe
[사용법] i-추가 d-삭제 w-단어검색 m-의미검색 p-출력 q-종료 =>i
> 삽입 단어: hello
> 단어 설명: 안녕하세요
삽입
[사용법] i-추가 d-삭제 w-단어검색 m-의미검색 p-출력 q-종료 =>i
> 삽입 단어: world
> 단어 설명: 아름다운 세상
삽입
[사용법] i-추가 d-삭제 w-단어검색 m-의미검색 p-출력 q-종료 =>i
> 삽입 단어: data
> 단어 설명: 자료
삽입
[사용법] i-추가 d-삭제 w-단어검색 m-의미검색 p-출력 q-종료 =>i
> 삽입 단어: linked list
> 단어 설명: 연결 리스트
삽입
[사용법] i-추가 d-삭제 w-단어검색 m-의미검색 p-출력 q-종료 =>i
> 삽입 단어: stack
> 단어 설명: 스택
삽입
[사용법] i-추가 d-삭제 w-단어검색 m-의미검색 p-출력 q-종료 =>i
> 삽입 단어: binary search tree
> 단어 설명: 이진탐색트리
삽입
[사용법] i-추가 d-삭제 w-단어검색 m-의미검색 p-출력 q-종료 =>p
>> 나의 단어장:
binary search tree : 이진탐색트리
data : 자료
hello : 안녕하세요
linked list : 연결 리스트
stack : 스택
world : 아름다운 세상
단어장 출력
[사용법] i-추가 d-삭제 w-단어검색 m-의미검색 p-출력 q-종료 =>w
> 검색 단어: queue
>> 등록되지 않은 단어: queue
단어로 검색
[사용법] i-추가 d-삭제 w-단어검색 m-의미검색 p-출력 q-종료 =>w
> 검색 단어: binary search tree
>> binary search tree : 이진탐색트리
단어로 검색
[사용법] i-추가 d-삭제 w-단어검색 m-의미검색 p-출력 q-종료 =>m
> 검색 의미: 연결 리스트
>> linked list : 연결 리스트
의미로 검색
[사용법] i-추가 d-삭제 w-단어검색 m-의미검색 p-출력 q-종료 =>d
> 삭제 단어: linked list
삭제
[사용법] i-추가 d-삭제 w-단어검색 m-의미검색 p-출력 q-종료 =>d
> 삭제 단어: stack
삭제
[사용법] i-추가 d-삭제 w-단어검색 m-의미검색 p-출력 q-종료 =>p
>> 나의 단어장:
binary search tree : 이진탐색트리
data : 자료
hello : 안녕하세요
world : 아름다운 세상
단어장 출력
[사용법] i-추가 d-삭제 w-단어검색 m-의미검색 p-출력 q-종료 =>