8.1 트리의 개념

선형 자료구조 리스트, 스택, 큐

계층 자료구조 트리

ex)가계도, 조직도, 디렉토리 구조, 인공지능

트리의 용어들 노드 트리의 구성요소

트리 = 한 개 이상의 노드로 이루어진 유한 집합

루트 노드 최상위노드

서브 트리 나머지노드

간선 루트노드, 서브트리를 연결하는 선

관계 부모 루트는 서브의 부모

자식 서브는 루트의 자식

조상 루트노드에서 임의의 노드까지의 경로를 이루고 있는 노드들

후손 임의의 노드 하위에 연결된 모든 노드

단말노드 자식노드가 없는 노드

비단말노드 단말노드의 반대 = 자식노드가 있는 노드

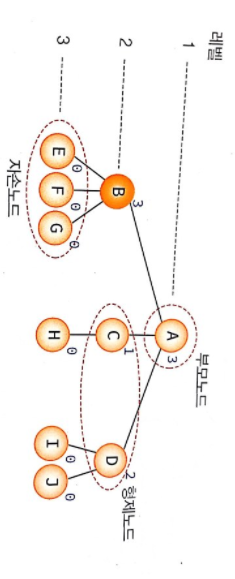
차수 어떤 노드가 가지고 있는 자식노드의 개수

트리의 차수 트리가 가지고 있는 노드의 차수중에서 제일 큰 값

레벨 트리의 각 층에 번호를 매기는 것

높이 트리가 가지고 있는 최대 레벨

포레스트 트리들의 집합



트리의 종류 일반트리 각 노드가 데이터를 저장하는 데이터필드와 자식노드를 가리키는 링크필드를 가지게 함

자식노드의 개수에 따라 노드의 크기가 달라짐

이진트리 자식노드의 개수가 2개임

Quiz 1. 단말노드

2. 차수

3. 50, 33, 14, 2, 4

8.2 이진 트리 소개

이진트리의 정의 모든 노드가 2개의 서브트리를 가지고 있는 트리

서브트리는 공집합일 수 있음

정의8.1 이진트리는 공집합이거나, 루트와 왼쪽 서브트리, 오른쪽 서브트리로 구성된 노드들의 유한 집합으로 정의된다. 이진트리의 서브트리들은 모두 이진트리여야 한다

🡪 이진트리의 서브트리도 이진트리의 성질을 만족해야 함

이진트리와 일반트리의 차이점 1. 이진트리의 모든 모드는 차수가 2이하 = 자식노드의 개수가 2이하 but 일반트리는 자식노드의 개수에 제한 X

2. 일반트리와는 달리 이진트리는 노드를 하나도 갖지 않을 수도 있음

3. 왼쪽 서브트리, 오른쪽 서브트리를 구분함 = 순서가 있음

이진트리의 성질 n개의 노드를 가진 이진트리 n-1개의 간선을 가짐

높이가 h인 이진트리 h~2h-1개의 노드를 가짐

n개의 노드를 가지는 이진트리 logn (n+1)(올림값)~n의 높이를 가짐

이진트리의 종류 포화 이진트리 트리의 각 레벨에 노드가 꽉 차있는 이진트리

높이 k인 포화 이진트리는 정확하게 2k-1개의 노드를 가짐

완전 이진트리 높이가 k일 때, 레벨 1부터 k-1까지는 노드가 모두 채워져있고, 마지막 레벨 k에서는 왼쪽부터 오른쪽으로 노드가 순서대로 채워져있는 이진트리

마지막 레벨에서는 노드가 꽉 차있지 않아도 되지만 중간에 빈 곳이 있으면 안됨

기타 이진트리

Quiz 1. e = n-1

2. 2h-1

3. log2(n+1) ~ n

8.3 이진 트리의 표현

배열을 이용하는 방법

포인터를 이용하는 방법

배열표현법 완전 이진트리라고 가정하고, 이진트리의 깊이가 k면 2k-1개의 공간을 연속적으로 할당

왼쪽부터 오른쪽까지 순서대로 노드들에 1부터 번호 부여 🡪 번호대로 배열 인덱스에 집어넣음

노드 i의 부모 노드 인덱스 i/2

노드 i의 왼쪽 자식 노드 인덱스 2i

노드 i의 오른쪽 자식 노드 인덱스 2i+1

링크표현법 노드가 구조체로 표현

노드가 포인터를 가지고 있음 🡪 노드끼리 연결

1개의 노드가 3개의 필드를 가짐

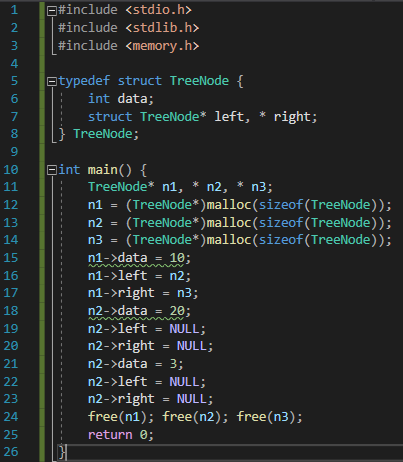
노드 구조체 typedef struct TreeNode {

int data;

struct TreeNode\* left, \* right;

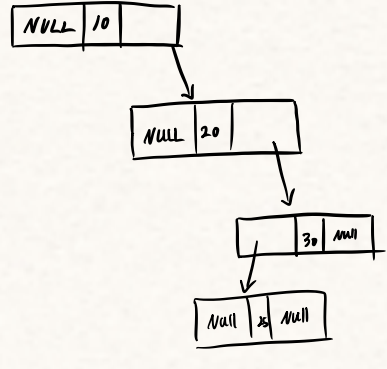
} TreeNode;

루트노드를 가리키는 포인터만 있으면 트리 안의 모든 노드들에 접근 가능

프로그램8-1 

Quiz 1. 인덱스 0 1 2 3 4 5 6 7 8 9 10 11 12

노드 10 20 30 25

2. 

8.4 이진 트리의 순회

순회 이진트리에 속하는 모든 노드들을 한 번씩 방문해 노드가 가지고 있는 데이터를 목적에 맞게 처리하는 것

트리는 선형 구조가 아님 🡪 여러가지 순서로 접근 가능

이진트리 순회방법 전위 루트를 서브트리에 앞서서 먼저 방문

중위 루트를 왼쪽과 오른쪽 서브트리 중간에 방문

후위 루트를 서브트리 방문 후에 방문

순환 알고리즘 사용

전위 순회 루트 🡪 왼쪽 서브트리 🡪 오른쪽 서브트리

알고리즘8.1 preorder(x):

if(x!=NULL) //NULL이면 더 이상 순환호출을 하지 않음

then print(DATA); //x의 데이터를 출력함

preorder(LEFT(x)); //x의 왼쪽 서브트리 순환호출

preorder(RIGHT(x)); //x의 오른쪽 서브트리 순환호출

중위 순회 왼쪽 서브트리 🡪 루트 🡪 오른쪽 서브트리

알고리즘8.2 inorder(x):

if (x != NULL) //NULL이면 더 이상 순환호출을 하지 않음

then inorder(LEFT(x)); //x의 왼쪽 서브트리를 순환호출

print(DATA(x)); //x의 데이터를 출력

inorder(RIGHT(x)); //x의 오른쪽 서브트리를 순환호출

후위 순회 왼쪽 서브트리🡪 오른쪽 서브트리 🡪 루트

알고리즘8.3 postorder(x) :

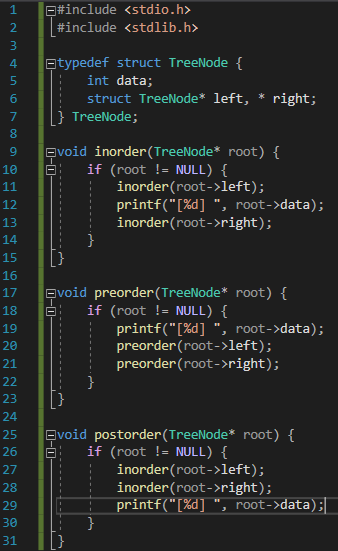
if (x != NULL) //x가 NULL이면 더 이상 순환호출을 하지 않음

then postorder(LEFT(x)); //x의 왼쪽 서브트리를 순환호출함

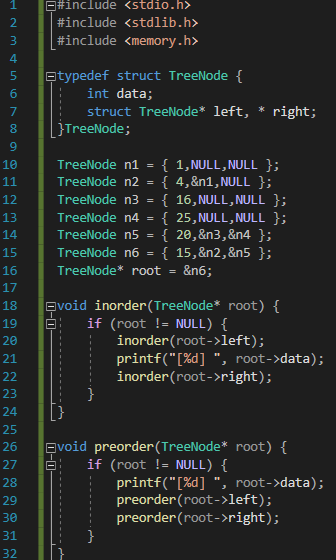
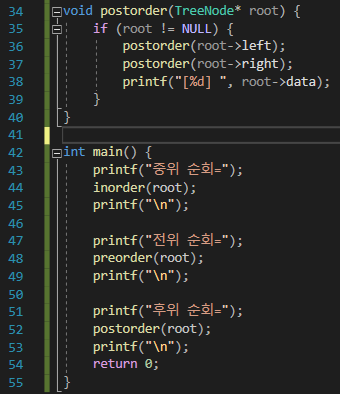
postorder(RIGHT(x)); //x의 오른쪽 서브트리를 순환호출함

print(DATA(x)); //x의 데이터를 출력

전위, 중위, 후위 순회 구현 순환 서브트리 방문 = 전체트리 방문 🡪 함수의 매개변수만 바꿔서 다시 호출 = 서브트리의 루트 노드 포인터를 매개변수로 다시 호출

프로그램8.2 

전체 프로그램 3가지 순회함수들의 전체적인 형태는 거의 동일 but if문 안의 문장들의 순서만 바뀜

프로그램8.3  

Quiz 1. 전위 17 🡪 15 🡪 05 🡪 93 🡪 35 🡪 22 🡪 95

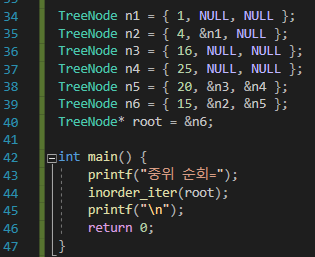
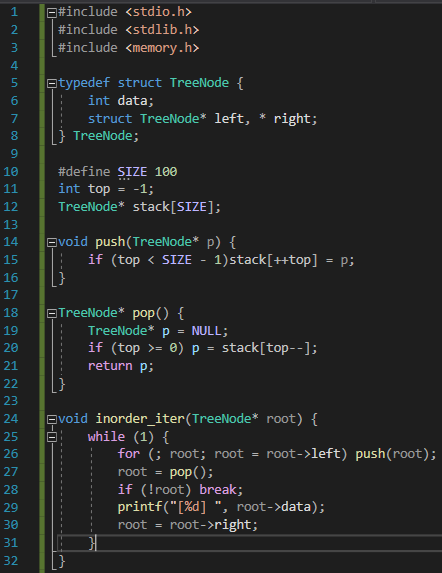
중위 05 🡪 15 🡪 17 🡪 22 🡪 35 🡪 93 🡪 95

후위 05 🡪 15 🡪 35 🡪 22 🡪 95 🡪 93 🡪 17

8.5 반복적 순회

반복을 이용한 트리 순회

별도의 스택 사용 시 가능

프로그램8.4 

8.6 레벨 순회

레벨 순회 각 노드를 레벨 순으로 검사하는 순회방법

동일한 레벨의 경우에는 좌에서 우로 방문

큐를 이용

코드 큐에 하나라도 노드가 있으면 계속 반복

1회반복 큐에 있는 노드를 꺼내 방문 🡪 자식노드를 큐에 삽입

알고리즘8.4 level\_order(root) :

initialize queue;

if(root == NULL) then return;

enqueue(queue, root);

while (is\_empty(queue) != true)

x = dequeue(queue);

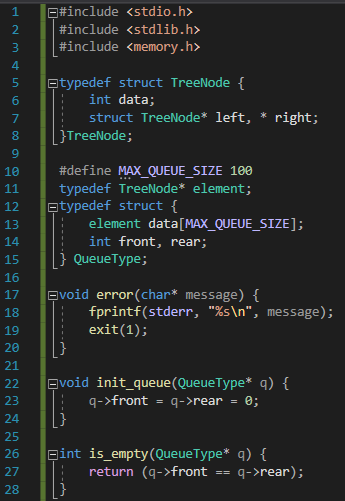
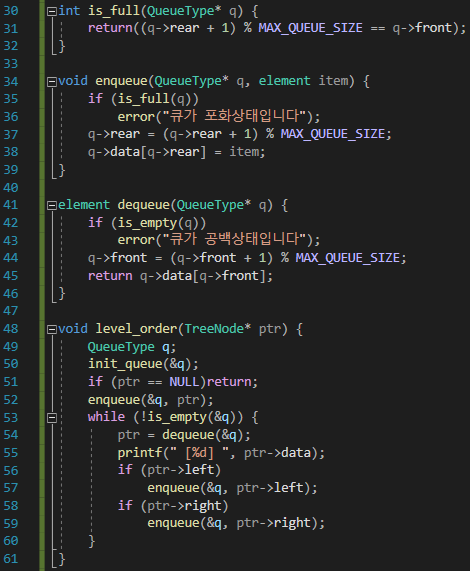
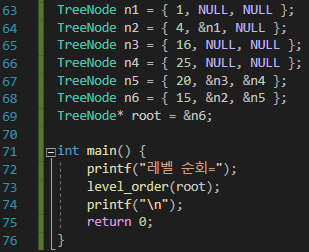
print(x->data);

if(x->left != NULL)

enqueue(queue, x->left)

if(x->right != NULL)

enqueue(queue, x->right)

프로그램8.5   

어떤 순회를 선택하여야 할까? 순서 중요 X 어떤 것이든 사용 가능

자식 🡪 부모 후위

부모 🡪 자식 전위

Quiz 1

2 3

4 5

8.7 트리의 응용: 수식 트리 처리

수식 트리 산술 연산자와 피연산자로 만들어짐

피연산자 단말노드

연산자 비단말노드

후위순회 사용

알고리즘8.5 evaluate(exp) :

if exp == NULL

then return 0;

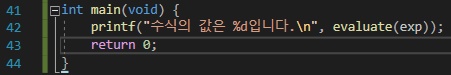
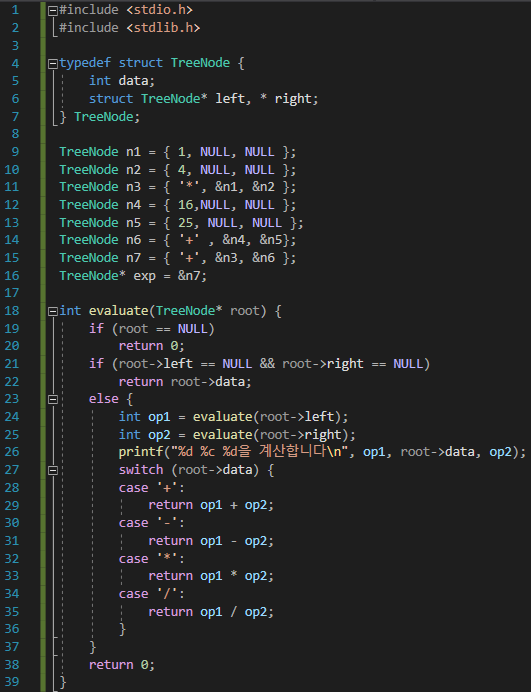
else

x = evaluate(exp.left);

y = evaluate(exp.right);

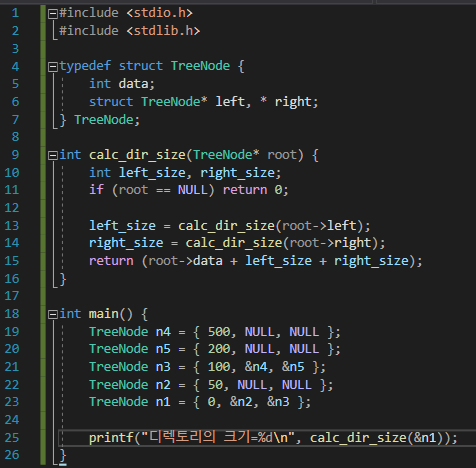
op = exp.data;

return (x op y);

프로그램8.6 

8.8 트리의 응용: 디렉터리 용량 계산

후위순회 서브 디렉터리 용량을 계산한 다음, 루트 디렉토리의 용량을 계산

프로그램 8.7 

8.9 이진트리의 추가 연산

노드의 개수 탐색 트리안의 노드의 개수를 세어 표시

각각의 서브트리에 대해 순환호출 🡪 반환되는 값에 +1

알고리즘8.6 get\_node\_count(x) :

if (x != NULL)

then return 1 + get\_node\_count(x.left) + get\_node\_count(x.right);

구현 int get\_node\_count(TreeNode\* node) {

int count = 0;

if (node != NULL)

count = 1 + get\_node\_count(node->left) + get\_node\_count(node->right);

return count;

}

단말노드 개수 구하기 탐색 트리안의 노드들을 순회해야 함

왼쪽자식과 오른쪽자식이 NULL일 경우 1을 반환

왼쪽자식과 오른쪽자식이 NULL이 아닐 경우 순환호출

알고리즘8.7 get\_leaf\_count(T) :

if (T != NULL)

then if (T.left == NULL and T.right == NULL)

then return 1;

else return get\_leaf\_count(T.left) + get\_leaf\_count(T.right);

구현 int get\_leaf\_count(TreeNode\* node) {

int count = 0;

if (node != NULL) {

if (node->left == NULL && node->right == NULL)

return 1;

else

count = get\_leaf\_count(node->left) + get\_leaf\_count(node->right);

}

return count;

}

높이 구하기 순환호출들의 반환값 중에서 최대값을 구해서 반환해야 함

알고리즘8.8 get\_height(T) :

if (T != NULL)

return 1 + max(get\_height(T.left), get\_height(T.right));

구현 int get\_height(TreeNode\* node) {

int height = 0;

if (node != NULL)

height = 1 + max(get\_height(node->left), get\_height(node->right));

return height;

}

Quiz 1. TreeNode n4 = { 500, NULL, NULL };

TreeNode n5 = { 200, NULL, NULL };

TreeNode n3 = { 100, &n4, &n5 };

TreeNode n2 = { 50, NULL, NULL };

TreeNode n1 = { 0, &n2, &n3 };

int main(void) {

printf("%d\n", get\_node\_count(&n1));

printf("%d\n", get\_leaf\_count(&n1));

printf("%d\n", get\_height(&n1));

return 0;

}

2. int get\_nonleaf\_count(TreeNode\* node) {

int count = 0;

if (node != NULL) {

if (node->left != NULL && node->right != NULL)

count = 1 + get\_nonleaf\_count(node->left) + get\_nonleaf\_count(node->right);

else

return 0;

}

return count;

}

3. int equal(TreeNode\* t1, TreeNode\* t2) {

if (t1 != NULL && t2 != NULL) {

if (t1->data == t2->data) {

if (!equal(t1->left, t2->left)) return 0;

if (!equal(t1->right, t2->right)) return 0;

return 1;

}

else {

return 0;

}

}

else if (t1 == NULL && t2 == NULL) {

return 1;

}

else {

return 0;

}

}

8.10 스레드 이진 트리

순회를 호출없이 하기

스레드 이진 트리 중위 선행자 중위순회 시의 선행 노드

중위 후속자 중위순회 시의 후속 노드

NULL링크에 중위 선행자나 중위 후속자를 저장시켜 놓은 트리

실을 이용해 노드들을 순회 순서대로 연결시켜 놓은 트리

노드의 구조 typedef struct TreeNode {

int data;

struct TreeNode\* left, \* right;

int is\_thread; //true이면 right는 중위후속자, false면 right는 오른쪽자식

}TreeNode;

중위순회 연산 중위 후속자를 반횐하는 함수 TreeNode\* find\_successor(TreeNode\* p) {

TreeNode\* q = p->right;

if (q == NULL || p->is\_thread == 1)

return q;

while (q->left != NULL)q = q->left;

return q;

}

중위순회를 하는 함수 void thread\_inorder(TreeNode\* t) {

TreeNode\* q;

q = t;

while (q->left) q = q->left;

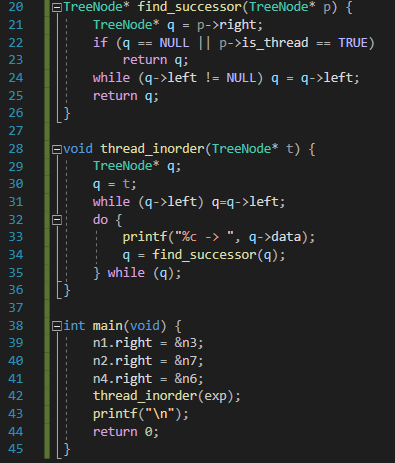
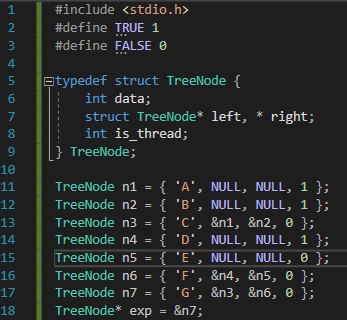
do {

printf("%c -> ", q->data);

q = find\_succssor(q);

} while (q);

}

프로그램8.8 

Quiz 1. n+1

2. 중위후속자

8.11 이진 탐색 트리

이진 탐색 트리 이진 트리 기반의 탐색을 위한 자료구조

탐색 레코드의 집합에서 특정한 레코드를 찾아내는 작업

레코드 하나 이상의 필드로 구성됨

테이블 레코드들의 집합

키 레코드를 식별하는 하나의 필드

주요 키 다른 키와 중복되지 않는 고유한 값

이진 탐색 트리의 정의 정의8.2 1. 모든 원소의 키는 유일한 키를 가진다.

2. 왼쪽 서브 트리 키들은 루트 키보다 작다.

3. 오른쪽 서브 트리의 키들은 루트의 키보다 크다.

4. 왼쪽과 오른쪽 서브 트리도 이진 탐색 트리이다.

이진트리의 루트노드의 키값과 비교 🡪 루트노드보다 작음 = 원하는 키값은 왼쪽 서브 트리에 있음

🡪 루트노드보다 큼 = 원하는 키값은 오른쪽 서브 트리에 있음

순환적인 탐색 연산 탐색키값과 루트노드의 키값 비교 🡪 비교한 결과가 같음 🡪 탐색 끝

🡪 탐색키값 < 루트노드의 키값 🡪 루트노드의 왼쪽 자식으로 다시 탐색

🡪 탐색키값 > 루트노드의 키값 🡪 루트노드의 오른쪽 자식으로 다시 탐색

알고리즘8.9 search(root, key) :

if (root == NULL)

then return NULL;

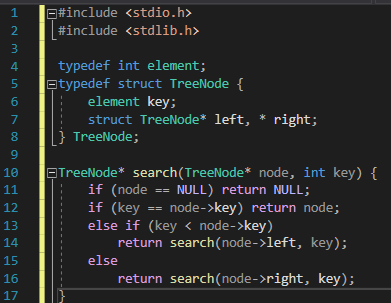
if (key == KEY(root))

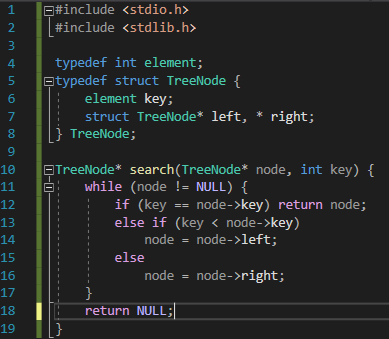
then return root;

else if (key<KEY(root))

then return search(LEFT(root), k);

else return search(RIGHT(root), k);

프로그램8.9 

반복적인 탐색 프로그램8.10 

이진탐색트리에서 삽입연산 탐색을 먼저 수행 같은 키값을 갖는 노드가 없어야 함

탐색에 실패한 위치가 새로운 노드를 삽입하는 위치가 됨

알고리즘8.2 insert(root, n) :

if (KEY(n) == KEY(root))

then return;

else if (KEY(n) < KEY(root))

if (LEFT(root) == NULL)

then LEFT(root) = n;

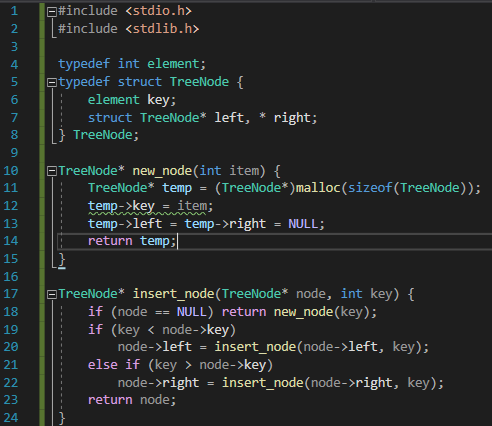
else insert(LEFT(root), n);

else

if (RIGHT(root) == NULL)

then RIGHT(root) = n;

else insert(RIGHT(root), n);

프로그램8.11 

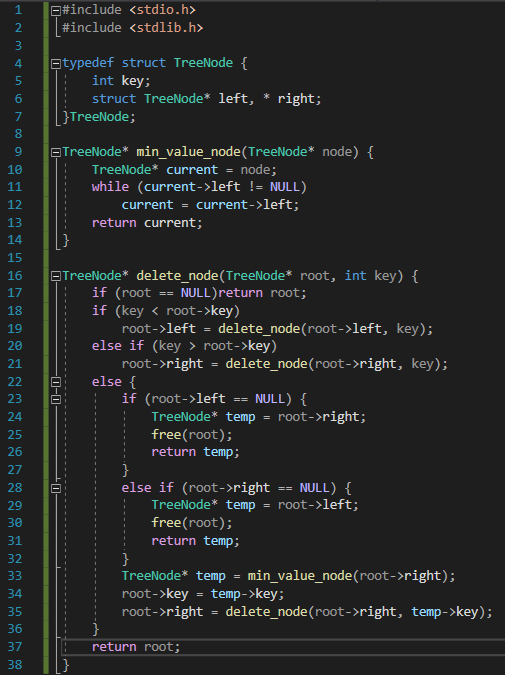
이진탐색트리에서 삭제연산 탐색을 먼저 수행

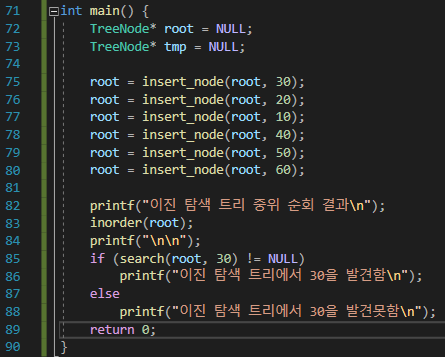
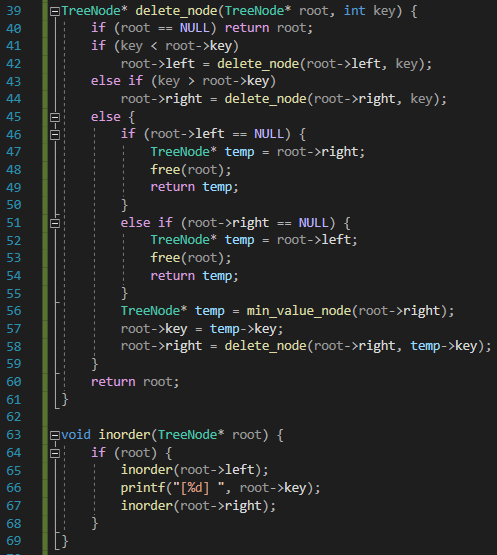
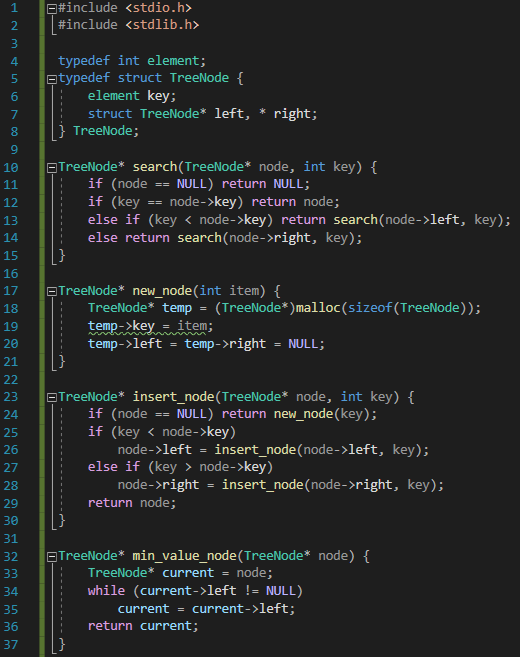
1. 삭제노드가 단말노드일 경우 가장 쉬움

부모노드의 링크필드를 NULL로 만듬

2. 삭제하려는 노드가 하나의 서브트리만 가지고 있는 경우 자기노드만 삭제하고 서브트리는 자기노드의 부모노드에 붙여주면 됨

3. 삭제하려는 노드가 두개의 서브트리를 가지고 있는 경우 가장 값이 가까운 노드를 후계자노드로 선택 = 왼쪽 서브트리에서 가장 큰 값 or 오른쪽 서브트리에서 가장 작은 값

프로그램8.12 

전체프로그램 프로그램8.13 

이진탐색트리의 분석 탐색, 삽입, 삭제 연산의 시간 복잡도 트리의 높이가 h일 때, O(h) = 노드의 개수가 n일 때, O(log2n)

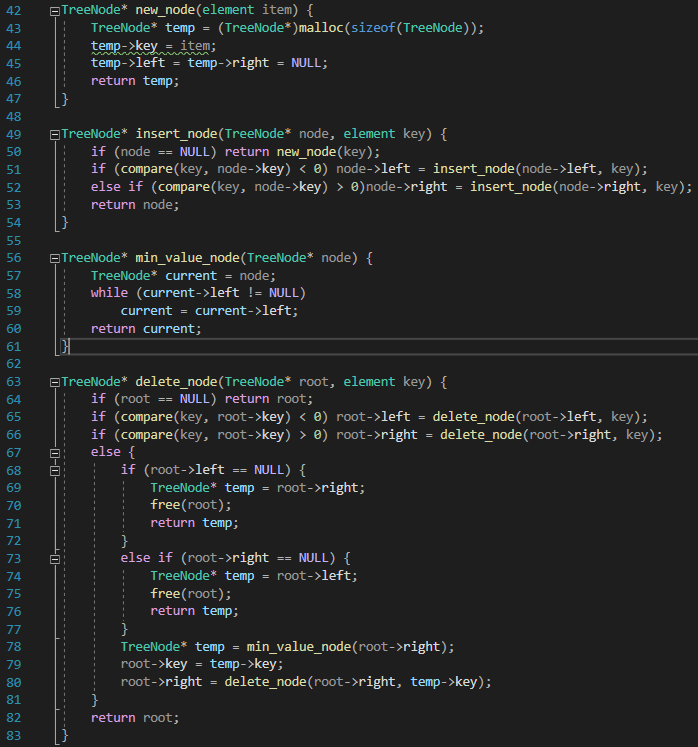
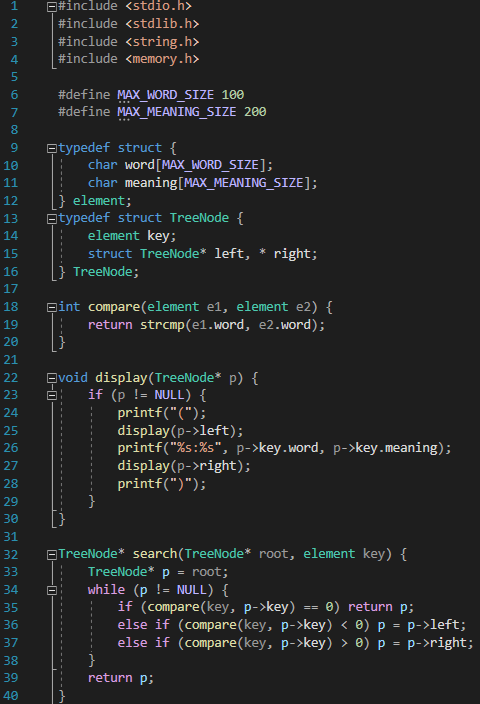
Quiz 1. 7

2. 17 🡪 14 🡪 26 🡪 02, 66 🡪 28 🡪 80 🡪 34

3. 14가 루트 노드가 됨

8.12 이진 탐색 트리의 응용: 영어 사전

입력, 탐색, 삭제

프로그램8.14 

연습문제

1. 4
2. 2
3. 4
4. 3
5. 1
6. 3
7. 4
8. 3
9. O(n)O(log­­ n)

(1) 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

6 4 9 2 5 7 10 1 3 8 11

(2) 6 4 2 1 3 5 9 7 10 8 11

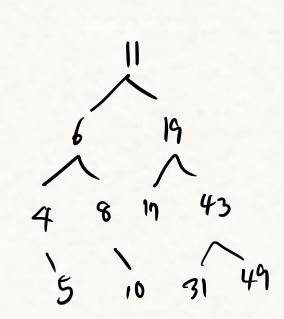
(3) 1 3 2 5 4 7 8 11 10 9 6

(4) 1 2 3 4 5 6 7 9 8 10 11

(5) 6 4 9 2 5 7 10 1 3 8 11

(6) 이진트리가 아님, 9>8

11. (1)



(2) 17이 루트노드가 됨

(3) 17의 자식노드가 됨

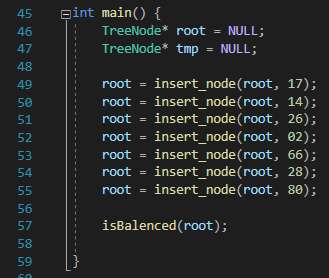
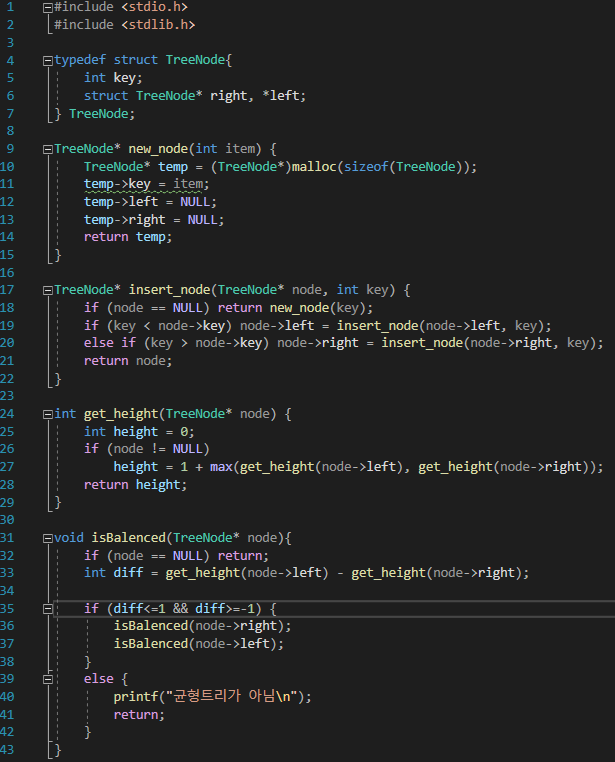
(4) 11 6 8

(5) 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

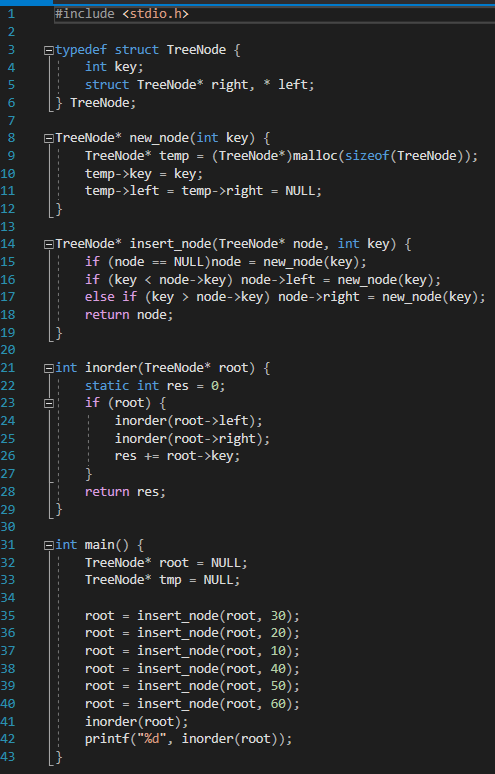
11 6 19 4 8 17 43 5 10 31 49

12. 8

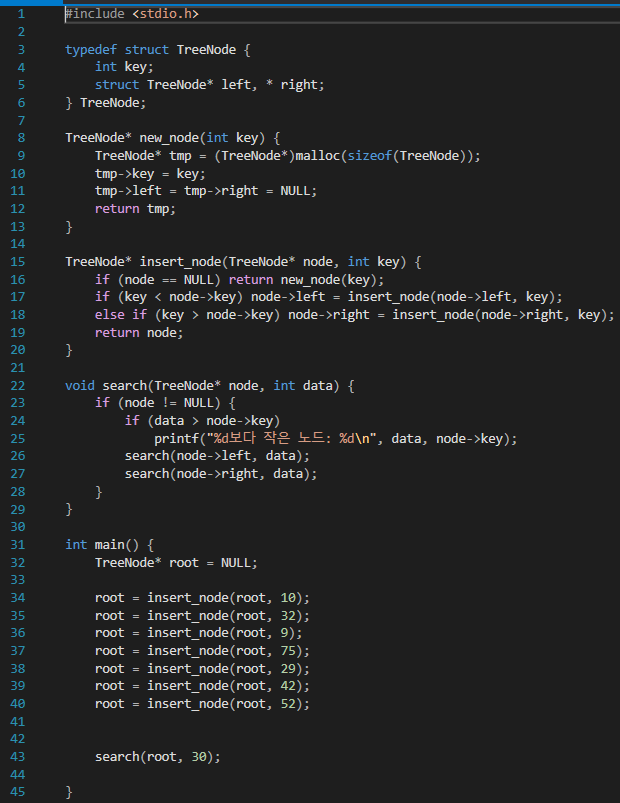
13.



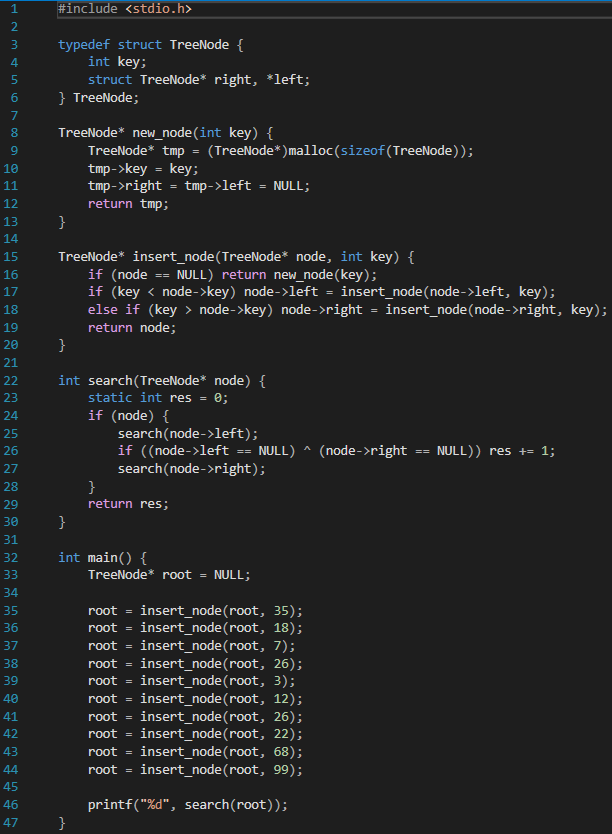
14.



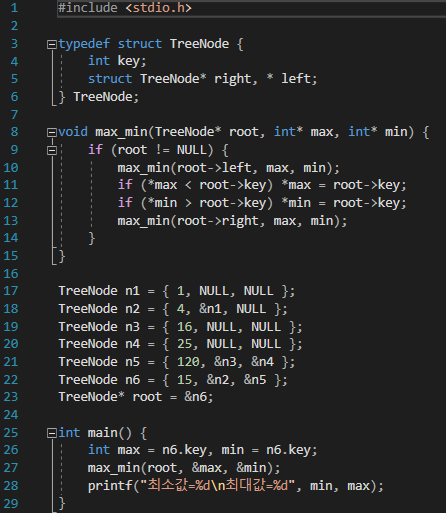
15.



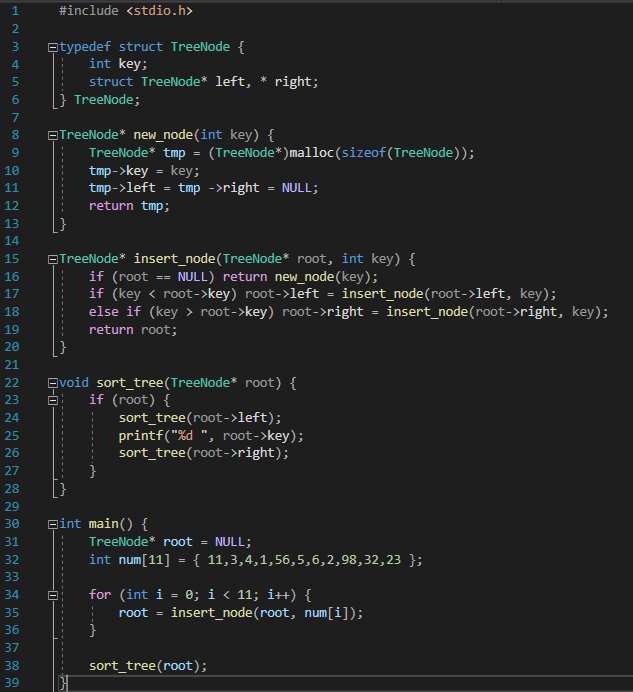
16.



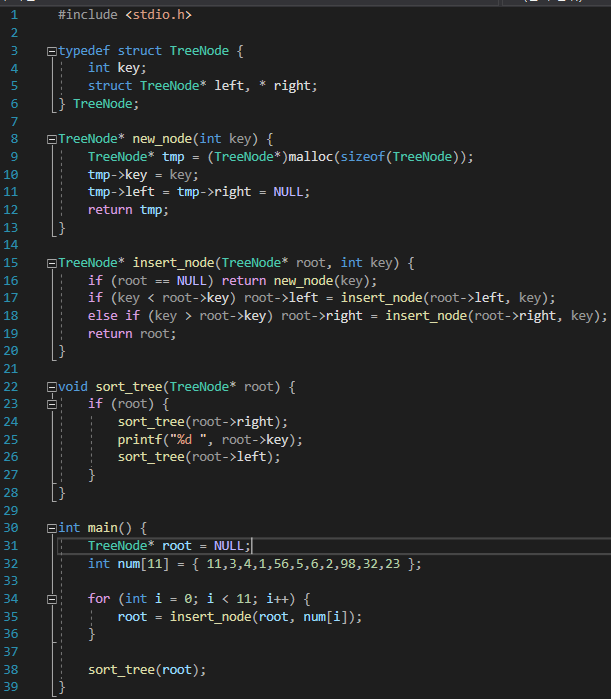
17.



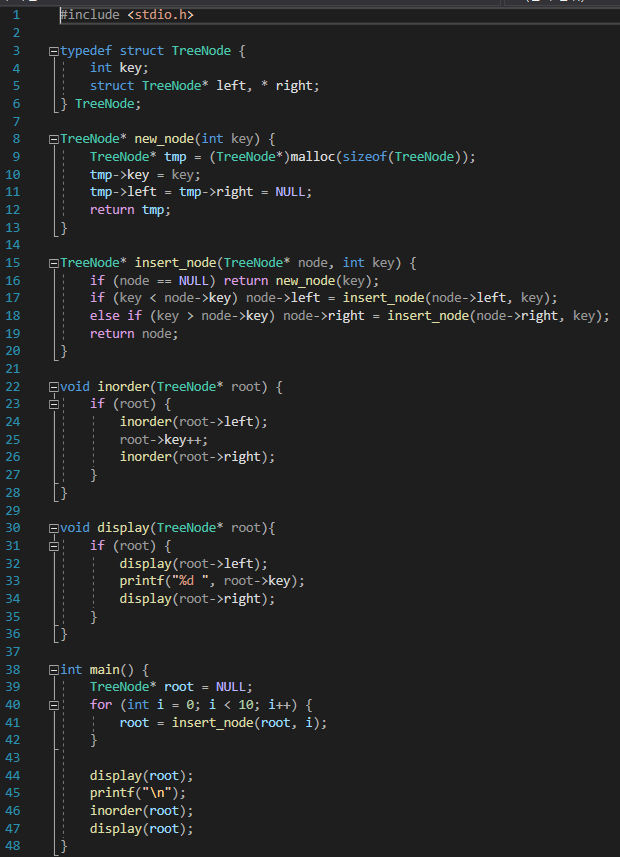
18.



19.



20.



21. 트리의 오른쪽 루트노드에 도달하면 그 값이 가장 큰 값

22.

## 