6. 트리(Trees)

한국항공대학교 안준선

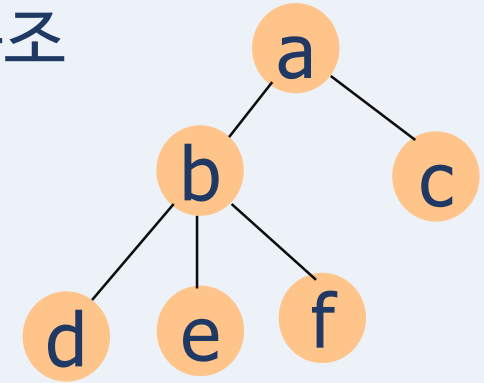
개요

- 트리와 이진나무
- 이진나무 구조의 구현
- 이진나무 순회
- 수식 트리

트리 구조

- 정의
  - 노드(node, vertex)와 (노드를 연결하는) 링크(link, edge)의 집합
  - 루트 노드와 0개 이상의 겹치거나 연결되지 않은(disjoint) 하위 *나무 구조* (subtrees)로 구성
- 특징
  - 노드에는 정보가 저장되며 링크는 정보간의 관계를 나타냄
  - 한 노드 밑에 다수의 하위 노드가 위치하는 2차원 구조를 가진다.
  - 주어진 노드에 대한 자식/부모 노드의 직접 접근 연산을 제공

예) 디렉토리(폴더) 구조



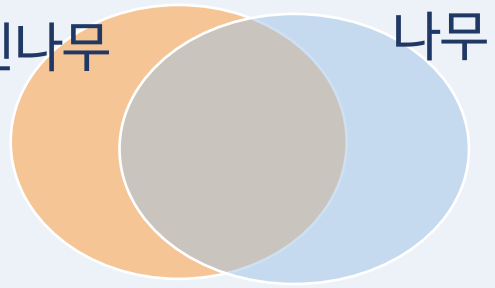
tree의 용어들

- 경로(path) : 링크에 의하여 연결된 일련의 노드들의 열(sequence)
  - $(n_1, n_2, n_3, \cdots, n_m)$ where $m>0$ and n_i and n_{i+1} is connected by a link.
- 루트(root) 노드 : 최상위의 노드

  [성질] 루트 노드에서 임의 노드로 중첩되지 않는 path는 언제나 하나만 존재한다.
- 부모(parent), 자식(children), 형제(sibling), 조부모(grandparent), 조상(ancestor)
- 잎(leaf) : 자식이 없는 노드, terminal node, 외부 노드(external node)
- 부트리(subtree) : 큰 나무에 속한 작은 나무
- 노드의 degree : 하위 서브트리의 개수
- 노드의 레벨(level) : 뿌리 노드에서 해당 노드까지의 중첩되지 않는 경로의 노드 수
- 나무의 높이 : 나무 내 노드의 레벨들 중 가장 큰 수
- 나무의 경로의 길이 : 각 노드들의 레벨의 총합

이진 나무 (Binary Tree)

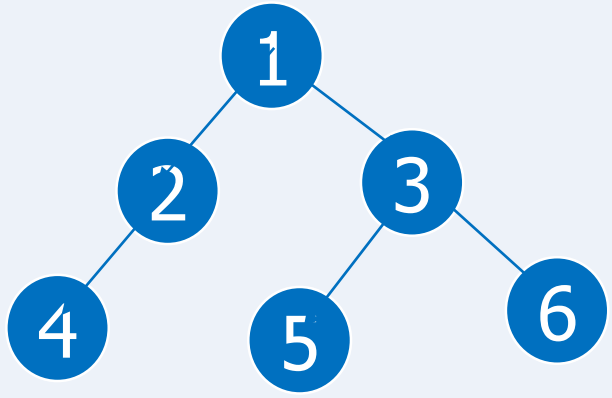
- 이진 나무 :
  - 정의 : 노드가 하나도 없는 공집합이거나, 루트 노드와 왼쪽 이진나무, 오른쪽 이진 나무로 이루어진 노드들의 집합
  - 이진 노드의 경우 자식 노드들의 위치 관계를 고려한다 : left child, right child
  - 완전한 이진 나무(complete binary tree) : 가장 마지막 레벨을 제외하고는 모든 노드들이 꽉 차 있고, 마지막 레벨은 제일 왼쪽부터 마지막 노드까지 빈칸이 없는 트리
  - 꽉 차있는 이진 나무(full binary tree) : 마지막 레벨이 꽉 차있는 트리
- 나무의 성질
  - 높이가 d인 full binary tree의 노드의 갯수 : $2^0 + 2^1 + 2^{(d-1)} = 2^d - 1$
  - N개의 노드를 가진 complete binary tree의 높이는 $\lfloor \log_2 N \rfloor + 1$ 이다.
  - 임의의 두 노드는 최소 공통 선조(least common ancestor)를 갖는다.
  - n개의 노드를 갖는 나무는 (n-1)개의 링크를 갖는다.
  - complete binary tree의 breadth first numbering의 경우에 부모의 번호가 n일 경우에 자식의 번호는 2n, 2n+1 이다.



이진 나무의 구현

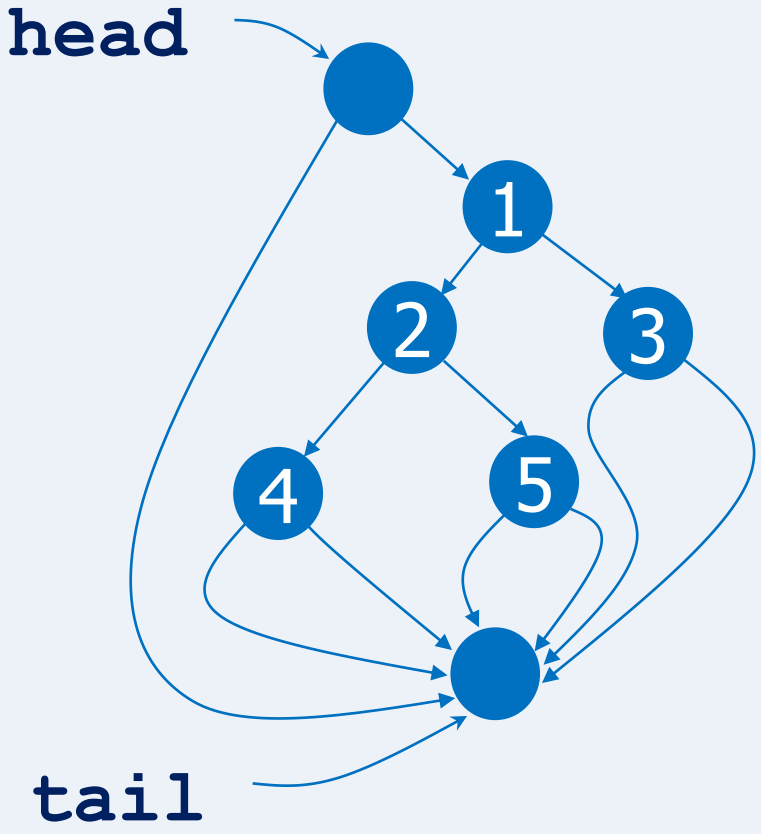
- 배열을 이용한 구현
  - 자식 노드의 breadth first 번호가 complete binary tree에서 2n, 2n+1 로 번호 매겨짐을 이용하여 breadth first 순서로 저장한다.
  - 중간에 비어있는 노드에 해당하는 자리는 비워둔다.
  - 자식 원소와 부모 원소를 *2 또는 /2로 직접 접근 가능
  - complete binary tree가 아닌 경우에 메모리 낭비가 심하다.
  - 미리 노드의 최대 개수가 정해진다.

0	1	2	3	4	5	6	7
	1	**2**	**3**	**4**		**5**	**6**



이진 나무의 구현

- 연결 리스트를 사용한 이진 나무 구조의 표현




```
typedef struct _node {
    int key;
    struct _node *left;
    struct _node *right;
} node;

node *head, *tail;
```

이진 나무의 구현

- 트리의 초기화 : 아무 원소도 가지고 있지 않은 트리 생성

```
void init_tree(void)
{
    head = (node*)malloc(sizeof(node));
    tail = (node*)malloc(sizeof(node));
    head->left = tail;
    head->right = tail;
    tail->left = tail;
    tail->right = tail;
}
```

- 나무 구조에 대한 동작
  - 주어진 노드의 자식/부모 노드의 직접 접근 제공
  - 나무 구조에 대한 노드의 삽입 및 삭제, 탐색 등

이진 나무의 구현

- 트리 만들기 예시

```
node* make_node (int v, node* l, node* r) {
     node* n = (node*)malloc(sizeof(node));
     n->left = l; n->right = r; n->key = v;
     return n;
}
main() {
    init_tree();
    head->right = make_node(1,tail,tail);
        …
}
```

이진 나무의 순회

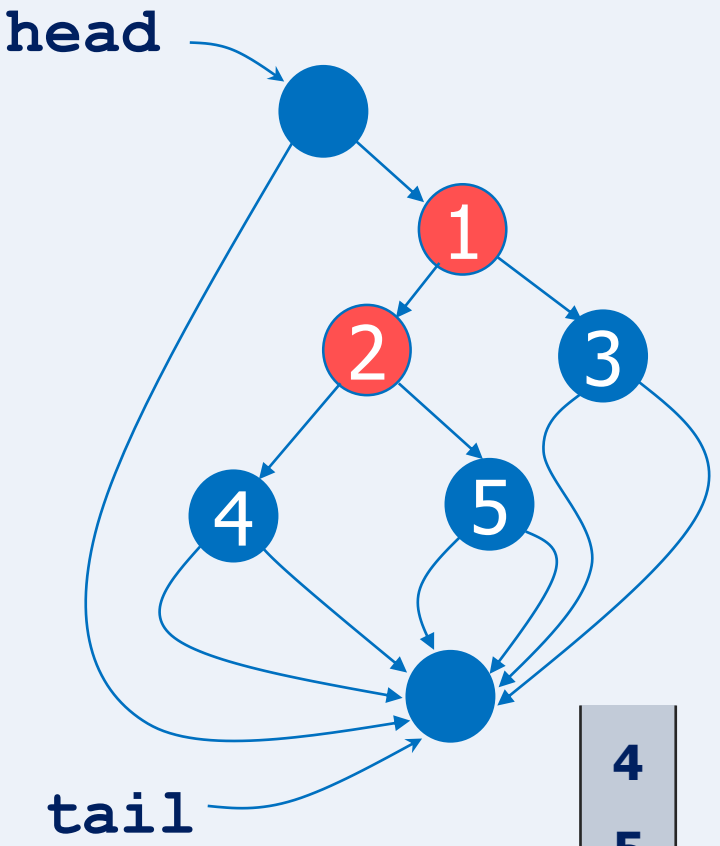
- 이진 나무 순회 (tree traverse)
  - tree traverse : 나무의 모든 노드들을 한 번씩 중복 없이 방문
    - 전위 순회 (Preorder traverse) : 뿌리 먼저 방문하고 왼쪽, 오른쪽 하위트리 전위 순회
    - 중위 순회 (Inorder traverse) : 왼쪽 하위 트리 중위순회 후 뿌리 방문 오른쪽 중위순회
    - 후위 순회 (Postorder traverse) : 하위 트리 모두 후위순회 한 후에 뿌리 방문
    - 층별 순회 (Level order traverse) : 위쪽 노드들부터 순서대로 방문
  - 전위/중위/후위 순회는 재귀 함수에 의하여 쉽게 구현 (층별 순회 : 큐 사용)
  - 전위 순회 방법
    1. 뿌리를 방문한다.
    2. 왼쪽 부분 트리(subtree)를 전위 순회로 방문한다.
    3. 오른쪽 부분 트리를 전위 순회로 방문한다.

이진 나무의 순회

- 전위 순회 함수

```
void preorder_traverse(node *t) {
    if (t != tail) {
        visit(t);
        preorder_traverse(t->left);
        preorder_traverse(t->right);
    }
}
```

```
/* 비재귀적 전위 순회 함수 */
void iterative_preorder_traverse (node *t) {
   init_stack(); push(t);
   while ((t=pop()) != NULL) {
        visit(t);
        if (t->right != tail) push(t->right);
        if (t->left != tail) push(t->left);
   }
}
```



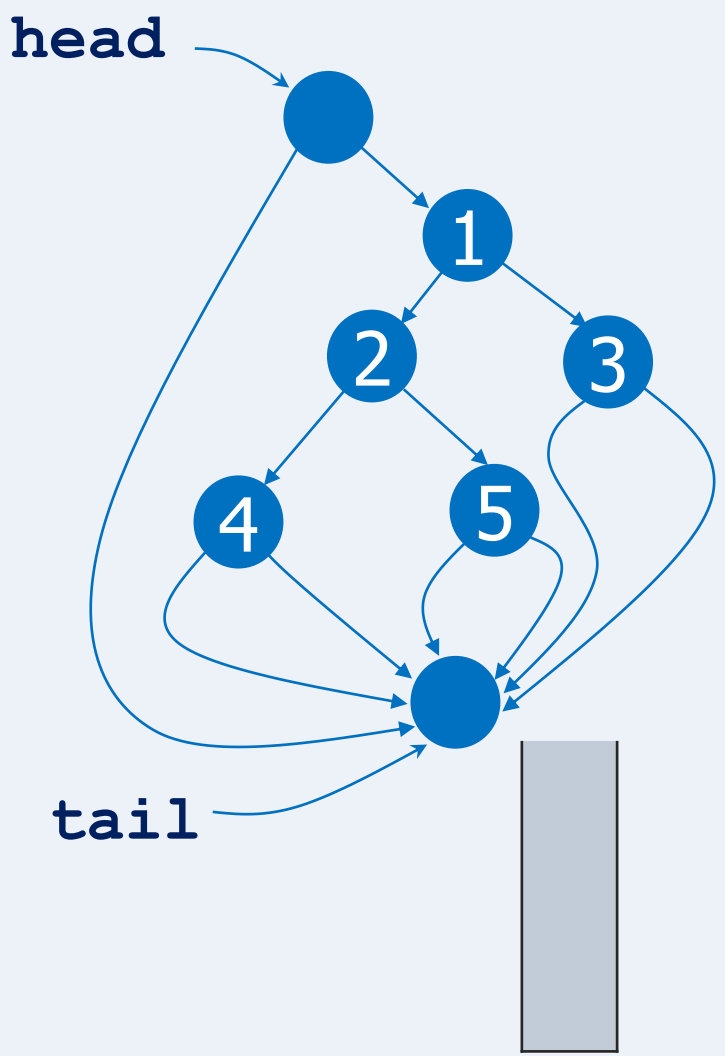
이진 나무의 순회

- 중위 순회
  - 순회 방법
    1. 왼쪽 작은 나무를 방문한다.
    2. 뿌리를 방문한다.
    3. 오른쪽 작은 나무를 방문한다.
  - 중위 순회 함수

```
void inorder_traverse(node *t) {
    if (t != tail) {
        inorder_traverse(t->left);
        visit(t);
        inorder_traverse(t->right);
    }
}
```

이진 나무의 순회

```
/* 비재귀적 중위 순회 함수 */
void iterative_inorder_traverse (node *t) {
    init_stack();
    while (t != tail) {
        push(t);
        t = t->left;
    }
    while ((t=pop()) != NULL) {
        visit(t);
        t = t->right;
        while (t != tail) {
            push(t);
            t = t->left;
        }
    }
}
```



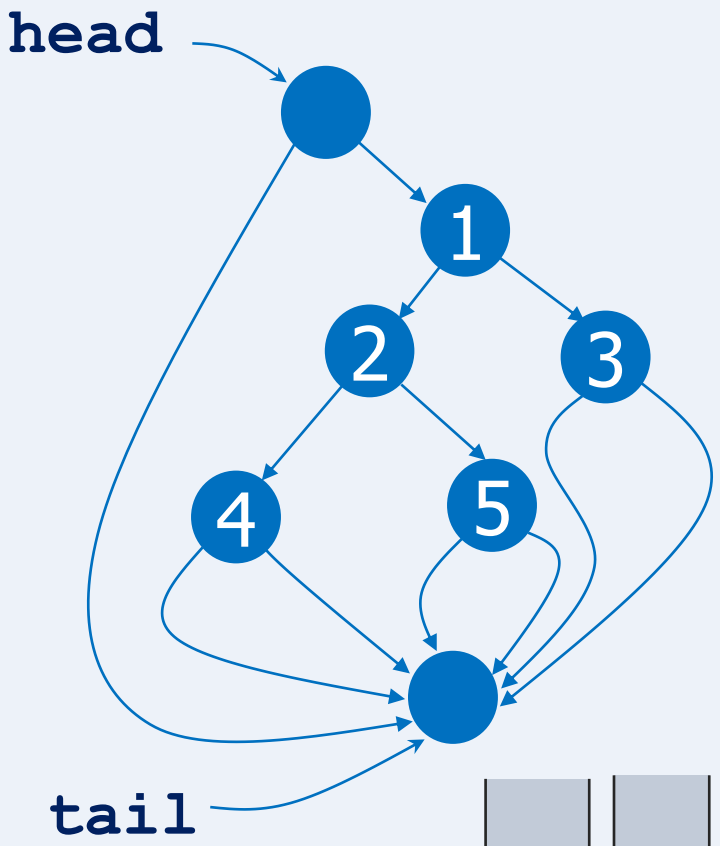
이진 나무의 순회

- 후위 순회 방법
  - 순회 방법 : 전위와 반대로 뿌리를 가장 나중에 방문
  - 순회 방법
    1. 왼쪽 부분 트리(subtree)를 후위 순회로 방문한다.
    2. 오른쪽 부분 트리를 후위 순회로 방문한다
    3. 뿌리를 방문한다.
  - 후위 순회 함수

```
void postorder_traverse(node *t) {
    if (t != tail) {
        postord`er_traverse(t->left);
        postorder_traverse(t->right);
        visit(t);
    }
}
```

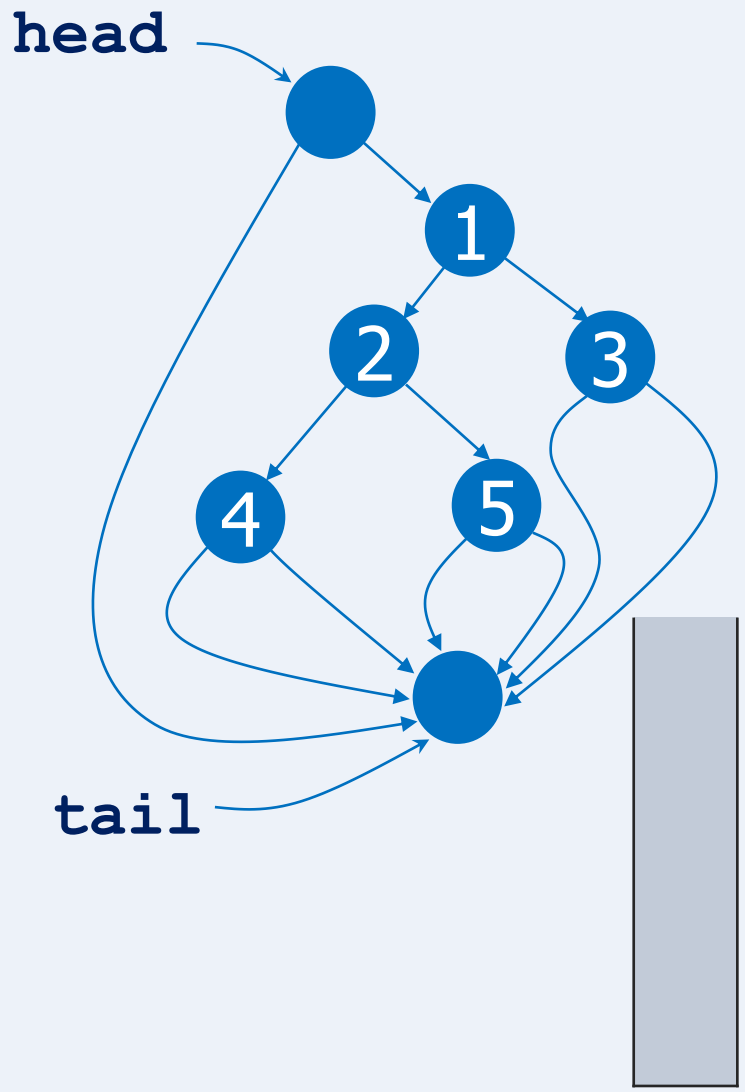
이진 나무의 순회

```
// 비재귀적 후위 순회 함수
void iterative_postorder_traverse(node *t) {
    init_stack1();
    init_stack2();

    push1(t);
    while ((t=pop1()) != NULL) {
        push2(t);
        if (t->left != tail) push1(t->left);
        if (t->right != tail) push1(t->right);
    }

    while ((t=pop2()) != NULL) visit(t);
}
```



이진 나무의 순회

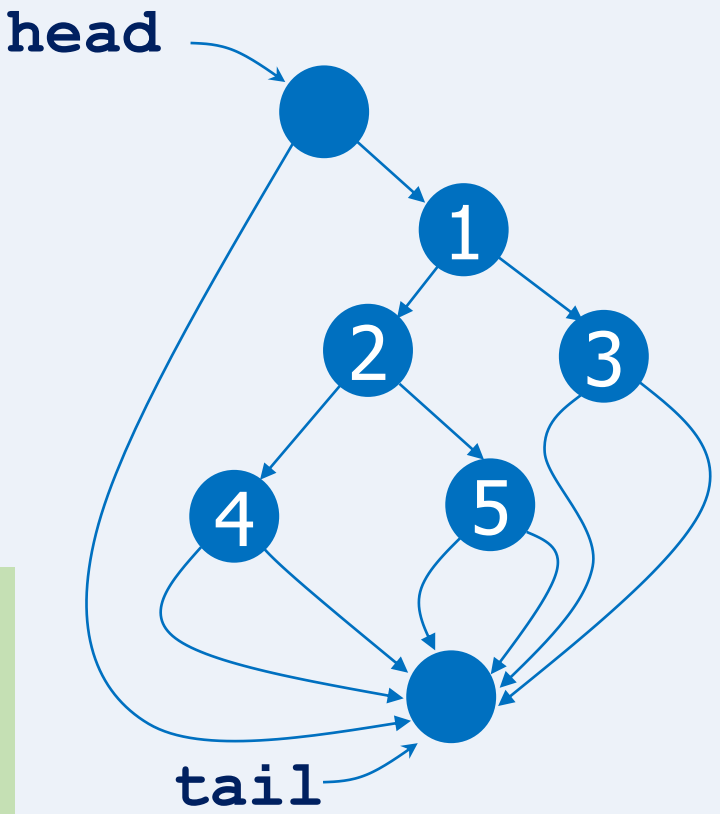
```
// 비재귀적 후위 순회 함수 #2
void iterative_postorder_traverse(node *n) {
    int tag;
    node *t = n;
    init_stack();
    push(t); push(1);
    while ((tag = pop())!= NULL) {
        t = pop();
        switch(tag) {
        case 1 :
            if (t == tail) break;
            push(t); push(2);
            push(t->right); push(1);
            push(t->left); push(1);
            break;
        case 2 :
            visit(t);
        }
    }
}
```



이진 나무의 순회

- 층별로 방문하기 : 위의 노드들부터 순서대로 방문한다.
  - 알고리즘 : 큐를 사용한다.
    1. 뿌리노드를 큐에 put 한다.
    2. 큐가 비어 있지 않은 동안에
       1. 큐에서 get한 노드 t를 방문한다.
       2. t의 왼쪽 자식 노드를 큐에 put 한다.
       3. t의 오른쪽 자식 노드를 큐에 put 한다.
  - 층별 순회 함수

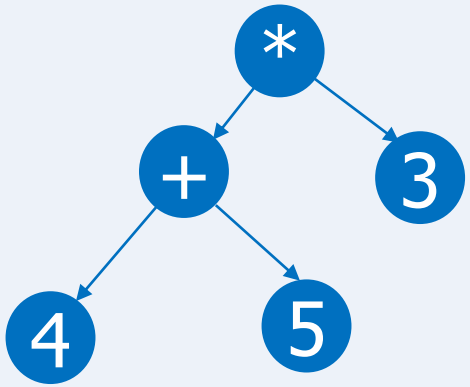
```
void levelorder_traverse(node *t) {
    put(t);
    while (!is_queue_empty()) {
        t = get();
        visit(t);
        if (t->left != tail) put(t->left);
        if (t->right != tail) put(t->right);
    }
}
```



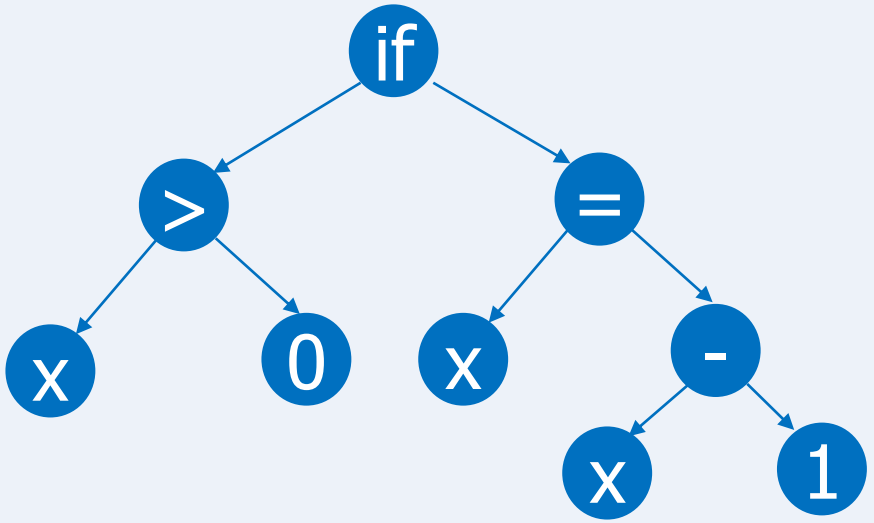
수식 나무

- 수식 나무(Parse Tree)
  - 수식(:문자열)을 그 문법적 구조에 따라 트리 형태로 표현한 것
  - 수식의 계산이나 수식의 표기법의 변환에 유용
  - 연산자를 저장한 루트노드와 피연산자를 나타내는 **수식나무**인 양쪽 서브트리의 형태 또는 노드 하나인 피연산자의 형태를 가짐

(4 + 5) * 3



if (x>0) x=x-1



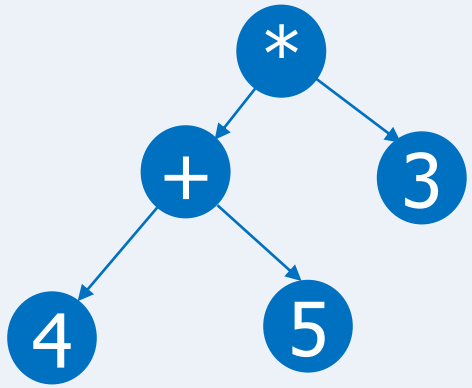
후위 표기법에서 수식 나무 만들기

- 알고리즘

  : 수식의 끝까지 하나씩 토큰을 읽으면서 다음을 수행

  - 피연산자를 만나면 노드를 생성해서 스택에 푸쉬한다.
  - 연산자를 만나면 해당 연산자를 위한 노드를 만들고,
    1. 스택에서 팝한 노드를 오른쪽 자식 노드로 할당
    2. 스택에서 팝한 노드를 왼쪽 자식 노드로 할당
    3. 연산자 노드를 스택에 푸쉬

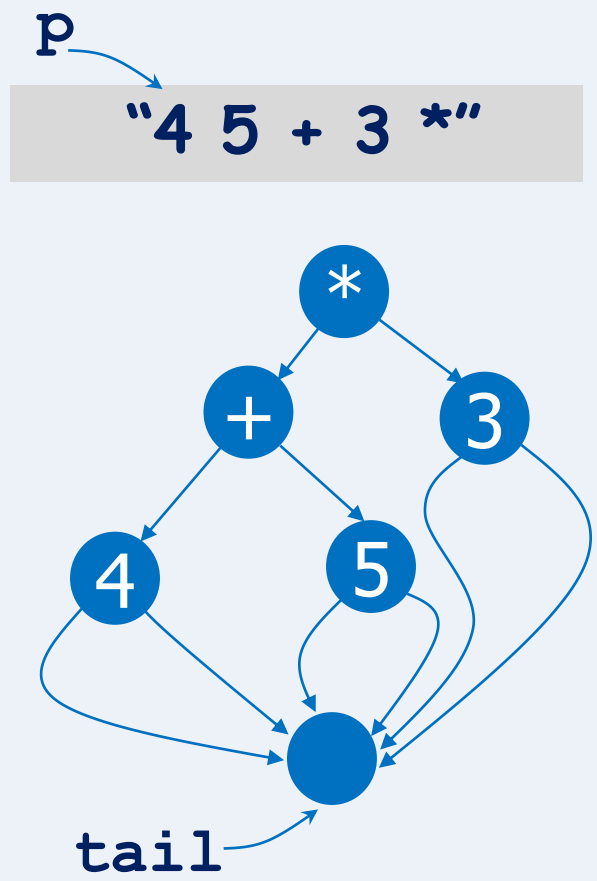
```
4 5 + 3 *
```



수식 나무

- 수식 나무 생성 프로그램

```
node *make_parse_tree(char *p) {
    node *t;
    while (*p) {
        while (*p == ' ') p++;
        t = (node*)malloc(sizeof(node));
        t->key = *p;
        t->left = tail;
        t->right = tail;
        if (is_operator(*p)) {
            t->right = pop();
            t->left = pop();
        }
        push(t);
        p++;
    }
    return pop();
} // 피연산자는 하나의 문자/숫자로 가정
```



수식 나무

- 후위 표기식의 적법성 검사 프로그램

```
int is_legal(char *s) {
    int f = 0;
    while (*s) {
        while (*s == ' ') s++;
        if (is_operator(*s))
            f--;
        else
            f++;
        if (f < 1) /* check situation like A+B */
            break;
        s++;
    }
    return (f == 1); /* legal if operand-operator==1 */
}
```

수식 나무

- main 함수

```
void main(void)  {
    char post[256];
    init_stack();
    init_queue();
    init_tree();
    while (1) {
       printf("\n\nInput Postfix expression -> ");
       gets(post);
       if (*post == NULL) {
           printf("\n Program ends...");
           exit(0);
       }
       if (!is_legal(post)) {
           printf("\nExpression is not legal.");
           continue;
       }
```

나무 구조의 응용

```
            head->right = make_parse_tree(post);

            printf("\nPreorder   traverse -> ");
            preorder_traverse(head->right);

            printf("\nInorder    traverse -> ");
            inorder_traverse(head->right);

            printf("\nPostorder  traverse -> ");
            postorder_traverse(head->right);

            printf("\nLevelorder traverse -> ");
            levelorder_traverse(head->right);
        }
    }
```

정리

- 트리 자료구조
- 트리 자료구조의 구현 (배열과 포인터 연결 구조)
- 트리 순회
- 수식 나무