자료구조 (Data Structure)
5주차: 트리

선형 (일차원) 자료구조 정리

- 리스트: 데이터를 추가하면 순서가 유지되는 구조
- 스택: 가장 최신 데이터만 꺼낼 수 있는 구조
- 큐: 가장 오래된 데이터만 꺼낼 수 있는 구조

이번 시간 목차

- 새로운 자료구조 소개
- 새로운 자료구조 예제
- 새로운 자료구조의 데이터들에서 반복하는 방법

새로운 자료 구조

- 리스트의 노드마다 리스트를 가지고 있다면?
- 예시) 파일 시스템, 소스 코드

새로운 자료 구조

- 세부 폴더들은 폴더에 연결. 일대다 (부모 자식) 관계
- 모든 폴더의 조상 (루트) 존재

트리란?

- 트리 정의: 루트로부터 부모 자식 관계로 연결된 구조

이진 트리

- 자식 노드가 최대 2개인 트리

  - root: 트리의 시작점 (타입: 이진 트리 노드)

이진 트리 노드 타입

- 자식 노드 2개 정보와 데이터의 묶음

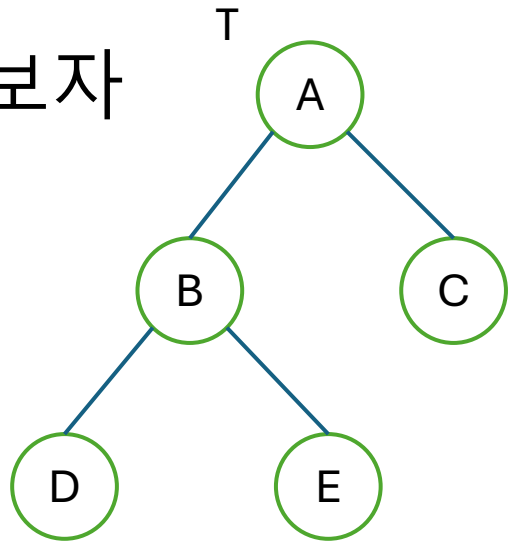
  - value: 저장된 데이터

  - left: 왼쪽 자식 노드 주소

  - right: 오른쪽 자식 노드 주소

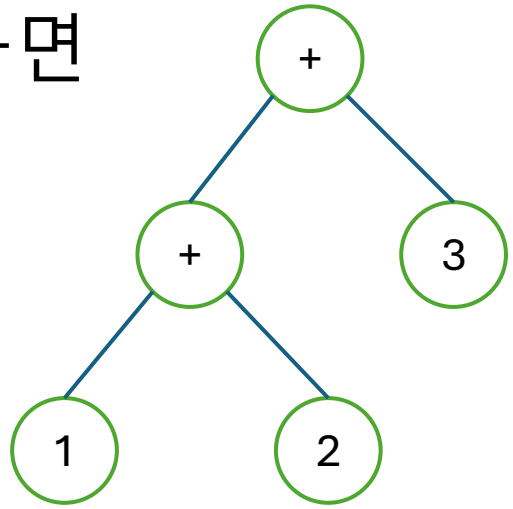
이진 트리 예시

- 다음과 같은 이진 트리를 만들어보자



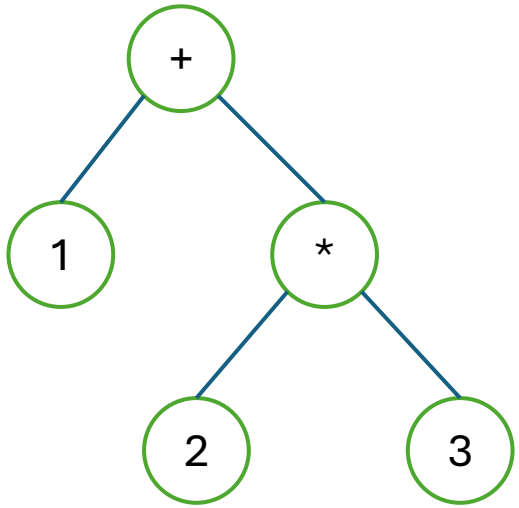
예제: 더하기/곱하기 수식 트리

- 1 + 2 + 3 수식을 이진 트리로 만들면 계산 순서가 명확히 드러난다



예제: 더하기/곱하기 수식 트리

- 1 + 2 * 3 수식을 이진 트리로 만들면 우선순위에 의해 모양이 달라진다



예제: 더하기/곱하기 수식 토큰

- 숫자 또는 연산자가 저장된다

  - type: 숫자, 더하기, 곱하기 구분
  - value: 숫자일 경우 값을 저장
  - predecence: 연산자 우선순위
    더하기는 1
    곱하기는 2
    숫자는 -1 (연산자 아님)

예제: 더하기/곱하기 수식 트리 생성

함수 expr_tree(수식 토큰 큐 Q, 최소우선순위 min)

- node = 새 노드 (dequeue(Q), NULL, NULL)
- 반복: size(Q) > 0인 동안

  op = front(Q)

  만약: 우선순위(op) < min이면 반복 종료

  dequeue(Q)

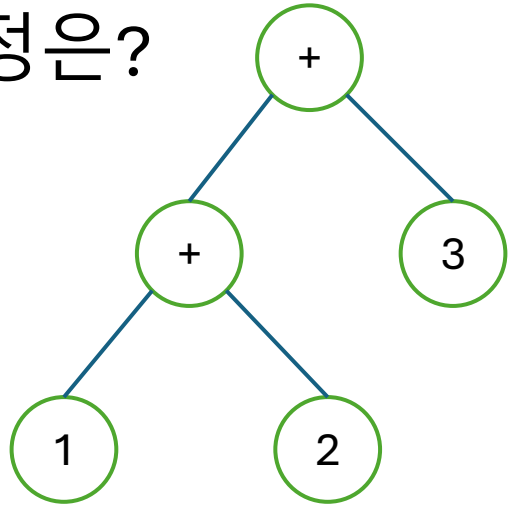
  node = 새 노드 (op,

  node,

  expr_tree(Q, 우선순위(op) + 1)
- 반환: node

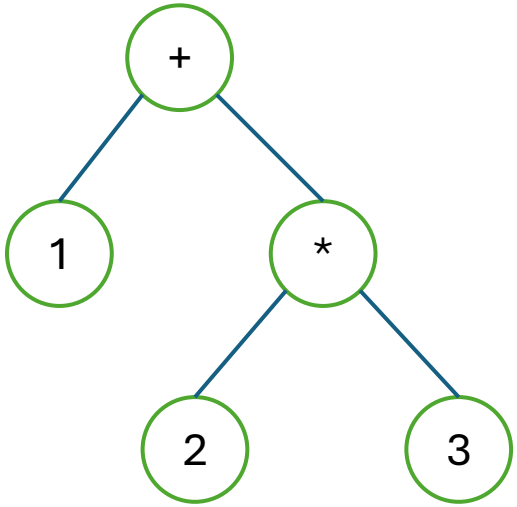
예제: 더하기/곱하기 수식 트리 생성

- expr_tree([1, '+', '2', '+', 3], 0) 실행 과정은?



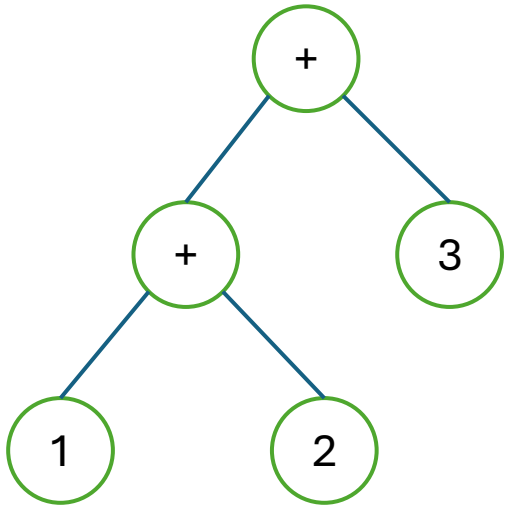
예제: 더하기/곱하기 수식 트리 생성

- expr_tree([1, '+', '2', '*', 3], 0) 실행 과정은?



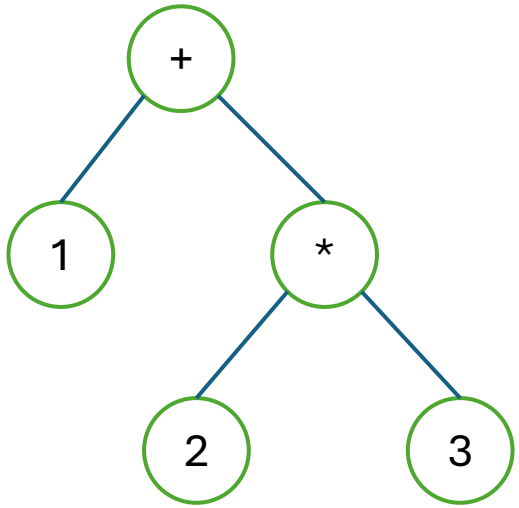
예제: 더하기/곱하기 수식 트리 계산

- 1 + 2 + 3 수식을 계산하려면?



예제: 더하기/곱하기 수식 트리 계산

- 1 + 2 * 3 수식을 계산하려면?



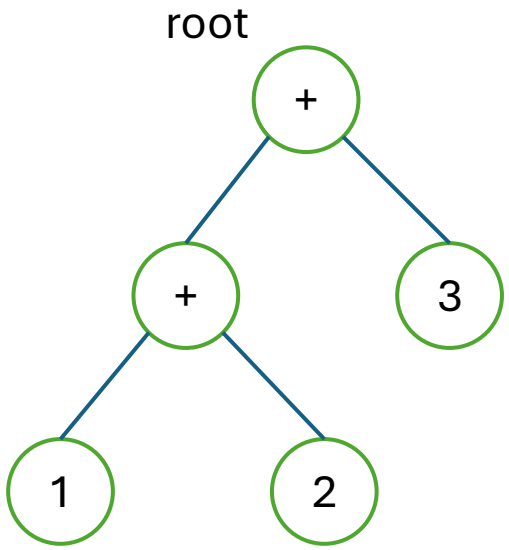
예제: 더하기/곱하기 수식 트리 계산

함수 EVAL(node):

- 만약: node가 숫자 노드면
  반환: node의 숫자 값
- 만약: node가 연산자 노드면
  left_val = EVAL(node->left)
  right_val = EVAL(node->right)
  만약: node의 연산자가 '+' 이면
  반환: left_val + right_val
  만약: node의 연산자가 '*' 이면
  반환: left_val * right_val

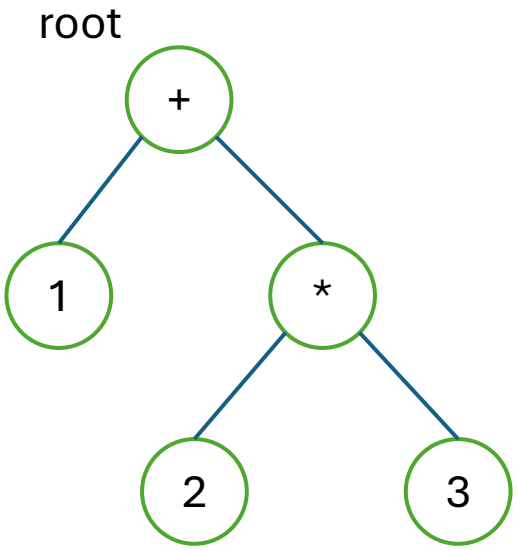
예제: 더하기/곱하기 수식 트리 계산

- EVAL(root)의 실행 과정은?



예제: 더하기/곱하기 수식 트리 계산

- EVAL(root)의 실행 과정은?



이진 트리 방문

- 목적: 이진 트리의 모든 노드에 대해 반복하기

- 깊이 우선: 자손들을 다 방문하고 다음 반복 진행
  데이터 처리 타이밍
  - Preorder: 자손들 방문 전 작업
  - Inorder: 왼쪽 자손들 방문 후 작업
  - Postorder: 모든 자손들 방문 후 작업

- 너비 우선: 한 세대를 다 방문하고 다음 반복 진행

깊이 우선 이진 트리 방문 구현

- 재귀 함수 호출:
  한 노드에 대한 동작만 정하면,
  노드들 전체에 적용된다

- 스택: 최적화가 가능하다

이진 트리 방문시 작업 타이밍 - Preorder

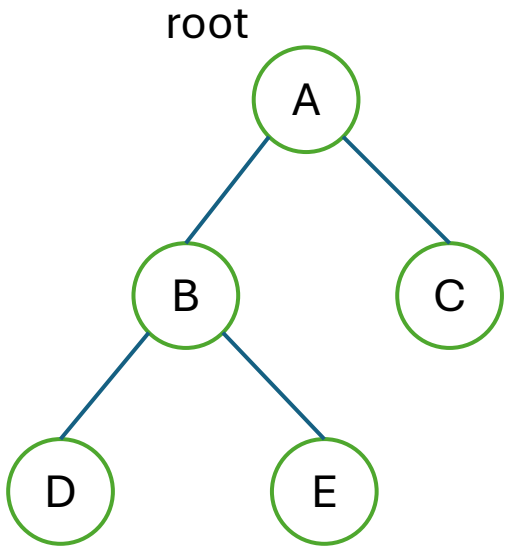
- 데이터 전처리를 하고, 자식들에서 반복한다

함수 TRAVERSE_PRE(node):

- 만약: node == NULL이면 반환
- PROCESS_PRE(node->value)
- TRAVERSE_PRE(node->left)
- TRAVERSE_PRE(node->right)

이진 트리 방문시 작업 타이밍 - Preorder

- TRAVERSE_PRE(root)에서
  전처리 작업이 실행되는 순서는?



이진 트리 방문시 작업 타이밍 - Inorder

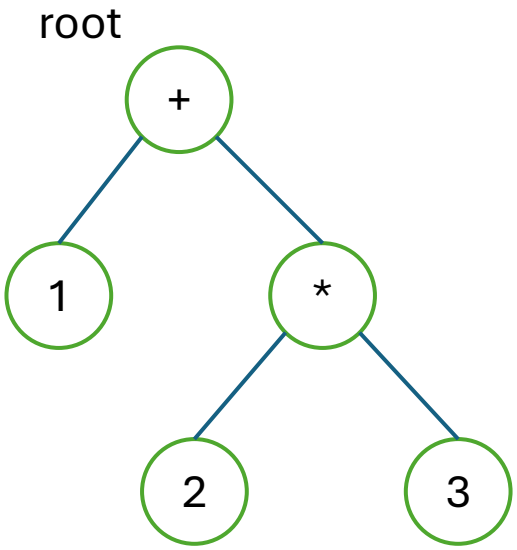
- 왼쪽 자손들 방문이 끝나면 데이터 중간 처리를 한다.

함수 TRAVERSE_IN(node):

- 만약: node == NULL이면 반환
- TRAVERSE_IN(node->left)
- PROCESS_IN(node->value)
- TRAVERSE_IN(node->right)

이진 트리 방문시 작업 타이밍 - Inorder

- TRAVERSE_IN(root)에서
  중간처리 작업이 실행되는 순서는?



이진 트리 방문시 작업 타이밍 - Postorder

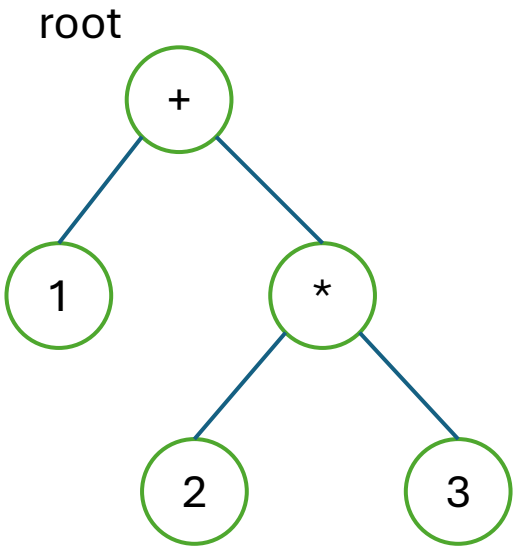
- 모든 자손들 방문이 끝나면 데이터 후처리를 한다.

함수 TRAVERSE_POST(node):

- 만약: node == NULL이면 반환
- TRAVERSE_POST(node->left)
- TRAVERSE_POST(node->right)
- PROCESS_POST(node->value)

이진 트리 방문시 작업 타이밍 - Postorder

- TRAVERSE_POST(root)에서
  후처리 작업이 실행되는 순서는?



이진 트리 방문 퀴즈

- Preorder traversal　　A B C D E F G
- Inorder traversal　　C D B A E G F
- Original tree???

요약

- 트리: 루트로부터 부모 자식 관계로 연결된 구조
- 이진 트리: 자식 노드가 최대 2개인 트리
- 이진 트리 노드: left, right, value
- 트리 방문: Preorder, Inorder, Postorder
- 다음 시간: C로 구현하기

이진 트리 방문 - Preorder (스택으로 구현)

함수 PREORDER_STACK(node):

- stack에 새로운 스택 저장
- PUSH(stack, node)
- 반복: stack이 비어있지 않으면
  node에 POP(stack) 저장
  만약: node == NULL이면, 다음 반복으로
  PROCESS_PRE(node->value)
  PUSH(stack, node->right)
  PUSH(stack, node->left)

이진 트리 방문 - Preorder (효율적 구현)

함수 PREORDER_STACK2(node):

- stack에 새로운 스택 저장
- 반복: node != NULL 또는 !EMPTY(stack)
  - 만약: node != NULL이면
    - PROCESS_PRE(node->value)
    - PUSH(stack, node)
    - node에 node->left 저장
  - 그렇지 않으면:
    - node에 pop(stack)->right 저장

이진 트리 방문 - Inorder (스택으로 구현)

함수 INORDER_STACK(node):

- stack에 새로운 스택 저장
- 반복: node != NULL 또는 !EMPTY(stack)
  - 만약: node != NULL이면
    - PUSH(stack, node)
    - node에 node->left 저장
  - 그렇지 않으면:
    - node에 POP(stack) 저장
    - PROCESS_IN(node)
    - node에 node->right 저장

이진 트리 방문 - Postorder (스택으로 구현)

함수 POSTORDER_STACK(node):

- stack에 새로운 스택 저장
- done에 NULL 저장
- 반복: node != NULL 또는 !EMPTY(stack)
  만약: node != NULL이면
  PUSH(stack, node)
  node에 node->left 저장
  다음 반복으로

이진 트리 방문 - Postorder (스택으로 구현)

node에 PEEK(stack) 저장
만약: node->right != done
        그리고 node->right != NULL이면
        node에 node->right 저장
        다음 반복으로
PROCESS_POST(node->value)
done에 node 저장
POP(stack)
node에 NULL 저장

이진 트리 방문 - 너비 우선 (큐로 구현)

함수 LEVELORDER_STACK(node):

- q에 새로운 큐 저장
- ENQUEUE(q, node)
- 반복: !EMPTY(q)

  node에 DEQUEUE(q) 저장

  만약: node != NULL이면

  PROCESS_LEVEL(node->value)

  ENQUEUE(q, node->left)

  ENQUEUE(q, node->right)