컴파일러

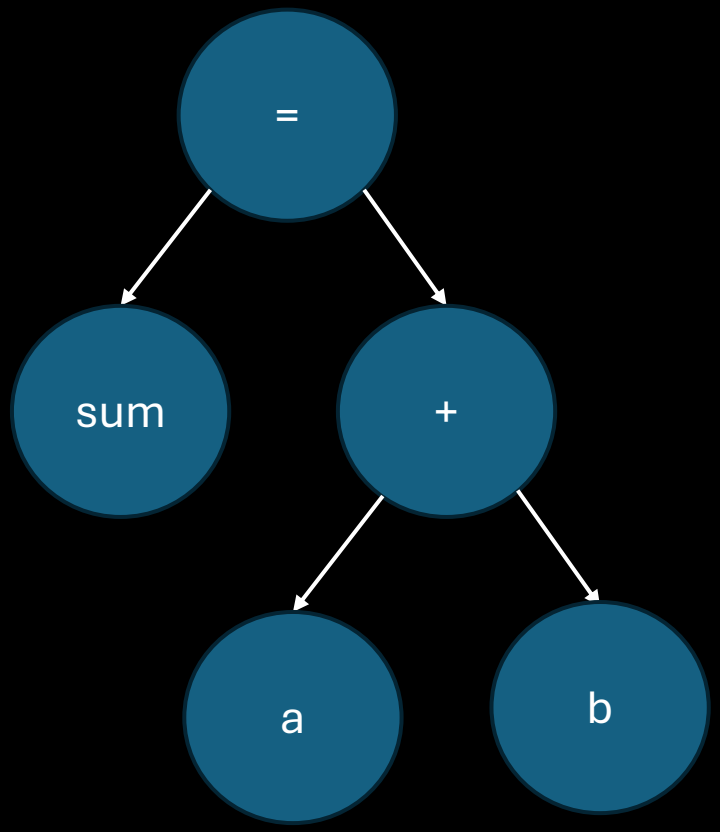
```c
#include <stdio.h>

int main() {
    int a = 5;
    int b = 10;
    int sum = a + b;

    printf("두 수의 합: %d\n", sum);

    return 0;
}
```

컴파일러는 코드를 읽으면
1. token을 나누고,
2.트리를 생성하고,
3. 순회를 한다.




```c
#include <stdio.h>

int main() {
    int a = 5;
    int b = 10;
    int sum = a + b;

    printf("두 수의 합: %d\n", sum);

    return 0;
}
```

컴파일러는 의미분석 과정에서 트리,순회를 사용한다

순회의 종류

1. 전위 순회 (preorder traversal)
2. 중위 순회 (inorder traversal)
3. 후위 순회 (postorder traversal)

전위 순회 (preorder traversal)

장점

- 트리 복제 및 복사

루트 노드를 먼저 처리하고 자식 노드를 재귀적으로 복사하면 된다.

- 계층 구조 표현

부모 노드가 항상 자식 노드 보다 먼저 출력 되는 로직이므로, 파일 시스템의 디렉터리 구조 등에 적합하다.

단점

- 트리 삭제 부적합

삭제를 하려면 자식 노드를 처리하고 부모 노드를 처리 해야 하는데, 전위 노드는 루트부터 탐색하는
로직이므로, 부적합 하다.

중위 순회 (inorder traversal)

장점

- 이진 탐색 트리(BST) 정렬

이진 탐색 트리를 중위 순회하면 노드들이 **오름차순으로 정렬**된 결과를 얻을 수 있다.

- 우수한 가독성

‘A + B’ 처럼 사람이 읽는 방법과 같아서, 트리를 사람이 읽기 좋은 형태로 출력하기 좋다.

단점

- 범용성 부족

이진 탐색 트리의 정렬된 출력이라는 특정 목적을 제외하면 전위 순회와 후위 순회에 비해 활용도가 떨어진다.

전위 순회 (preorder traversal)

장점

- 트리 삭제 및 메모리 해제

자식 노드를 먼저 방문하고 루트를 나중에 방문하는 방식이므로 트리 구조를 안전하게 삭제하거나 해제할때 적합하다.

단점

- 가독성 부족

사람이 코드를 읽는 순서와 달라 가독성이 부족하다.