MiniC 컴파일러 만들기(3):
Syntax → Semantic Analysis 단계로 확장하기

컴파일러, 9번째 시간

경민기, 2025-11-11

컴파일러 구조

- 소스코드 → 어휘 분석 → 구문 분석 → 의미 분석 → 코드 생성

- 지금까지: Lexical + Syntax Analysis
  - lexer.l: 토큰화
  - parser.y: 문법 분석, AST 생성
  - ast.c/h: 추상 구문 트리(AST) 구조 정의

- 이번 단계: Semantic Analysis
  - 심볼 테이블(Symbol Table)
  - 타입 검사(Type Checking)

주의 사항

- 심볼 테이블(Symbol Table) 은 AST(Abstract Syntax Tree)과 직접적인 실행 동작에는 영향을 주지 않음.
- 언어 변환(즉, 컴파일이나 코드 생성 단계) 에서는 결정적으로 중요한 역할

Symbol Table

- 역할: 식별자(identifier)의 선언 정보 저장
  - 이름, 종류(변수/함수/파라미터), 스코프, 타입 등
  - 스코프 관리: {} 블록마다 scope_push / scope_pop
  - 중복 선언 방지 및 미선언 참조 탐지

- 해시 테이블 + 연결 리스트 구조
- 스코프 경계: guard 노드로 구분
- 주요 함수:
  - symtab_init()
  - scope_push()
  - scope_pop()
  - sym_put()
  - sym_get()
  - symtab_dump()
- 검색 평균 복잡도: O(1)

해시(Hash)를 사용하는 이유

- 빠른 검색을 위해 문자열 → 정수로 변환
  - 선형 검색: O(n)
  - 해시 검색: O(1) 평균
- 충돌 시 연결 리스트로 관리 (체이닝)
- 실제 컴파일러(GCC/Clang)도 해시 기반 구조 사용

9wk > minic_symbol > C main.c

```c
#include <stdio.h>
#include "ast.h"
#include "sema.h"

int yyparse(void);
extern Node *root;

int main(void){
  if(yyparse()==0){
    printf("=== AST ===\n");
    print_ast(root, 0);

    printf("\n=== SYMBOL CHECK ===\n");
    sema_run(root);
    if(sema_errors==0) puts("Symbols: OK");
    else               printf("Symbols: %d error(s)\n", sema_errors);

    free_ast(root);
    return sema_errors ? 1 : 0;
  }
  return 2;
}

```

Type

- 단순 타입 분류: int, float, char, void, error
- TypeKind 열거형으로 관리
- 연산 시 타입 승격 규칙 적용

- 예: int + float → float
- 비교/논리 연산 결과 → int

Scope 경계

- Scope(스코프): 식별자(변수/함수)가 유효한 코드 영역
- Scope Boundary(경계): 새로운 블록이 시작하거나 끝나는 지점
- 예: int x; // 전역 스코프 void f() { int y; // 함수 스코프 { int z; } // 블록 스코프 (여기가 경계) } • 컴파일러는 '{'에서 scope_push(), '}'에서 scope_pop() 호출 • Guard node를 이용해 각 스코프의 경계를 표시 → 스코프 종료 시, 해당 블록 내 변수 심볼을 해제(pop) → 이름 중복, shadowing, 지역 변수 유효 범위를 관리

Symantic Analysis

- AST 순회하며 의미 검사 수행
  - 선언/참조 관계 확인(sym_get)
  - 타입 일관성 검사(TypeKind)
  - 주요 검사 항목:
    - 1) 중복 선언
    - 2) 미선언 변수 참조
    - 3) 타입 축소(narrowing)
    - 4) 함수 인자 개수/타입 불일치

9wk > minic_type > test.mc

```
1   int sum(int a, int b) { return a + b; }
2   float g(int x) { return x + 0.5; }
3   void v() { return; }
4
5   int main() {
6     int x = 0;
7     float f = 3.0;
8     x = f;              // float→int 축소 경고
9     y = 1;              // 미선언 사용
10    int y = 10, z = 1.2; // 1.2는 float→int 축소 경고
11    if (x < y) x = x + y;
12    return sum(x, y, 42); // 인자 수 불일치
13  }
```

```
=== SYMBOL CHECK ===
[SEMA] line 8: narrowing assignment to 'x' (float→int)
[SEMA] line 9: assignment to undeclared identifier 'y'
[SEMA] line 10: narrowing init from float to int in 'z'
[SEMA] line 12: wrong # of args in call to 'sum'
Symbols: 4 error(s)
```

다음 시간

- 현재 단계: Lexical → Syntax → Semantic
- 변수, 함수, 스코프, 타입 검사 수행

- 다음 단계:
  - (미사용 변수 경고, do-while, switch 추가)
  - AST → IR (중간 코드 생성)
  - Constant Folding, Mini-Compiler 완성