**Design and Development of Compiler**

**for C- Language**

**Phase3: Design and Implementation of Semantic Analyzer**

과목명: [CSE4120] 기초 컴파일러 구성

담당교수: 서강대학교 컴퓨터공학과 정 성 원

개발자: 5조

김민우 (20141510)

정재훈 (20151607)

프로젝트 제목: Design and Development of Compiler for C- Language:

Phase3: Design and Implementation of Semantic Analyzer

참여조원: 김민우 (20141510), 정재훈 (20151607)

1. 개발 목표

Phase3의 개발 목표는 C- Language로 작성된 소스 코드에 대하여 Semantic Analysis를 수행하는 것이다. 주어진 C- 소스 코드의 분석을 통하여 BNF Grammar을 만족하지만 Semantic하게 오류가 생기는 원인을 확인할 수 있어야 한다.

1. 개발 범위 및 내용

Phase2에서 구현한 parser를 이용하여 semantic analyzer를 구현한다. Semantic analyzer는 symbol table을 생성하여 출력하고, type check를 수행하여 semantic한 오류가 존재하는지 확인한다.

symbol table은 각 Scope에 있는 Variable과 Parameter, Function에 대한 정보를 출력해주어야 한다. 구체적으로 node의 이름, scope의 깊이(scope), frame pointer로부터 offset(Loc), node의 종류(V/P/F), array 여부, array인 경우 array size, type(int, void), line Numbers(처음 등장하는 코드상의 위치) 등을 출력해야 한다.

Type Check는 Language 문법에서 정의한 Semantic에 따라 각 Statement의 Type을 검사하고 만약 예상되는 Type과 다른 Type을 발견하면 해당하는 line number와 Error message를 출력한다. 또한 Synthesized Attribute인 data type을 post order로 계산하여 각 node의 타입을 결정해줘야 한다.

1. 제작 내용

symbol table

parsing 이후 생성된 AST에 대해서 preorder traverse를 하면서 symbol table을 생성한다. AST의 각 노드들을 방문하면서 insertNode()와 get\_out\_of\_scope()를 수행한다. insertNode()에서는 노드의 종류에 따라서 상이한 과정을 수행한다. 선언문에 존재하는 name의 경우에는 해당 name이 존재하는지 현재 scope의 symbol table을 확인하여 재선언 여부를 확인하고, 아닌 경우에 symbol table에 노드 정보를 추가한다. expression에 존재하는 name의 경우에는, 해당 name이 선언되어 있는지 모든 symbol table을 확인하고, 선언이 되어 있으면 symbol table에 노드 정보를 추가한다. compound statement에 대한 node라면, 새로운 scope를 생성한다. insertNode()를 수행한 다음 get\_out\_of\_scope()를 수행한다. 이 함수는 만일 node가 compound statement라면, 현재 scope를 벗어난다. AST의 모든 노드들을 방문을 하면 symbol table이 생성되고, printSymTab()는 각 scope에 대해서 symbol table을 출력한다.

**symbol table 출력 방식**

함수의 경우, Loc에 함수가 소스코드에서 등장한 순서를 출력한다. Loc의 값이 낮을수록 먼저 등장한 함수이다.

함수의 line number는 함수가 선언된 line number와 호출된 line number를 출력한다.

global scope의 scope 값은 0이다.

symbol table은 아래와 같은 예외들만 확인하고 문제가 없을 경우 table을 출력한다. 나머지 예외처리는 type check에서 처리한다.

* 1. 선언되지 않은 변수나 함수는 사용 불가
  2. 변수나 함수, 함수 파라미터 선언 시 중복된 이름 체크
  3. array 아닌 변수를 array처럼 사용하려는 경우
  4. 함수 호출 시 호출된 것이 함수가 맞는지 확인
  5. 함수의 return type이 void일 경우 return이 없어야 함
  6. 변수나 parameter 선언 시 void type으로 선언할 수 없음

Type Check

Symbol Table이 완성된 후 Post order로 syntax Tree를 traverse하여 아래서부터 위로 Type을 계산한다. Const나 Arr에 대한 Type은 해당 노드에서 바로 계산할 수 있지만, ID Variable에 대한 type은 Declaration을 봐야 한다. 따라서 Symbol table의 Insert Node 과정에서 Symbol table을 look up 하고 ID variable에 대한 type을 부여한다. 이와 같은 과정을 거치면 각 leaf node에는 data type이 모두 부여되어 있으며, 자식 type들을 통해 부모 type을 계산한다. Main function에 대한 위치, Parameter 조건은 Postorder나 Preorder 둘 다 상관없지만, main함수의 return statement 존재 여부가 post order에서 검사할 수 있으므로 Type Check에서 해당 조건을 검사한다. 따라서 typeCheck()에서는 상이한 과정을 검사한다.

* 1. conditional statement에 INT값이 오는가
  2. Binary Operation의 두 operand가 INT 값인가
  3. INT로 선언된 함수에서 return하는 값이 INT인가
  4. Call procedure에서 parameter 개수와 type이 일치하는가
  5. Main함수가 가장 끝에 오는가, main함수가 void로 선언되었는가
  6. main함수의 parameter가 void인가

Type Check가 마무리 되면 Semantic Analysis를 종료한다.

1. 평가

(test.c)



(결과)



