컴파일러 설계 project3 Report

2017029616 박병현

* Compilation

1) 환경: Ubuntu 20.04

2) 방법: make => ./cminus\_semantic <input filename>

* Code explanation
  1. main.c
     1. NO PARSE 를 FALSE로 설정
     2. NO ANALYZE 를 FALSE로 설정
     3. TRACEPARSE 를 TRUE로 설정’ 후 디버깅
  2. symtab.h
     1. struct ScopeList: Scope를 관리하는 구조체이다. 이름과 bucket의 list를 인자로 가지고 있다. scope에 포함된 모든 요소를 bucket list로 관리한다.
     2. struct BucketList: Scope에 포함된 Variable, Function등을 저장하는 구조체이다.
     3. struct LineList: Bucket에 해당하는 Variable, function등이 몇번째 라인에 등장하는지 저장하는 구조체이다.
     4. struct FuncList: Function을 관리하는 구조체이다. 함수에 대한 정보를 저장한다.
     5. scopeList 배열: 프로그램에 존재하는 모든 scope를 저장하는 배열이다. 이 배열을 통해 scope이름이 주어질 때 찾고자 하는 scope 구조체를 찾게 된다.
     6. funcList 배열: 프로그램에 존재하는 모든 함수를 저장하는 배열이다. 이 배열을 통해 모든 함수를 관리한다.
  3. symtab.c
     1. makeScope(): 함수 정의, compound statement 등을 발견할 때 새로운 scope를 생성하는 함수이다. 이름과 parent scope 구조체를 인자로 받아 parent scope을 가르킴을 통해 계층 구조를 만든다.
     2. find\_scope(): scope이름을 통해 scope 구조체를 찾는 함수이다.
     3. find\_func(): funcList 배열에서 함수 이름으로 funcList 구조체를 찾을 때 사용하는 함수이다.
     4. add\_parameter: 함수 정의에서 매개변수들을 찾았을 때 해당 매개변수에 대한 정보를 해당 함수와 연결시켜주는 함수이다.
     5. st\_insert(): 특정 scope에 name에 해당하는 변수 혹은 함수명을 찾는다. 존재한다면 이미 존재하는 곳에 line number만 추가해주고, 존재하지 않는다면 table에 새로 추가한다.
     6. st\_lookup(): 매개변수로 주어진 scope부터 global scope까지 name에 해당하는 변수 혹은 함수명이 존재하는지 찾는다. 존재한다면 해당하는 bucketlist 구조체를 return 하고, 존재하지 않는다면 NULL을 return 한다.
     7. st\_lookup\_excluding\_parent(): 특정 scope에 변수나 함수에 대한 중복된 선언이 있는지 확인하기 위해 특정 scope에서만 존재여부를 확인하는 함수이다.
     8. printSymTab(): symbol table과, 새로 추가한 function table을 출력하는 함수이다.
  4. analyze.c
     1. buildSymtab(): Symbol table을 처음 만들 때 호출되는 함수이다. 현재 scope을 global로 설정하고, input 과 output함수에 대한 정의를 추가한다.
     2. insertNode(): buildSymtab에 의해 호출되어 symbol table을 만드는 함수이다. 각 treenode의 type마다 알맞은 행동을 하도록 switch case 문으로 구현하였다. scope의 변화를 반영하기 위해, 함수의 정의 혹은 Compound statement가 있을 때에는 새로운 scope을 만들었고, compound statement가 끝날 때 다시 원래의 scope으로 되돌아간다. 되돌아가는것은 traverse함수를 호출시 postOrder로 실행하는 부분에 해당 작업을 하도록 추가하였다. 또한 function call후 곧바로 오는 compound statement에 대해서는 scope을 추가하게 되면 중복 될 수 있으므로, madeScope이란 변수를 선언하여 이러한 상황을 막았다.
     3. typeCheck(): symbol table을 이용해 semantic error을 확인하는 함수이다.
     4. checkNode(): typeCheck()에 의해 호출되어 semantic error을 확인하는 함수이다. scope의 구현을 위 경우와 마찬가지로, traverse함수를 호출할 때 function 정의나 compound statement가 있으면 해당 scope를 찾고, compound statement가 끝나면 scope를 빠져나오는 방법으로 구현하였다.
* Example and Screenshot

1. input and output of input file: mytest.1.txt

텍스트이(가) 표시된 사진

자동 생성된 설명

symbol table && function table

텍스트이(가) 표시된 사진

자동 생성된 설명

error messages





2)input and output of input file: mytest.2.txt

텍스트이(가) 표시된 사진

자동 생성된 설명

symbol table && function table, error message 텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명