컴파일러설계

이헌준 교수님

2019032160 유석원

Project 2

개요

이번 Project 2 과제는 Bison을 활용한 C-Minus Parser를 만드는 것이 목표입니다. Parser가 입력 소스 코드 문자열을 읽고, C-Minus 문법에 맞춰 토큰화하고 parse한 뒤, abstract syntax tree를 출력하게 됩니다. Scanner는 Project 1에서 만들었던 Lex scanner를 사용했고, LALR(1) parser를 만들기 위해 CFG grammar를 *cminus.y* 파일에 정의했습니다.

구현: C 코드

정상적으로 parser를 구현하기 위해, *globals.h*, *util.h*, *util.c* 파일들을 수정했습니다. CFG에 맞춰 parser가 작동할 수 있도록 *globals.h*에서 TreeNode의 kind를 새로 정의했습니다. 기존의 TreeNode의 kind는 NodeKind(node들의 종류를 담는 열거형), StmtKind(IfK, WhileK, ReturnK, CompoundK로 수정됨), ExpKind(OpK, ConstK, AssignK, CallK, VarAccessK로 수정됨)가 존재했으나, 정의에 관한 state가 필요해, DeclKind(VarK, FuncK, ParamK)를 추가했고, 별도로 변수의 이름, 타입, 연산자의 입력을 정상적으로 읽어와 TreeNode에 정상적으로 저장할 수 있도록, AST에는 추가되지 않지만, CFG를 처리하는 단계에서 사용하는 ParserKind(TypeK, OpcodeK, IdK)를 새로 만들었습니다. 추가로 실제로 C-Minus에서 사용하는 타입은 Int와 Void 두 가지이지만, parser를 위해 VoidArray, IntegerArray, None 타입 세 개를 추가했습니다. 또, Void parameter나 If-Else statement를 처리하기 위해 TreeNode의 attr에 flag를 추가했습니다.

기존의 *util.h*와 *util.c*는 StmtNode와 ExpNode를 생성하는 함수만 정의되었으나, TreeNode에 새로운 타입들이 생겼기 때문에 DeclNode와 ParserNode를 생성하는 함수도 추가되었습니다. 이 함수는 빈 TreeNode를 생성하는 함수이기 때문에 nodekind만 서로 다르고 다른 멤버 변수는 동일한 값으로 초기화해 주었습니다. 그다음 printTree 함수에서 먼저 if문으로 nodekind를 체크한 뒤 해당하는 nodekind의 열거형을 switch-case문에 넣고 TreeNode의 멤버 변수들을 확인해 알맞은 형식으로 출력하도록 구현했습니다.

구현: *cminus.y*

PPT에 제공된 CFG를 보고 기존의 tiny.y 파일을 참고해 C-Minus 문법에 맞는 parser를 구현했습니다. Token으로 예약어(6종 중 ELSE는 제외)를 지정했고, RPAREN은 nonassoc, ELSE도 마찬가지

로 nonassoc, ID와 NUM을 token, 심볼들 (19종 중 사칙 연산자와 ASSIGN, RPAREN 제외)를 token, ERROR를 token, 그다음으로 덧셈과 뺄셈을 left, 곱셈과 나눗셈도 left로 지정하고 마지막으로 ASSIGN을 right의 순서로 지정했습니다. 그 뒤 C-Minus의 CFG를 구현했는데, 전체 내용을 다 적기에는 양이 너무 많기 때문에 생략하고 특이 사항만 적도록 하겠습니다. 모든 term은 TreeNode 타입으로 생성되기 때문에 처음 직접적으로 입력을 받아 attr의 name이나 op, val을 copyString 함수나 atoi로 저장하려 했을 때 에러가 발생했습니다. 따라서 안전하게 값들을 저장하기 위해 AST에는 사용되지 않는 별도의 TreeNode type이 필요했기 때문에 위의 *globals.h*에 ParserNode를 정의하게 된 것입니다. 따라서, 해당 type의 Node에만 attr의 name, op, val을 처리할 수 있게 만들어 정상적으로 값들만 가져와서 AST의 TreeNode들에 저장할 수 있었습니다. 또한, If문은 If 하나만 존재하거나 Else가 따라오는 If-Else문으로 나눌 수 있는데, If-Else를 별도로 처리하기 위해 If-Else문은 flag를 TRUE로 설정했습니다. Parameter도 Void parameter는 별도로 처리해야 하므로 마찬가지로 Void는 flag를 TRUE로 설정했습니다.

빌드

PPT의 설명을 따라 main.c 파일에서 NO_ANALYZE와 TraceParse를 TRUE로 설정했고, 나머지는 FALSE로 설정했습니다. Makefile이 제공되기 때문에 해당하는 makefile과 test/result 텍스트 파일을 기존의 컴파일러 파일들이 있는 디렉토리에 함께 뒀으며, 모든 결과를 확인한 뒤 현재 "make clean" 명령어를 실행해 오브젝트 파일과 cminus_parser 파일은 삭제된 상태입니다.

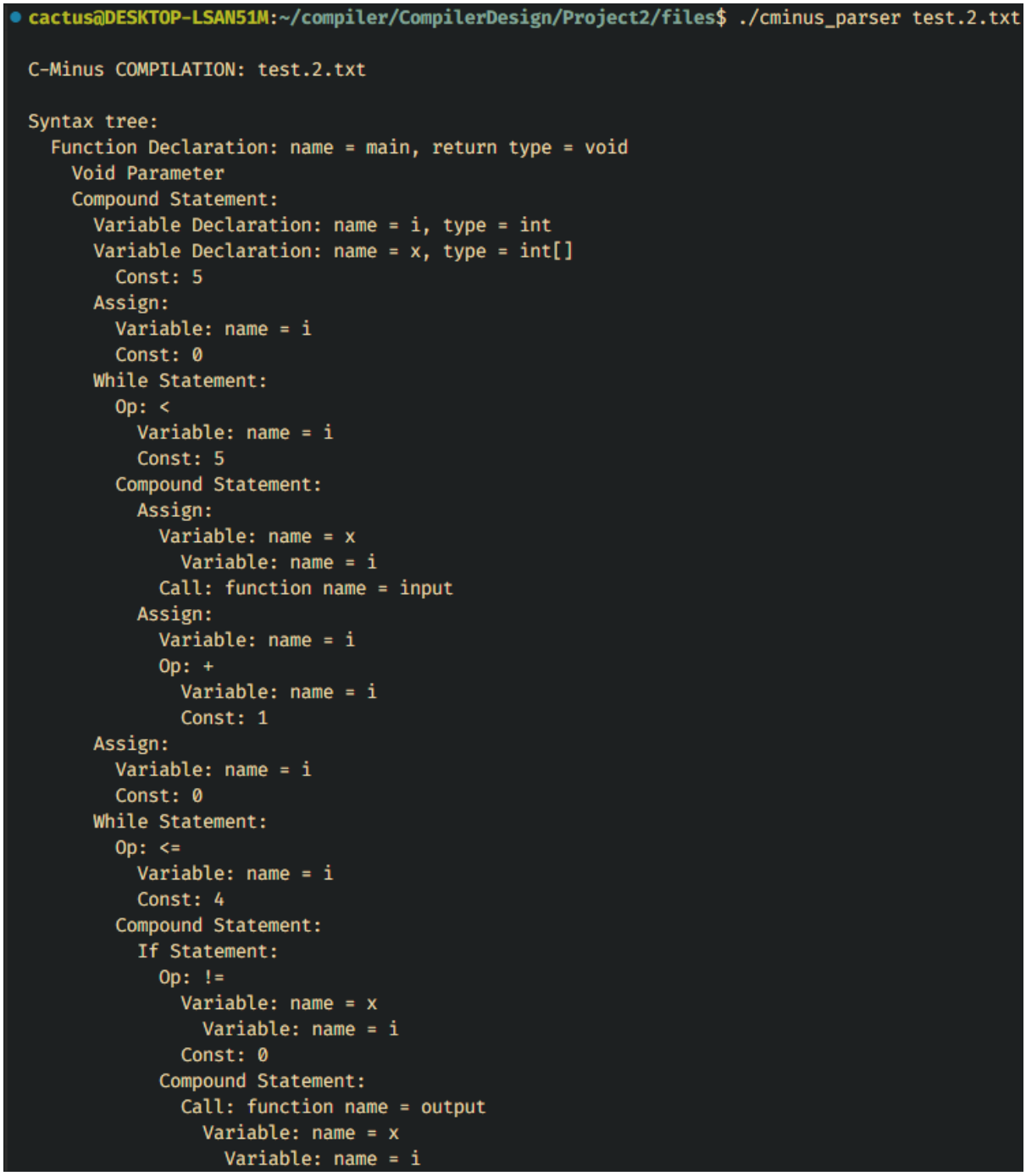
테스트 결과

```
cactus@DESKTOP-LSAN51M:~/compiler/CompilerDesign/Project2/files$ ./cminus_parser test.1.txt

C-Minus COMPILATION: test.1.txt

Syntax tree:
  Function Declaration: name = gcd, return type = int
    Parameter: name = u, type = int
    Parameter: name = v, type = int
    Compound Statement:
      If-Else Statement:
        Op: ==
          Variable: name = v
          Const: 0
        Return Statement:
          Variable: name = u
        Return Statement:
          Call: function name = gcd
            Variable: name = v
            Op: -
              Variable: name = u
              Op: *
                Op: /
                  Variable: name = u
                  Variable: name = v
                Variable: name = v
  Function Declaration: name = main, return type = void
    Void Parameter
    Compound Statement:
      Variable Declaration: name = x, type = int
      Variable Declaration: name = y, type = int
      Assign:
        Variable: name = x
        Call: function name = input
      Assign:
        Variable: name = y
        Call: function name = input
      Call: function name = output
        Call: function name = gcd
          Variable: name = x
          Variable: name = y
```

test.1.txt를 입력한 결과



```
cactus@DESKTOP-LSAN51M:~/compiler/CompilerDesign/Project2/files$ ./cminus_parser test.2.txt

C-Minus COMPILATION: test.2.txt

Syntax tree:
  Function Declaration: name = main, return type = void
    Void Parameter
    Compound Statement:
      Variable Declaration: name = i, type = int
      Variable Declaration: name = x, type = int[]
        Const: 5
      Assign:
        Variable: name = i
        Const: 0
      While Statement:
        Op: <
          Variable: name = i
          Const: 5
        Compound Statement:
          Assign:
            Variable: name = x
              Variable: name = i
            Call: function name = input
          Assign:
            Variable: name = i
            Op: +
              Variable: name = i
              Const: 1
      Assign:
        Variable: name = i
        Const: 0
      While Statement:
        Op: <=
          Variable: name = i
          Const: 4
        Compound Statement:
          If Statement:
            Op: !=
              Variable: name = x
                Variable: name = i
              Const: 0
            Compound Statement:
              Call: function name = output
                Variable: name = x
                  Variable: name = i
```

test.2.txt를 입력한 결과

두 기초적인 test case를 정상적으로 통과하는 것을 볼 수 있습니다.