Project #2 Parse Tree Report

2022076062 김유찬

Project Goal

Bison을 사용해서 C - Minus Parser 구현하기

- C-Minus 문법에 따라 토큰화하고 구문 분석을 수행한 뒤 AST를 출력
- C-Minus Scanner는 Lex를 사용
- Bison을 사용하여 Parser 구현
  - Bison은 CFG를 입력 받아 parser 생성
  - Ambiguous grammar인 경우 conflicts 발생

main.c

```
/* set NO_PARSE to TRUE to get a scanner-only compiler */
#define NO_PARSE FALSE
/* set NO_ANALYZE to TRUE to get a parser-only compiler */
#define NO_ANALYZE TRUE
```

Parser에 맞게 수정

globals.h (추가된 내용 위주)

```
61  /*****************************************/
62  /***********   Syntax tree for parsing ************/
63  /*****************************************/
64
65  typedef enum {StmtK,ExpK} NodeKind;
66  typedef enum {AssignK,CompK,RetK,VarK,WhileK,SelectK} StmtKind;
67  typedef enum {OpK,ConstK,IdK,VarDK,TypeK,FuncDK,ParamK,CallK} ExpKind;
68
69  /* ExpType is used for type checking */
70  typedef enum {None,Void,Integer,VoidArr,IntegerArr} ExpType;
71
72  #define MAXCHILDREN 3
73
74  typedef struct treeNode
75     { struct treeNode * child[MAXCHILDREN];
76       struct treeNode * sibling;
77       int lineno;
78       NodeKind nodekind;
79       union { StmtKind stmt; ExpKind exp;} kind;
80       union { TokenType op;
81               int val;
82               char * name; } attr;
83       char * typestring;
84       ExpType type; /* for type checking of exps */
85     } TreeNode;
```

- NodeKind는 기존대로 Stmt와 Exp 두개로 나눠서 구현
- TreeNode 구조체에 typestring 멤버 변수 추가 (type에 대한 변수)

▼ StmtKind

- CompK : compound statement
- RetK : return statement
- VarK : variable
- WhileK : while statement
- SelectK : if / if - else statement

▼ ExpKind

- VarDK : variable declaration
- TypeK : type_specification
- FunDK : function declaration
- ParamK : parameter
- CallK : function call

util.c (추가된 내용 위주)

StmtK

```
if (tree->nodekind==StmtK)
{ switch (tree->kind.stmt) {
    case AssignK:
      fprintf(listing,"Assign:\n");
      break;
    case CompK:
      fprintf(listing,"Compound Statement:\n");
      break;
    case EmptyK:
      fprintf(listing,"empty\n");
      break;
    case RetK:
      fprintf(listing,"Return Statement:\n");
      break;
    case VarK:
      fprintf(listing,"Variable: name = %s\n",tree->attr.name);
      break;
    case WhileK:
      fprintf(listing,"While Statement:\n");
      break;
    case SelectK:
      fprintf(listing,"%s Statement:\n",tree->attr.name);
      break;
    default:
      fprintf(listing,"Unknown ExpNode kind\n");
      break;
  }
```

Statement에 들어갔을 때 출력되는 내용들

VarK는 아래 VarDK와 구분하기 위해 statement에 작성

SelectK에서는 reduce를 할 때 맞는 문법에 맞게 if인지 if else인지 상황에 맞게 값 할당

ExpK

```
else if (tree->nodekind==ExpK)
{ switch (tree->kind.exp) {
    case OpK:
      fprintf(listing,"Op: ");
      printToken(tree->attr.op,"\0");
      break;
    case ConstK:
      fprintf(listing,"Const: %d\n",tree->attr.val);
      break;
    case IdK:
      fprintf(listing,"Id: %s\n",tree->attr.name);
      break;
    case VarDK:
      fprintf(listing,"Variable Declaration: name = %s, type = %s\n",tree->attr.name, tree->typestring);
      break;
    case FuncDK:
      fprintf(listing,"Function Declaration: name = %s, return type = %s\n",tree->attr.name, tree->typestring);
      break;
    case TypeK:
      fprintf(listing,"Void Parameter\n");
      break;
    case ParamK:
      fprintf(listing,"Parameter: name = %s, type = %s\n",tree->attr.name, tree->typestring);
      break;
    case CallK:
      fprintf(listing,"Call: function name = %s\n",tree->attr.name);
      break;
    default:
      fprintf(listing,"Unknown ExpNode kind\n");
      break;
  }
```

Expression에 들어갔을 때 출력되는 내용들

variable, function등 type을 출력하기 위해 typestring 멤버 변수에 값 할당

cmiuns.y (핵심 부분 위주)

기본적인 방법은 child를 계속 늘려가 트리 형태로 만들어 주는 것

bison이 알아서 reduce를 해주기 때문에 문법에 맞게 TreeNode에 값, 포인터 할당

declaration

```
%token IF WHILE RETURN INT VOID ELSE
%token ID NUM
%token ASSIGN EQ NE LT LE GT GE LPAREN RPAREN LBRACE RBRACE LCURLY RCURLY SEMI COMMA
%token ERROR

%nonassoc IFX
%nonassoc ELSE
%left PLUS MINUS
%left TIMES OVER
```

- terminal symbols (tokens) 정의
- 아래로 갈 수록 우선 순위가 높다.

  %token, $left, %rigth, %nonassoc을 각 토큰의 우선순위, associative에 맞춰 선언

리스트 형태들의 구현

- 자기 non-terminal이 계속해서 늘어나도록 구현된 list 형태
- while 반복문을 통해 sibling 연결을 해준다
- 리스트 형태 : declaration_list, param_list, local_declarations, statement_list, arg_list

identifier, number, empty 구현

- terminal symbol을 따로 빼서 구현
- tokenString이 나중에 잘못 참조될 수 있기 때문

type_specifier 구현

- int, int[], void, void[]에 대한 타입을 적절히 적용시킬 수 있도록 구현
- array일 경우엔 LBRACE와 RBRACE 사이에 있는 토큰을 child로 넘기도록 구현(모든 배열 형태)

selection_stmt 구현

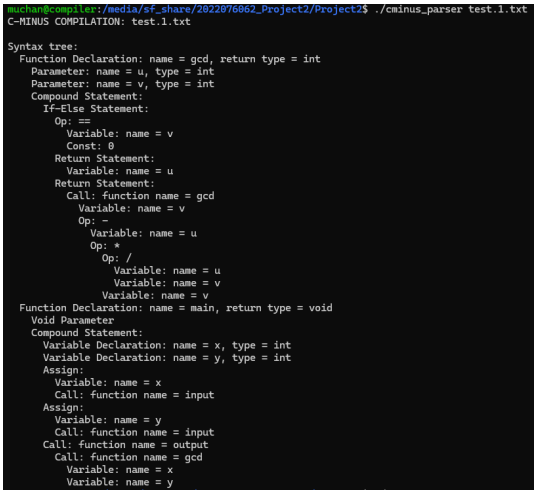
```
selection_stmt      : IF LPAREN expression RPAREN statement %prec IFX
                    {
                        $$ = newStmtNode(SelectK);
                        $$->attr.name = "If";
                        $$->child[0] = $3;
                        $$->child[1] = $5;
                    }
                    | IF LPAREN expression RPAREN statement ELSE statement
                    {
                        $$ = newStmtNode(SelectK);
                        $$->attr.name = "If-Else";
                        $$->child[0] = $3;
                        $$->child[1] = $5;
                        $$->child[2] = $7;
                    }
                    ;
```

%prec사용

1. 우선순위 조정 : %prec을 사용하여 우선순위 수동 부여 → 모호성 해결
2. if - else 구문에서는 if와 else가 짝이 애매하게 맞는 상황이 있음(dangling else)
3. if문에서 else 없는 구문에 낮은 우선순위를 부여해 모호성 해결 → if와 가까운 else가 결합
4. IFX를 사용하여 가장 가까운 if와 else가 결합하도록 구현
5. selection_stmt 두 번째 규칙에서만 else와 매칭됨

실행결과

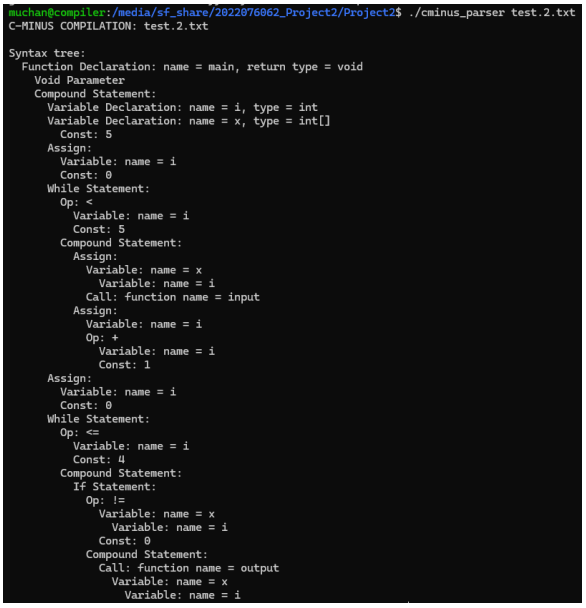
test.1.txt 결과



```
muchan@compiler:/media/sf_share/2022076062_Project2/Project2$ ./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.2.txt 결과

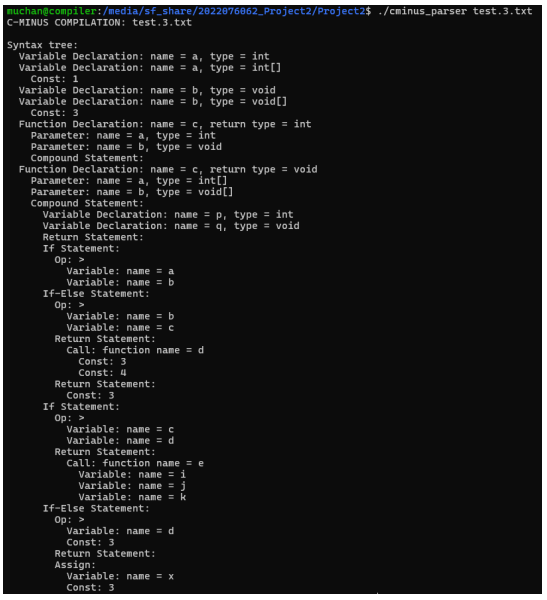


```
muchan@compiler:/media/sf_share/2022076062_Project2/Project2$ ./cminus_parser test.2.txt
C-MINUS COMPILATION: test.2.txt

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.3.txt 코드 및 결과

```
/* test.3.txt*/
int a; int a[1];
void b; void b[3];
int c(int a, void b){}
void c(int a[], void b[]){
int p;
void q;
return;
if (a>b);
if (b>c) return d(3,4);
else return 3;
if (c>d) return e(i,j,k);
if (d>3) return;
else x = 3;
}
```



- 마지막 if 의 중첩이 ambigous없이 잘 실행됨
- return; 만 있으면 Return Statement만 호출하고 뒤에 expression이 있으면 그거에 맞게 출력

난관

1. LE,LT,GT... 등등 따로 CFG를 새로 만들어서 하려다가 새로운 Node를 할당하는것 고민

   → relop, addop, mulop 에 관한 건 한번 더 문법을 거치지 않고 바로 경우의 수 나눠서 구현

2. if/if-else를 구현할 때 무작정 강의 자료에 있는 MIF/UIF를 썼는데 무한루프를 돌았다. 처음에 %prec에 대해 잘 몰랐고 어떻게 쓰는지도 잘 몰라서 구현하는데 오래 걸렸다.
3. 처음에 $$에 NewNode를 만들거나 $$에 $1, $2등 값을 넣는게 헷갈렸는데 차근차근 segmentation fault 랑 연결이 안돼서 출력이 끊기는 경험을 해보면서 어떻게 하는지 깨달았음