HYU-ELE4029, Compilers 2. Parser

컴퓨터소프트웨어학부

2018008013 김영중

1. Compilation method and environment

- Env: Windows 10 WSL – Ubuntu 18.04
- GNU Make 4.1. gcc 7.4.0
- GNU Bison 3.0.4, flex 2.6.4
- make 명령어를 통해 cminus 실행파일 생성

2. Scanner 변경점

1) yylex 함수명 변경

기존까지의 구현체에서는 lexer의 함수명을 "yylex" 그대로 사용하고 있었습니다. 이는 scanner의 flex 구성에서 getToken 함수가 호출하여 token을 하나씩 읽어오는 역할을 하고 있었습니다. getToken함수는 yylex의 wrapper함수로써, 실행 초기에 lexer 상태를 초기화하거나, 현재 읽어 들인 string을 tokenString 문자열에 복사하는 역할을 하고 있었습니다.

하지만 yacc의 parser가 도입되면서 getToken을 yylex 함수 대신에 lexer로 사용하고자 하였고, 이에 따라 내부적으로 yylex 함수를 static method로 재구성하였습니다. 이 과정에서 yylex의 symbol이 재정의되었고, 이를 해결하기 위해 flex의 macro인 "YY_DECL"을 통해 lexer함수의 함수명을 "_yylex"로 변경하여 활용했습니다.

```
12
13   #define YY_DECL int _yylex (void)
14
```

[그림 1] cminus.l 내부에 선언된 YY_DECL 매크로

2) tokenString 배열 확장

parser 구현 과정에서 Identifier로 인식된 string이 tokenString 변수에 의해 정상적으로

읽어 들여지지 않는 오류가 있었습니다. 예로 "int a[10];"라는 declaration이 있다면, ID token에서의 tokenString이 "a"가 아닌 "["을 가리키고 있는 것입니다. 이는 yacc parser가 LALR 방식으로 작동하여 lookahead symbol을 읽어오는 과정에서 현재의 tokenString이 lookahead symbol에 의해 덮어 쓰였기 때문이라 추측하였고, 이에 tokenString을 문자열의 배열로 구성하여 lookahead symbol을 읽더라도 tokenString이 덮어 쓰이지 않게 구성하였습니다.

```
14
15    /* lexeme of identifier or reserved word */
16    int current;
17    char tokenString[2][MAXTOKENLEN+1];
18    %}
```

[그림 2] cmius.l 내부의 tokenString 문자열

```
82    currentToken = _yylex();
83    current = 1 - current;
84    strncpy(tokenString[current],yytext,MAXTOKENLEN);
85    if (TraceScan) {
```

[그림 3] cminus.l, getToken 함수 내의 tokenString 저장 방식. (current = 0으로 초기화)

3. TINY Compiler에서 CMinus로의 yacc parser 변경점

1) Global: DeclKind 추가

Globals는 프로그램 전체에서 사용하는 전역변수, enumeration 등을 관리하고 있습니다. 기존의 TINY compiler는 언어 스펙에서 선언과 정의를 구별하지 않기 때문에 Statement와 declaration 두가지의 kind만으로 구성되어 있었습니다. 하지만 C-Minus로 넘어오면서 선언과 정의를 구별하기 시작했고, 이에 따라 tree node의 kind를 설정할 때 선언을 분리하여 설정할 수 있도록 구성하였습니다.

```
64
65    typedef enum {DeclK,StmtK,ExpK} NodeKind;
66    typedef enum {ParamK,VarK,FnK} DeclKind;
67    typedef enum {CompK,IfK,WhileK,ReturnK} StmtKind;
68    typedef enum {AssignK,OpK,ConstK,IdK,CallK,IdxK} ExpKind;
69
```

[그림 4] globals.h 내부의 tree kind 관련 enumerations.

2) savedName의 Stack 구성

flex/yacc의 구성은 token을 scanning 하면서 곧장 parsing을 진행하기 때문에, lookahead에 따라 parsing policy가 구성된 후에는 이전에 읽었던 identifier의 tokenString이 무엇인지 알기 어렵습니다. 이에 따라 TINY compiler에서는 savedName 변수를 두어 ID token이 읽히면 parsing policy를 정하기 전에 미리 savedName 변수에 값을 저장해둡니다. 그리고 이는 reduce parsing이 정해졌을 때 savedName에서 tokenString을 복원하여 어떤 identifier가 쓰였는지 확인하게 됩니다.

하지만 TINY compiler에서는 single string을 savedName으로 구성하였고, 이는 "input(x[i])"와 같이 중첩된 identifier가 입력으로 들어온 상황에서 이전에 읽은 tokenString을 나중에 읽은 tokenString으로 덮어쓰는 상황이 발생합니다. 이 경우 정확한 identifier의 파악이 어려워집니다.

이를 해결하기 위해 C-Minus parser에서는 savedName을 Stack으로 구성하였고, 중첩된 identifier가 입력으로 들어오더라도 개개의 tokenString을 보존할 수 있도록 구성하였습니다.

```
15  #define MAXNAMESAVING 30
16  static char * savedName[MAXNAMESAVING]; /* for use in assignments */
17  static int nameidx;
18  static int savedNum;      /* for use in array assignments */
```

[그림 5] cminus.y 내부 savedName 배열

```
52  var_decl    : type_spec
53                ID { savedName[nameidx++] = copyString(tokenString[1 - current]
54                     savedLineNo = lineno; }
55                SEMI
56                  { $$ = newDeclNode(VarK);
57                    $$->attr.name = savedName[--nameidx];
58                    $$->lineno = savedLineNo;
```

[그림 6] cminus.y 내부 savedName의 stack 구조로의 활용

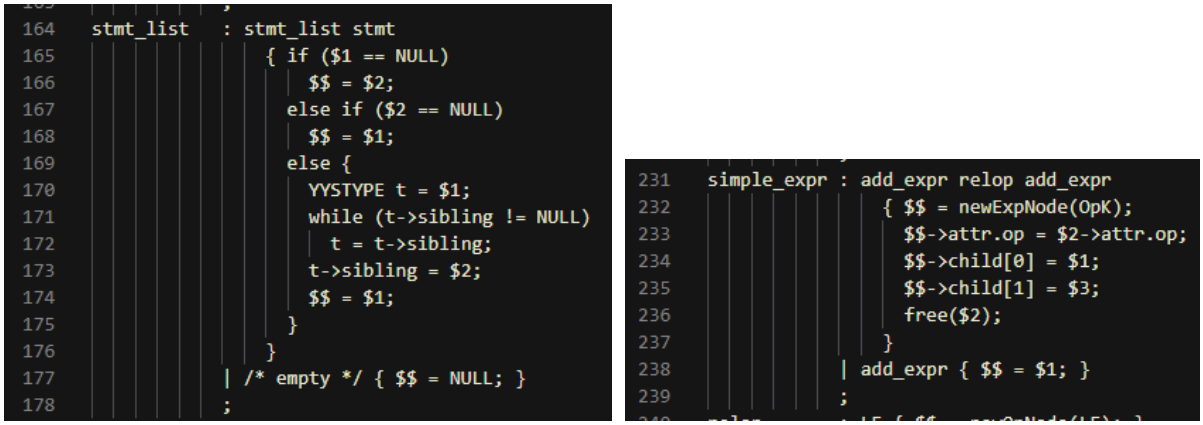
3) Sibling과 Child의 적절한 활용

TreeNode에는 Sibling과 Child라는 변수로 근방의 node를 관리하고 있습니다. Sibling의 경우 동일한 수준에 존재하는 statements 혹은 declarations로, 시간 순서에 따라 나열된 동일 선상의 구문들이 linkedlist의 형태로 이어지게 됩니다. 대표적인 예로 declaration

list와 parameter list, statement list가 있습니다.

Child의 경우는 명확한 상하 관계가 있음을 이야기합니다. If와 while 같이 조건문의 expression과 몸체에 해당하는 compound statements는 명확히 두 statement에 종속된 관계이고, 2항 연산 +,-,*,/ 의 경우 좌우의 두 operand가 하나의 operation에 종속된 구조로 나타납니다.

Sibling과 Child를 적절히 활용하여 syntax tree를 구성할 수 있도록 하였습니다.



```
164   stmt_list   : stmt_list stmt
165                 { if ($1 == NULL)
166                     $$ = $2;
167                   else if ($2 == NULL)
168                     $$ = $1;
169                   else {
170                     YYSTYPE t = $1;
171                     while (t->sibling != NULL)
172                       t = t->sibling;
173                     t->sibling = $2;
174                     $$ = $1;
175                   }
176                 }
177               | /* empty */ { $$ = NULL; }
178               ;
```

```
231   simple_expr : add_expr relop add_expr
232                 { $$ = newExpNode(OpK);
233                   $$->attr.op = $2->attr.op;
234                   $$->child[0] = $1;
235                   $$->child[1] = $3;
236                   free($2);
237                 }
238               | add_expr { $$ = $1; }
239               ;
```

[그림 7] cminus.y 내부 sibling 관계(좌)와 child 관계(우)

그 외의 부분은 yacc specification에 속하거나, 구현상 trivial한 design choice라 생각하여 자세한 설명은 코드와 코드 내 주석으로 남기겠습니다.

4. Samples

"./2__Parser/samples"에 test.cm, test2.cm, test3.cm을 두었습니다. 처음 두 개는 "1_Scanner" 과제에서 제시된 sample이며, 부가적인 parsing 여부를 검증하기 위해 부족한 부분은 세번째 "test3.cm"의 결과에 담았습니다.

아래는 GCD sample에 대한 간단한 실행 결과입니다.

```
C-MINUS COMPILATION: samples/test.cm

Syntax tree:
  Function declaration, name : gcd, return type: int
    Single parameter, name : u, type : int
    Single parameter, name : v, type : int
    Compund Statement :
      If (condition) (body) (else)
        Op : ==
          Id : v
          Const : 0
        Return :
          Id : u
        Return :
          Call, name : gcd, with arguments below
            Id : v
            Op : -
              Id : u
              Op : *
                Op : /
                  Id : u
                  Id : v
                Id : v
```

[그림 8] GCD test 샘플