Compiler project

2. Parser

2015004493 김형순

프로젝트 개요

- C minus 라는 프로그래밍의 컴파일러를 만들기 위한 두번째 단계인 parser 를 만든다.
- Yacc (Bison) 을 이용하여 CFG 를 정의 하고, AST 를 만드는 rule 을 정의 하여 parsing 과 함께 AST 를 생성한다.

컴파일 환경

- GCC 4.8.2 버전을 사용하여 컴파일 하였다.
- Ubuntu 14.04 LTS 환경에서 컴파일 및 실행 하였다.
- flex 2.5.35, bison 3.0.2 를 사용하였다.

실행 방법

- Makefile 을 이용하여 컴파일을 한다.
- cminus 라는 실행파일이 만들어지면 ./cminus [input file] 을 입력하여 실행한다.

구현방법

1. globals.h
   Parsing 의 결과가 될 node 를 만들어주기 위해 stmtKind enum type 에 FunK, CompK, If1K, IfElseK, WhileK, ReturnK 와 expKind enum type 에 VarK, AvarK, SparamK, AparamK, AssignK, IdK, AidK, OpK, ConstK, CallK 를 선언한다.
2. cminus.l
   Parsing 후 AST 에 ID 를 저장하기 위해 prevString 을 global 변수로 선언 하고, scanning 을 하며 ID 를 읽었을 때는 prevString 에 저장하여 parsing 시 이름 식별에 사용 할 수 있도록 하였다.
3. util.c
   Parsing 후 생성 한 AST 를 출력하기 위해 printTree 함수에 각 node 마다 출력하는 메시지를 다르게 해주었다. Variable 이나 function 같이 이름과 type 을 갖고있는 node 는 이름과 type 을 출력해주었고, const 같이 값을 갖고 있는 node 는 값을 출력해 주었다.

4. cminus.y

- Definition
- CFG 에 사용할 token 들을 정의한다. Scanner 에서 사용한 token 들을 그대로 사용하면 되는데, IF, ELSE, WHILE, RETURN, VOID, INT, ASSIGN EQ, NE, PLUS, MINUS, TIMES, OVER, LPAREN, RPAREN, LCURLY, RCURLY, LBRACE, RBRACE, LT, GT, LE, GE, COMMA, SEMI 가 그 token 들이다.
- Dangling else problem 을 해결하기 위해 ELSE 가 있는 If 문 과 와 ELSE 가 없는 IF 문의 우선순위를 구분하기 위해 ELSE 보다 낮은 우선순위의 pseudo token LT_ELSE 를 nonassoc 으로 정의한다. 그후, ELSE 의 우선순위를 높히기 위해 마지막에 ELSE 를 nonassoc 으로 재정의 한다
- Rules
- C – minus 문법에 따라 CFG 를 정의한다. 기본적으로 PDF 에 있는 grammar 를 참고하였다.
- CFG 의 예시를 몇개 들어보면 다음과 같다.
    - var_dclar (변수 선언) -> type_specif ID SEMI | type_specif ID LBRACE NUM RBRACE SEMI
    - fun_dclar (함수 선언) -> type_specif ID LPAREN params RPAREN comp_stmt
    - selection_stmt -> IF LPAREN exp RPAREN stmt | IF LPAREN exp RPAREN stmt ELSE stmt
    - var (변수 사용) -> ID | ID LBRACE exp RBRACE
    - call (함수 call) -> ID LPAREN args RPAREN
- CFG 를 모두 정의 한 후 각 rule 에서 reduce 를 함과 동시에 AST 를 만들 수 있도록 rule 을 정의한다. 각 rule 에서 새로운 node 를 만들어야 할지, node 들의 child 는 어떻게 구성 할 지, node 의 종류 과 이름, 값, operation 등은 무엇인지를 저장 할 수 있도록 한다.
- AST 를 만드는 rule 까지 예시를 들어보면 다음과 같다. (다음 장에 있습니다.)

```
var_dclar   : type_specif
             ID { $$ = newExpNode(VarK);
                // VarK 타입의 node 생성
                  $$->type = savedType;
              // type_specif 에서 저장한 type 을 node 의 type으로 설정
                  $$->attr.name = copyString(prevString);
              // 바로 전에 읽은 token 인 ID 를 node 의 name 으로 설정
                  $$->lineno = lineno;
              // 현재 line 의 번호를 저장
                 }
              SEMI
             | type_specif ID
                 { $$ = newExpNode(AvarK);
                // AvarK 타입의 node 생성
                  $$->type = savedType;
              // type_specif 에서 저장한 type 을 node 의 type으로 설정
                  $$->child[0] = newExpNode(ConstK);
              // 현재 node 의 첫번째 child 를 ConstK 타입의 node 로 생성
                  $$->attr.name = copyString(prevString);
              // 바로 전에 읽은 token 인 ID 를 node 의 name 으로 설정
                  $$->lineno = lineno;
              // 현재 line 의 번호를 저장
                 }
              LBRACE NUM
                 { $$ = $3;
          // 현재 node 를 3번째 (중괄호 내) 에서 만든 node 를 가리키도록 지정
                  $$->child[0]->attr.val = atoi(tokenString);
          // 마지막으로 읽은 token Num 을 첫번째 child 의 값으로 설정
                 }
              RBRACE SEMI
                 { $$ = $3; }
// reduce 된 node 가 3번째 (중괄호 내) 에서 만든 node 를 가리키도록 지정
             ;
```

```
param_list  : param_list COMMA param
                { YYSTYPE t = $1;
                  if (t != NULL)
                  { while (t->sibling != NULL)
                       t = t->sibling;
                    t->sibling = $3;
// 첫번째 nonterminal의 sibling 을 세번째 nonterminal 로 만들어
// 모든 param 이 sibling 이 되도록 한다.
                    $$ = $1; }
                    else $$ = $3;
                }
            | param
                { $$ = $1; }
          // 첫번째 nonterminal node 가 현재 node 가 되도록
            ;
```

```
selec_stmt   : IF LPAREN exp RPAREN stmt
                { $$ = newStmtNode(If1K);
                // If1K 타입의 node 생성
                  $$->child[0] = $3;
                  $$->child[1] = $5;
          // 첫번째, 두번째 child 를 세번째, 다섯번째 nonterminal 로 설정
                 } %prec LT_ELSE;
          // 이 rule 을 LT_ELSE 의 우선순위를 갖도록 설정
             | IF LPAREN exp RPAREN stmt ELSE stmt
                { $$ = newStmtNode(IfElseK);
                // IfElseK 타입의 node 생성
                  $$->child[0] = $3;
                  $$->child[1] = $5;
                  $$->child[2] = $7;
          // 첫번째, 두번째, 세번째 child 를 세번째, 다섯번째, 일곱번째
          //nonterminal 로 설정
                 }
     // 이 rule 은 위 rule 과 비교 했을 때 다음 token 으로 ELSE 가 오므로
     // ELSE 와 같은 우선순위를 갖게 된다.
             ;
```

실행결과

1) PDF 에 올라와 있는 example (test.cm)

- 파일 정보

```
parallels@ubuntu:~/Desktop/compiler/git/2_Parser$ cat test.cm
/* A program to perform Euclid's
  Algorithm to computer gcd */

int gcd (int u, int v)
{
    if (v == 0) return u;
    else return gcd(v,u-u/v*v);
     /* u-u/v*v == u mod v */
}


void main(void)
{
    int x; int y;
    x = input(); y = input(); output(gcd(x,y));
}
```

- Parsing 결과

```
parallels@ubuntu:~/Desktop/compiler/git/2_Parser$ ./cminus test.cm

C-minus COMPILATION: test.cm

Syntax tree:
  Function declaration, name:gcd, return type: int
    Single parameter: u, type: int
    Single parameter: v, type: int
    Compound statement:
      If (condition) (body) (else)
        Op: ==
          Variable: v
          Const: 0
        Return:
          Variable: u
        Return:
          Call, name: gcd, with arguments below
            Variable: v
            Op: -
              Variable: u
              Op: *
                Op: /
                  Variable: u
                  Variable: v
                Variable: v
  Function declaration, name:main, return type: void
    Compound statement:
      Variable declaration: x, type: int
      Variable declaration: y, type: int
      Assign (destination) (source)
        Variable: x
        Call, name: input, with arguments below
      Assign (destination) (source)
        Variable: y
        Call, name: input, with arguments below
      Call, name: output, with arguments below
        Call, name: gcd, with arguments below
          Variable: x
          Variable: y
```

- PDF 의 예시와 거의 동일하게 AST 를 만들 수 있었다. 다른 점은 PDF 에서 ID 라고 출력한 부분을 변수를 사용하는 부분이므로 Variable 이라고 출력 해주었고, 함수 선언 시 parameter 로 void 를 넣었을 때 PDF 에서는 single parameter 인 void 를 받은 것으로 했지만, void 를 받은것은 parameter 를 안받은것과 똑같다 생각하여 parameter 를 아예 출력하지 않게 하였다.

2) 추가적인 test (test.c)

- test.cm 으로는 모든 경우가 다 test 되는것 같지 않아 추가적인 test 용 code 를 만들었다.
- 배열을 parameter, 변수 선언, 변수 사용, function call 인자 로 주었을 때 잘 parsing 이 되는지 확인 하였다.
- 여러번의 사칙연산이 같이 나올 때 잘 parsing 하는지 확인 하였다.
- 배열 변수의 index 로 배열 변수를 사용 했을 때 parsing 이 잘 되고, 이름도 잘 저장하는지 확인 하였다.
- 파일 정보

```
parallels@ubuntu:~/Desktop/compiler/git/2_Parser$ cat test.c
int hello(int a[]){
  int z[10];
  1+1+1/2*3+1-1;
  x=1;
  z[5] = 1;
  a[b[c[i]]] = 2;
  delete(z[1],a[b[c[1]]]);

  return 1;
}
```

- Parsing 결과

```
C-minus COMPILATION: test.c

Syntax tree:
  Function declaration, name:hello, return type: int
    Array Parameter: a, type: int
    Compound statement:
      Array variable declaration: z, type: , with size below
        Const: 10
      Op: -
        Op: +
          Op: +
            Op: +
              Const: 1
              Const: 1
            Op: *
              Op: /
                Const: 1
                Const: 2
              Const: 3
          Const: 1
        Const: 1
      Assign (destination) (source)
        Variable: x
        Const: 1
      Assign (destination) (source)
        Array: z, with index below
          Const: 5
        Const: 1
      Assign (destination) (source)
        Array: a, with index below
          Array: b, with index below
            Array: c, with index below
              Variable: i
        Const: 2
      Call, name: delete, with arguments below
        Array: z, with index below
          Const: 1
        Array: a, with index below
          Array: b, with index below
            Array: c, with index below
              Const: 1
      Return:
        Const: 1
```

- 배열 변수를 어느곳에 사용하여도 잘 parsing 하는것을 확인 하였다.
- 배열을 선언 할때는 배열의 크기, 배열을 사용 할 때는 배열의 index 를 잘 출력 하였다.
- 배열 변수의 index 로 배열 변수를 사용 했을 때도 parsing 이 잘 되는것을 볼 수 있다.
- 여러번의 사칙 연산을 사용 하였을 때 우선순위에 맞게 잘 parsing 하는것을 볼 수 있다.

3) Dangling else problem

- Dangling else problem 이 잘 해결 되는지도 확인 해 보았다.
- 파일 정보

```
parallels@ubuntu:~/Desktop/compiler/git/2_Parser$ cat test3.c
void main(void){
    if(a==1)
        if(b==2)
            if(c==3)
                x=1;
            else{
                x=2;
            }
}
```

- Parsing 결과

```
parallels@ubuntu:~/Desktop/compiler/git/2_Parser$ ./cminus test3.c

C-minus COMPILATION: test3.c

Syntax tree:
  Function declaration, name:main, return type: void
    Compound statement:
      If (condition) (body)
        Op: ==
          Variable: a
          Const: 1
        If (condition) (body)
          Op: ==
            Variable: b
            Const: 2
          If (condition) (body) (else)
            Op: ==
              Variable: c
              Const: 3
            Assign (destination) (source)
              Variable: x
              Const: 1
            Compound statement:
              Assign (destination) (source)
                Variable: x
                Const: 2
```

- Dangling else problem 이 잘 해결 된것을 확인 하였다.

4) test.2.txt

- 마지막으로 지난 과제의 예시로 주어졌던 test.2.txt 의 parsing 결과도 첨부 한다.
- 파일 정보

```
parallels@ubuntu:~/Desktop/compiler/git/2_Parser$ cat test.2.txt
void main(int x[], int y)
{
        int i; int x[5];

        i = 0;
        while( i < 5 )
        {
                x[i] = input();

                i = i + 1;
        }

        i = 0;
        while( i <= 4 )
        {
                if( x[i] != 0 )
                {
                        output(x[i]);
            output();
            output(z[i]);
                }
        else{
            x = 2;
        }
        if (x == 2)
            y = 1;
        }
    return;
}
```

- Parsing 결과

```
parallels@ubuntu:~/Desktop/compiler/git/2_Parser$ ./cminus test.2.txt

C-minus COMPILATION: test.2.txt

Syntax tree:
  Function declaration, name:main, return type: void
    Array Parameter: x, type: int
    Single parameter: y, type: int
    Compound statement:
      Variable declaration: i, type: int
      Array variable declaration: x, type: int, with size below
        Const: 5
      Assign (destination) (source)
        Variable: i
        Const: 0
      While (condition) (body)
        Op: <
          Variable: i
          Const: 5
        Compound statement:
          Assign (destination) (source)
            Array: x, with index below
              Variable: i
            Call, name: input, with arguments below
          Assign (destination) (source)
            Variable: i
            Op: +
              Variable: i
              Const: 1
      Assign (destination) (source)
        Variable: i
        Const: 0
      While (condition) (body)
        Op: <=
          Variable: i
          Const: 4
        Compound statement:
          If (condition) (body) (else)
            Op: !=
              Array: x, with index below
                Variable: i
              Const: 0
            Compound statement:
              Call, name: output, with arguments below
                Array: x, with index below
                  Variable: i
              Call, name: output, with arguments below
              Call, name: output, with arguments below
                Array: z, with index below
                  Variable: i
            Compound statement:
              Assign (destination) (source)
                Variable: x
                Const: 2
          If (condition) (body)
            Op: ==
              Variable: x
              Const: 2
            Assign (destination) (source)
              Variable: y
              Const: 1
      Return:
parallels@ubuntu:~/Desktop/compiler/git/2_Parser$
```