Compiler Parser 보고서

2012003686
임준수

1. 컴파일 방법 및 환경

<Ubuntu 14.04>

<GCC compiler>

2. 파서 구현 과정 및 주요 소스코드 설명

Modified files

- main.c
- globals.h
- cminus.y
- util.c
- util.h

1) main.c

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

해당 Macro 에 대하여 FALSE 를 줘야 parsing 을 실행한다.

```
/* allocate and set tracing flags */
int EchoSource = TRUE;
int TraceScan = FALSE;
int TraceParse = TRUE;
```

해당 value 를 True 로 해줘야지 parsing 출력 결과를 확인 할 수 있다.

2) globals.h

```
Syntax tree:
  Function declaration, name : gcd, return type : int
    Single parameter, name : u, type : int
    Single parameter, name : v, type : int
    Compound 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
  Function declaration, name : main, return type : void
    Single parameter, name : (null), type : void
    Compound statement :
      Var declaration, name : x, type : int
      Var declaration, name : y, type : int
      Assign : (destination) (source)
        Id : x
        Call, name : input, with arguments below
      Assign : (destination) (source)
        Id : y
        Call, name : input, with arguments below
      Call, name : output, with arguments below
        Call, name : gcd, with arguments below
          Id : x
          Id : y
```

과제 설명 ppt 에 다음과 같은 출력 예시가 있다.
이 출력 결과와 C-minus 문법을 통해서 우리는 자료구조를 어떻게 만들어야 하는지 도출 해 낼 수 있다. 그 구조와 밑에 코드에 서술 되어있다.

```
typedef enum {StmtK,ExpK,DeclK,ParamK,TypeK} NodeKind;
typedef enum {CompK,IfK,IterK,RetK} StmtKind;
typedef enum {AssignK,OpK,ConstK,IdK,ArrIdK,CallK} ExpKind;
typedef enum {FuncK,VarK,ArrVarK} DeclKind;
typedef enum {ArrParamK,NonArrParamK} ParamKind;
typedef enum {TypeNameK} TypeKind;

/* ArrayAttr is used for attributes for array variables */
typedef struct arrayAttr {
    TokenType type;
    char * name;
    int size;
} ArrayAttr;

/* ExpType is used for type checking */
typedef enum {Void,Integer,Boolean, IntegerArray} ExpType;
```

```
typedef struct treeNode
   { struct treeNode * child[MAXCHILDREN];
     struct treeNode * sibling;
     int lineno;
     NodeKind nodekind;
     union { StmtKind stmt;
             ExpKind exp;
             DeclKind decl;
             ParamKind param;
             TypeKind type; } kind;
     union { TokenType op;
             TokenType type;
             int val;
             char * name;
             ArrayAttr arr;
             struct ScopeRec * scope; } attr;
     ExpType type; /* for type checking of exps */
   } TreeNode;
```

Cminus 는 tiny 보다 많이 attr과 여러 종류의 statement 를 가지고 있기 때문에 Cminus 문법에 맞게 위와 같이 treeNode 를 재정의 해준다.

3) cminus.y

이 파일에서 실제적 cminus parsing rule 이 명시된다. 이 rule 을 통해 parse tree 가 만들어지고 parsing 이 이루어진다.

1. *program* → *declaration-list*
2. *declaration-list* → *declaration-list declaration* | *declaration*
3. *declaration* → *var-declaration* | *fun-declaration*
4. *var-declaration* → *type-specifier* **ID ;** | *type-specifier* **ID [NUM] ;**
5. *type-specifier* → **int** | **void**
6. *fun-declaration* → *type-specifier* **ID (** *params* **)** *compound-stmt*
7. *params* → *param-list* | **void**
8. *param-list* → *param-list* **,** *param* | *param*
9. *param* → *type-specifier* **ID** | *type-specifier* **ID [ ]**
10. *compound-stmt* → **{** *local-declarations statement-list* **}**
11. *local-declarations* → *local-declarations var-declarations* | *empty*
12. *statement-list* → *statement-list statement* | *empty*
13. *statement* → *expression-stmt* | *compound-stmt* | *selection-stmt* | *iteration-stmt* | *return-stmt*
14. *expression-stmt* → *expression* **;** | **;**
15. *selection-stmt* → **if (** *expression* **)** *statement* | **if (** *expression* **)** *statement* **else** *statement*
16. *iteration-stmt* → **while (** *expression* **)** *statement*
17. *return-stmt* → **return ;** | **return** *expression* **;**
18. *expression* → *var* **=** *expression* | *simple-expression*
19. *var* → **ID** | **ID [** *expression* **]**
20. *simple-expression* → *additive-expression relop additive-expression* | *additive-expression*
21. *relop* → **<=** | **<** | **>** | **>=** | **==** | **!=**
22. *additive-expression* → *additive-expression addop term* | *term*
23. *addop* → **+** | **-**
24. *term* → *term mulop factor* | *factor*
25. *mulop* → ***** | **/**
26. *factor* → **(** *expression* **)** | *var* | *call* | **NUM**
27. *call* → **ID (** *args* **)**
28. *args* → *arg-list* | *empty*
29. *arg-list* → *arg-list* **,** *expression* | *expression*

위에 쓰여진 c-minus 문법을 보고 parse tree 를 만들었다.

4) util.c

```
void printTree( TreeNode * tree )
{ int i;
  int tmp;
  char type_name[10];
  INDENT;
  while (tree != NULL) {
    printSpaces();
    if (tree->nodekind==StmtK)
    { switch (tree->kind.stmt) {
        case CompK:
          fprintf(listing,"Compound Statment\n");
          break;
        case IfK:
          fprintf(listing,"If (condition) (body) (else)\n");
          break;
        case IterK:
          fprintf(listing,"Repeat\n");
          break;
        case RetK:
          fprintf(listing,"Return\n");
          break;
        default:
          fprintf(listing,"Unknown ExpNode kind\n");
          break;
      }
    }
    else if (tree->nodekind==ExpK)
    { switch (tree->kind.exp) {
        case AssignK:
          fprintf(listing,"Assign: (destination) (source)");
          //printToken(tree->attr.op,"\0");
          break;
        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 ArrIdK:
          fprintf(listing,"ArrId\n");
          break;
        case CallK:
          fprintf(listing,"Call, name : %s, with arguments below\n", tree->attr.name);
          break;
        default:
          fprintf(listing,"Unknown ExpNode kind\n");
          break;
      }
    }

    else if (tree->nodekind==DeclK)
    { switch (tree->kind.decl) {
        case FuncK:
          tmp = tree->child[0]->attr.type;
          if (tmp == INT)
            fprintf(listing,"Function Declaration, name : %s, return type : int\n",tree->attr.name);
          else if (tmp == VOID)
            fprintf(listing,"Function Declaration, name : %s, return type : void\n",tree->attr.name);
          tree->child[0] = NULL; // child[0] : type
          break;
        case VarK:
          tmp = tree->child[0]->attr.type;
          if (tmp == INT)
            fprintf(listing,"Var Declaration, name : %s, type : int\n",tree->attr.name);
          else if (tmp == VOID)
            fprintf(listing,"Var Declaration, name : %s, type : void\n",tree->attr.name);
          tree->child[0] = NULL;
          break;
        case ArrVarK:
          fprintf(listing,
                  "Var Dec(following const:array length): %s %d\n",
                  tree->attr.arr.name,
                  tree->attr.arr.size);
          break;
        default:
          fprintf(listing,"Unknown DeclNode kind\n");
          break;
      }
    }
    else if (tree->nodekind==ParamK)
    { switch (tree->kind.param) {
        case ArrParamK:
          fprintf(listing,"Array Parameter: %s\n",tree->attr.name);
          break;
        case NonArrParamK:
          tmp = tree->child[0]->attr.type;
          if (tmp == INT)
            fprintf(listing,"Single Parameter, name : %s, type : int\n",tree->attr.name);
          if (tmp == VOID)
            fprintf(listing,"Single Parameter, name : %s, type : void\n",tree->attr.name);
          tree->child[0] = NULL;
          break;
        default:
          fprintf(listing,"Unknown ParamNode kind\n");
          break;
      }
    }
    else if (tree->nodekind==TypeK)
    { switch (tree->kind.type) {
        case TypeNameK:
          fprintf(listing,"Type: ");
          switch (tree->attr.type) {
            case INT:
```

위의 코드들은 yacc 를 이용해 만들어진 parse tree 의 정보에 대하여 출력하는 코드이다. 첫번째로는 nodeKind 에 대해 찾아간다. 해당 node 에서 어떤 kindType 인지 확인하여. 이에 맞는 출력 결과를 뱉어준다.

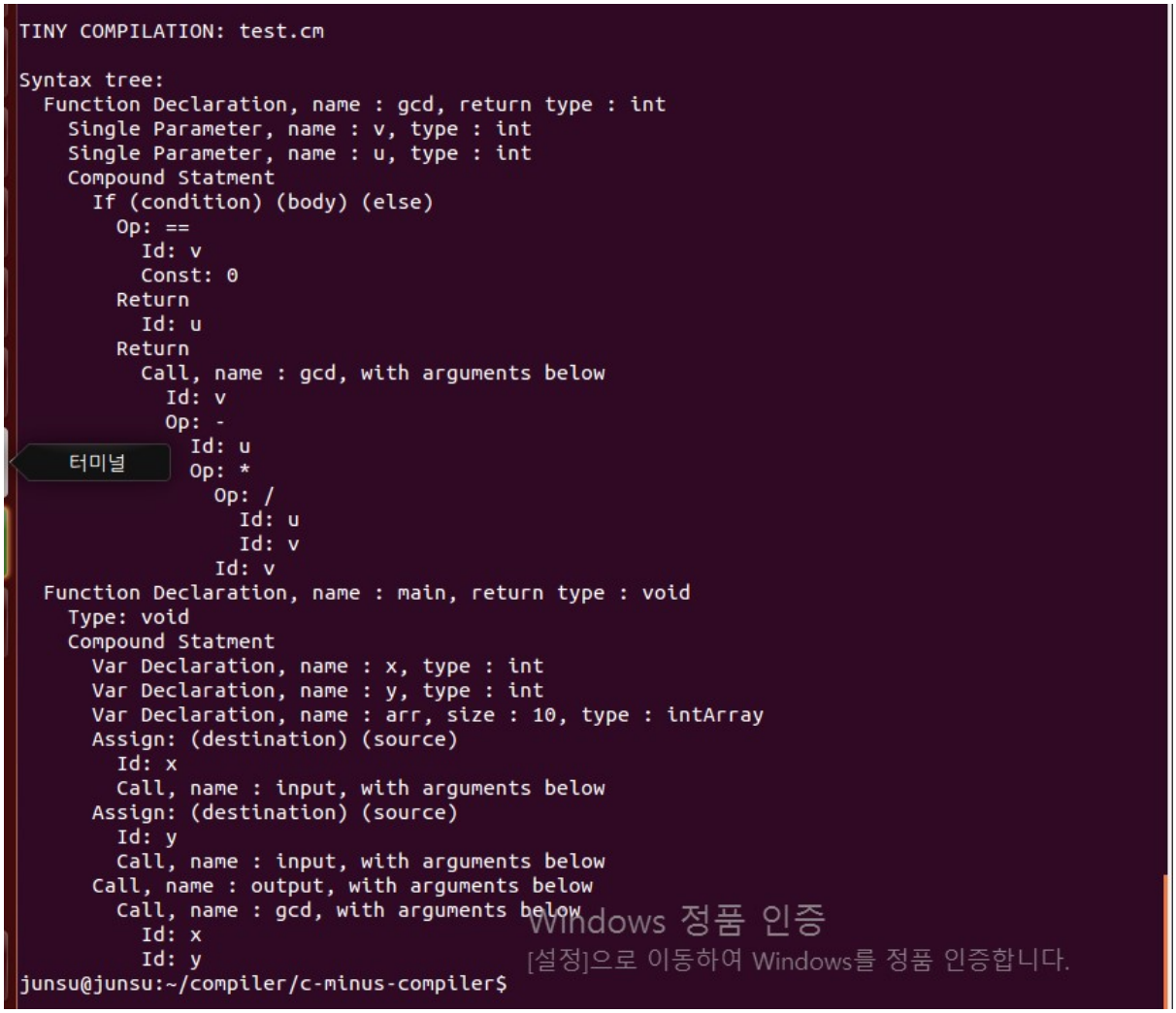
이 후 child가 있다면 child 를 탐색 하고 sibling 이 있다면 sibling 을 탐색해서 결과를 출력해준다.

5) util.h

새로운 nodeKind 와 node 들이 생겼기 때문에 header file 에 해당 목록들을 추가해준다.

```
1  /****************************************************/
2  /* File: util.h                                     */
3  /* Utility functions for the TINY compiler          */
4  /* Compiler Construction: Principles and Practice   */
5  /* Kenneth C. Louden                                */
6  /****************************************************/
7
8  #ifndef _UTIL_H_
9  #define _UTIL_H_
10
11 /* Procedure printToken prints a token
12  * and its lexeme to the listing file
13  */
14 void printToken( TokenType, const char* );
LibreOffice Calc
16 /* Function newStmtNode creates a new statement
17  * node for syntax tree construction
18  */
19 TreeNode * newStmtNode(StmtKind);
20
21 /* Function newExpNode creates a new expression
22  * node for syntax tree construction
23  */
24 TreeNode * newExpNode(ExpKind);
25
26 /* Function newParamNode creates a new declation
27  * node for syntax tree construction
28  */
29 TreeNode * newDeclNode(DeclKind);
30
31 /* Function newParamNode creates a new parameter
32  * node for syntax tree construction
33  */
34 TreeNode * newParamNode(ParamKind);
35
36 /* Function newTypeNode creates a new type
37  * node for syntax tree construction
38  */
39 TreeNode * newTypeNode(TypeKind);
40
41 /* Function copyString allocates and makes a new
42  * copy of an existing string
```

3. 예시 및 결과화면 (스크린 캡쳐)



Parsing 의 결과는 다음과 같이 출력된다. Arr 에 대한 설명은 IntegerArray를 test 하기 위해 내가 넣은 int arr[10]; 선언이다.