

Project 2: Parser

2016024875 손정우

1. Compilation methods and environment

To compile the project, I used Makefile example given by the TA. The environment that I build this project was as below:

- Ubuntu 16.04.7 (64-bits, on VMware Workstation)
- gcc 5.4.0 20160609
- flex 2.6.0
- GNU Bison 3.0.4

2. Implementation of C-parser

1) Makefile

I used the makefile given as an example in the project specification. However, source files not used in this project were excluded from creation of executable file *cminus*.

```
#OBS = main.o util.o lex.yy.o y.tab.o symtab.o analyze.o code.o cgen.o
OBS = main.o util.o lex.yy.o y.tab.o
```

2) globals.h

In order to implement given BNF Grammar, additionally necessary types were defined. Also, in the case of *attr*, which is a union type, since multiple variables in *attr* cannot be used at same time, a struct called *ArrayAttr* was declared and added. *ArrayAttr* holds data of array variable such as variable name, array size and type.

```
/* globals.h */

typedef enum {StmtK, DeclK, ExpK, ParamK} NodeKind;
typedef enum {CompK, SelK, IterK, RetK} StmtKind;
typedef enum {VarK, VarArrK, FunK} DeclKind;
typedef enum {AssignK, OpK, ConstK, IdK, IdArrK, CallK, CalcK, TypeNameK} ExpKind;
typedef enum {SingleParamK, ArrParamK} ParamKind;

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

#define MAXCHILDREN 3

typedef struct arrayAttr
{
    TokenType type;
    char * name;
    int size;
}
```

```

    } ArrayAttr;

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

```

3) util.c & util.h

New functions (*newDeclNode()*, *newParamNode()*) were added to express the newly added type in *globals.h* as a syntax tree. Also, the existing *printTree()* were modified to handle the added syntax tree.

4) cminus.y

The *cminus.y* file was implemented by modifying the *tiny.y* file. The main parts of implementing *cminus.y* are as follows.

In order to handle *ID* and *NUM* correctly, *inputName* and *inputNumber* have been added to the grammar rules. This is because when it is necessary to get a variable name or function name, only the last token is stored, so the previously stored token is blown away when the latter token is processed, making the identifier unreadable. To prevent this, these two rules have been added, and are used instead where *ID* and *NUM* are used.

```

inputName      : ID
                { savedName = copyString(tokenString); }
                ;
inputNumber    : NUM
                { savedNumber = atoi(tokenString); }
                ;

```

However, since *savedName* and *savedNumber* are global variables, there was a problem that data was overwritten when *inputName* was used again after using *inputName*. One of the problems caused by this was shown in the following figure. Clearly, it does not correctly handle multiple IDs.

```

Assign : (destination) (source)
IdArr  : testC, with array index below
IdArr  : testC, with array index below
IdArr  : testC, with array index below
Const  : 1
Const  : 1

```

Figure 1 When the parser met **testA[testB[testC[1]]] = 1** statement

To prevent this happening, some of the grammar rules for using *inputName* have been implemented as

follows. (Shift after using *inputName*)

[코드 추가]

```
var      : inputName
        { $$ = newExpNode(IdK);
          $$->attr.name = savedName;
        }
    | inputName
        { $$ = newExpNode(IdArrK);
          $$->attr.name = savedName;
        }
    LBACE expression RBACE
        { $$ = $2;
          $$->child[0] = $4;
        }
    ;

call     : inputName
        { $$ = newExpNode(CallK);
          $$->attr.name = savedName;
        }
    LPAREN args RPAREN
        { $$ = $2;
          $$->child[0] = $4; // Arguments
        }
    ;
```

- The priority of *selection_stmt* was specified using *%prec*.

There was a shift/reduce conflict in *selection_stmt*. To solve this, I tried various attempts such as implementing the unambiguous selection rule in yacc by referring to textbook (Compiler Construction, Louden, 1997). However, the shift/reduce conflict in yacc could not be resolved by making the syntax unambiguous. Here are one of the unambiguous grammars I've tried.

$$\begin{aligned} \text{statement} &\rightarrow \text{matched-stmt} \mid \text{unmatched-stmt} \\ \text{matched-stmt} &\rightarrow \text{if (exp) matched-stmt else matched-stmt} \mid \text{other} \\ \text{unmatched-stmt} &\rightarrow \text{if (exp) statement} \\ &\quad \mid \text{if (exp) matched-stmt else unmatched-stmt} \\ \text{exp} &\rightarrow 0 \mid 1 \end{aligned}$$

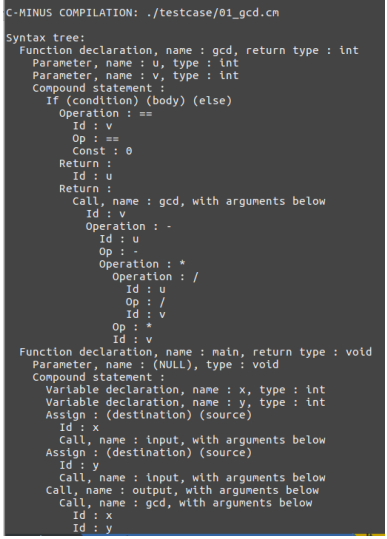
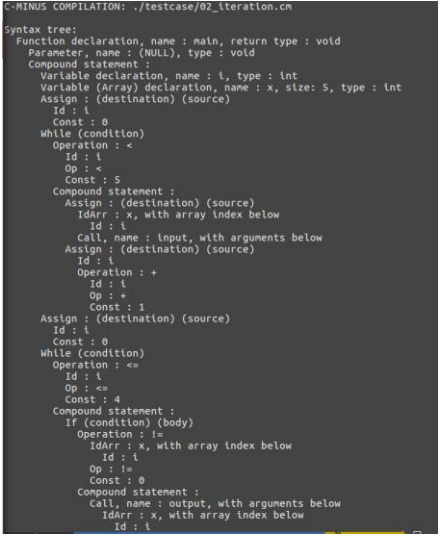
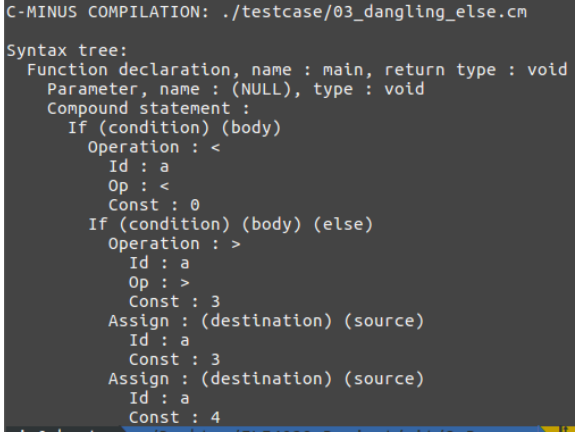
Figure 2 What I tried (Unambiguous selection statement by Louden)

This shift/reduce conflict problem was solved using bison's grammar rule *%prec* which explicitly prioritizes grammar.

3. Example Results

In the previous project, there was a rudimentary mistake, such as using the wrong direction of the inequality sign, but I did not know this because I neglected the test. As a result, there was a catastrophe in the project score.

Through the last experience, I made an additional test case in this project to check whether the compiler's parser works properly.

Description	Source Code	Parsing Result
gcd	<pre> /* 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)); } </pre>	
Iteration	<pre> void main(void) { 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]); } } } </pre>	
Dangling else	<pre> /* dangling else example */ void main (void) { if(a < 0) if(a > 3) a = 3; else a = 4; } </pre>	

<p>Semantic error</p>	<pre>int main (void) { int a; int b; c = a + b; }</pre>	<p>C-MINUS COMPILATION: ./testcase/04_semantic_error.cm</p> <p>Syntax tree:</p> <p>Function declaration, name : main, return type : int</p> <p>Parameter, name : (NULL), type : void</p> <p>Compound statement :</p> <p>Variable declaration, name : a, type : int</p> <p>Variable declaration, name : b, type : int</p> <p>Assign : (destination) (source)</p> <p>Id : c</p> <p>Operation : +</p> <p>Id : a</p> <p>Op : +</p> <p>Id : b</p>
<p>Selection sort</p>	<pre>/* Selection sort */ int data[10]; int minLoc (int parm[], int low, int high) { int i; int k; int t; k = low; x = parm[low]; t = low + 1; while (t < high) { if (parm[t] < x) { k = t; } t = t + 1; } return k; } void sort (int parm[], int low, int high) { int i; int k; t = low; while (t < high-1) { k = minLoc(parm, t, high); t = parm[k]; parm[k] = parm[t]; parm[t] = t; t = t + 1; } } void main (void) { int i; t = 0; while (t < 10) { data[t] = input(); t = t + 1; } sort(data, 0, 10); while (t < 10) { output(data[t]); t = t + 1; } }</pre>	
<p>Function call</p>	<pre>int functionA(int inputA) { inputA = inputA + 1; return inputA; } int functionB(int inputB) { inputB = inputB + 2; return inputB; } int functionC(int inputC) { inputC = inputC + 3; return inputC; } int main(void) { int i; i = 0; i = functionC(functionB(functionA(i))); return i; }</pre>	<p>C-MINUS COMPILATION: ./testcase/08_func_call.cm</p> <p>Syntax tree:</p> <p>Function declaration, name : functionA, return type : int</p> <p>Parameter, name : inputA, type : int</p> <p>Compound statement :</p> <p>Assign : (destination) (source)</p> <p>Id : inputA</p> <p>Operation : +</p> <p>Id : inputA</p> <p>Op : +</p> <p>Const : 1</p> <p>Return :</p> <p>Id : inputA</p> <p>Function declaration, name : functionB, return type : int</p> <p>Parameter, name : inputB, type : int</p> <p>Compound statement :</p> <p>Assign : (destination) (source)</p> <p>Id : inputB</p> <p>Operation : +</p> <p>Id : inputB</p> <p>Op : +</p> <p>Const : 2</p> <p>Return :</p> <p>Id : inputB</p> <p>Function declaration, name : functionC, return type : int</p> <p>Parameter, name : inputC, type : int</p> <p>Compound statement :</p> <p>Assign : (destination) (source)</p> <p>Id : inputC</p> <p>Operation : +</p> <p>Id : inputC</p> <p>Op : +</p> <p>Const : 3</p> <p>Return :</p> <p>Id : inputC</p> <p>Function declaration, name : main, return type : int</p> <p>Parameter, name : (NULL), type : void</p> <p>Compound statement :</p> <p>Variable declaration, name : i, type : int</p> <p>Assign : (destination) (source)</p> <p>Id : i</p> <p>Const : 0</p> <p>Assign : (destination) (source)</p> <p>Id : i</p> <p>Call, name : functionC, with arguments below</p> <p>Call, name : functionB, with arguments below</p> <p>Call, name : functionA, with arguments below</p> <p>Id : i</p> <p>Return :</p> <p>Id : i</p>
<p>*Note that some of the results have been omitted to meet the report specifications. (less than 5 pages)</p> <p>Although screenshots are hard to see, you can still check the code and result by enlarging the PDF file.</p>		