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*.

```
#OBJS = main.o util.o lex.yy.o y.tab.o symtab.o analyze.o code.o cgen.o
OBJS = 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;
                            }
                          LBRACE expression RBRACE
                            { $$ = $2;
                              $$->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.

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

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.

<table>
<tr><th>Description</th><th>Source Code</th><th>Parsing Result</th></tr>
<tr><td>gcd</td><td>

```c
/* 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));
}
```

</td><td>

```
C-MINUS COMPILATION: ./testcase/01_gcd.cm

Syntax tree:
  Function declaration, name : gcd, return type : int
    Parameter, name : u, type : int
    Parameter, name : v, type : int
    Compound statement :
      If (condition) (body) (else)
        Operation : ==
          Id : v
          Op : ==
          Const : 0
        Return :
          Id : u
        Return :
          Call, name : gcd, with arguments below
            Id : v
            Operation : -
              Id : u
              Op : -
              Operation : *
                Operation : /
                  Id : u
                  Op : /
                  Id : v
                Op : *
                Id : v
  Function declaration, name : main, return type : void
    Parameter, name : (NULL), type : void
    Compound statement :
      Variable declaration, name : x, type : int
      Variable 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
```

</td></tr>
<tr><td>Iteration</td><td>

```c
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]);
                }
        }
}
```

</td><td>

```
C-MINUS COMPILATION: ./testcase/02_Iteration.cm

Syntax tree:
  Function declaration, name : main, return type : void
    Parameter, name : (NULL), type : void
    Compound statement :
      Variable declaration, name : i, type : int
      Variable (Array) declaration, name : x, size: 5, type : int
      Assign : (destination) (source)
        Id : i
        Const : 0
      While (condition)
        Operation : <
          Id : i
          Op : <
          Const : 5
        Compound statement :
          Assign : (destination) (source)
            IdArr : x, with array index below
              Id : i
            Call, name : input, with arguments below
          Assign : (destination) (source)
            Id : i
            Operation : +
              Id : i
              Op : +
              Const : 1
      Assign : (destination) (source)
        Id : i
        Const : 0
      While (condition)
        Operation : <=
          Id : i
          Op : <=
          Const : 4
        Compound statement :
          If (condition) (body)
            Operation : !=
              IdArr : x, with array index below
                Id : i
              Op : !=
              Const : 0
            Compound statement :
              Call, name : output, with arguments below
                IdArr : x, with array index below
                  Id : i
```

</td></tr>
<tr><td>Dangling else</td><td>

```c
/* dangling else example */

void main (void) { if(a < 0) if(a > 3) a = 3; else a = 4; }
```

</td><td>

```
C-MINUS COMPILATION: ./testcase/03_dangling_else.cm

Syntax tree:
  Function declaration, name : main, return type : void
    Parameter, name : (NULL), type : void
    Compound statement :
      If (condition) (body)
        Operation : <
          Id : a
          Op : <
          Const : 0
        If (condition) (body) (else)
          Operation : >
            Id : a
            Op : >
            Const : 3
          Assign : (destination) (source)
            Id : a
            Const : 3
          Assign : (destination) (source)
            Id : a
            Const : 4
```

</td></tr>
</table>

<table>
<tr>
<td>Sementic error</td>
<td>

```c
int main ( void )
{
    int a;
    int b;
    c = a + b;
}
```

</td>
<td>

```
C-MINUS COMPILATION: ./testcase/04_semantic_error.cm

Syntax tree:
  Function declaration, name : main, return type : int
    Parameter, name : (NULL), type : void
    Compound statement :
      Variable declaration, name : a, type : int
      Variable declaration, name : b, type : int
      Assign : (destination) (source)
        Id : c
        Operation : +
          Id : a
          Op : +
          Id : b
```

</td>
</tr>
<tr>
<td>Selection sort</td>
<td>

```c
/* Selection sort */

int data[10];

int miniLoc ( int parm[], int low, int high ) {
    int i;
    int x;
    int k;
    k = low;
    x = parm[low];
    i = low + 1;

    while ( i < high ) {
        if ( parm[i] < x ) {
            x = parm[i];
            k = i;
        }
        i = i + 1;
    }

    return k;
}

void sort ( int parm[], int low, int high ) {
    int i;
    int k;
    i = low;

    while ( i < high-1 ) {
        int t;
        k = miniLoc(parm, i, high);
        t = parm[k];
        parm[k] = parm[i];
        parm[i] = t;
        i = i + 1;
    }
}

void main (void)
{
    int i;

    i = 0;
    while ( i < 10 ) {
        data[i] = input();
        i = i + 1;
    }

    sort(data, 0, 10);

    while ( i < 10 ) {
        output(data[i]);
        i = i + 1;
    }
```

</td>
<td>

</td>
</tr>
<tr>
<td>Function call</td>
<td>

```c
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;
}
```

</td>
<td>

```
C-MINUS COMPILATION: ./testcase/08_func_call.cm

Syntax tree:
  Function declaration, name : functionA, return type : int
    Parameter, name : inputA, type : int
    Compound statement :
      Assign : (destination) (source)
        Id : inputA
        Operation : +
          Id : inputA
          Op : +
          Const : 1
      Return :
        Id : inputA
  Function declaration, name : functionB, return type : int
    Parameter, name : inputB, type : int
    Compound statement :
      Assign : (destination) (source)
        Id : inputB
        Operation : +
          Id : inputB
          Op : +
          Const : 2
      Return :
        Id : inputB
  Function declaration, name : functionC, return type : int
    Parameter, name : inputC, type : int
    Compound statement :
      Assign : (destination) (source)
        Id : inputC
        Operation : +
          Id : inputC
          Op : +
          Const : 3
      Return :
        Id : inputC
  Function declaration, name : main, return type : int
    Parameter, name : (NULL), type : void
    Compound statement :
      Variable declaration, name : i, type : int
      Assign : (destination) (source)
        Id : i
        Const : 0
      Assign : (destination) (source)
        Id : i
        Call, name : functionC, with arguments below
          Call, name : functionB, with arguments below
            Call, name : functionA, with arguments below
              Id : i
      Return :
        Id : i
```

</td>
</tr>
<tr>
<td colspan="3">*Note that some of the results have been omitted to meet the report specifications. (less than 5 pages)
Although screenshots are hard to see, you can still check the code and result by enlarging the PDF file.</td>
</tr>
</table>