Syntax Analysis Parser using Bison : 2nd project of ‘Compiler’ lecture.

- StuId: **2022094839** Name: 노수찬

- IDE and System Environment: Docker ubuntu latest
- Compilation environment and method: ubuntu & make (gcc, flex, bison)

To implement the XIF mechanism and address the "dangling else" problem, the cminus.y file reflects a non-associative precedence for IF and ELSE using %nonassoc. Using this approach, each IF statement explicitly binds to its corresponding ELSE, thus preventing ambiguity. The selection statement rules are structured to handle both IF statements with and without an accompanying ELSE, ensuring that the correct logical structure is maintained, especially in nested conditions. This effectively resolves potential conflicts and ensures accurate parsing of conditional constructs.

In the implementation, the isArray attribute is utilized. For example, in the var_declaration rules, when a variable is declared with square brackets, the isArray attribute is set to 1, indicating that it is an array. Conversely, for standard variable declarations, isArray is set to 0. This differentiation allows the abstract syntax tree (AST) to accurately represent the data structure of variables, enabling the compiler to handle variable types.

In the design phase, some node kind and ‘decl’ kind have been appended in the global file. Using this struct treeNode, and other kind such as ExpKind, it is seamlessly conditioned by its kind. Under the implementation phase, using a parsing and print formatted while loop and switch cases block, it branches using structured attributes.

// globals.h

```
typedef enum {StmtK, ExpK, DeclK, TypeK} NodeKind;
typedef enum {IfK, WhileK, CallK, ReturnK, AssignK, CompK} StmtKind;
typedef enum {OpK, ConstK, IdK} ExpKind;
typedef enum {FunK, VarK, ParamK} DeclKind;
typedef enum {Void, Integer} TypeKind; // ... , Boolean // is it for sema
typedef enum {Function, Variable} IdentifierKind;
/* TypeKind is used for type checking */
        //typedef enum {Void,Integer,Boolean} ExpType;

#define MAXCHILDREN 3

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

Screenshot_1: Compile (using a following command, flex and bison are linked: make all)

```
root@b1fbc5679b3c:/workspace/basic# ls
Makefile   analyze.h  cgen.h    cminus.output  cminus.tab.h  code.c  globals.h  main.
analyze.c  cgen.c     cminus.l  cminus.tab.c   cminus.y      code.h  lex        parse.
root@b1fbc5679b3c:/workspace/basic# make clean
rm -vf cminus_parser *.o lex.yy.c y.tab.c y.tab.h y.output
root@b1fbc5679b3c:/workspace/basic# make all
yacc -d -v cminus.y
gcc -W -Wall -c main.c
gcc -W -Wall -c util.c
flex cminus.l
gcc -W -Wall -c lex.yy.c
gcc -W -Wall -c y.tab.c
gcc -W -Wall main.o util.o lex.yy.o y.tab.o -o cminus_parser -lfl
root@b1fbc5679b3c:/workspace/basic#
root@b1fbc5679b3c:/workspace/basic#
```

Test and example:

Test cases: From test.0.txt ~ ... ~ test.a.txt ~ ... to test.m.txt (through this path: test_cases/test.*.txt directory location)
additional comment/ (Except for test.1.txt and test.2.txt, all files have been newly created)
(Makefile is the same version with an given file. Nothing changed.)

Screenshots.…. (test.1.txt and test.2.txt)*

```
root@b1fbc5679b3c:/workspace/basic# ./cminus_parser test_cases/test.1.txt

C-MINUS COMPILATION: test_cases/test.1.txt

Syntax tree:
  Function Declaration: name = gcd, return type = int
    Parameter: name = u, type = int
    Parameter: name = v, type = int
    Compound Statement:
      If-Else Statement:
        Op: ==
          Variable: name = v
          Const: 0
        Return Statement:
          Variable: name = u
        Return Statement:
          Call: function name = gcd
            Variable: name = v
            Op: -
```

(continued)

```
            Variable: name = u
            Op: *
              Op: /
                Variable: name = u
                Variable: name = v
              Variable: name = v
  Function Declaration: name = main, return type = void
    Void Parameter
    Compound Statement:
      Variable Declaration: name = x, type = int
      Variable Declaration: name = y, type = int
      Assign:
        Variable: name = x
        Call: function name = input
      Assign:
        Variable: name = y
        Call: function name = input
      Call: function name = output
        Call: function name = gcd
          Variable: name = x
          Variable: name = y
root@b1fbc5679b3c:/workspace/basic#
```

*: In ubuntu bash terminal, using a following command its result can be obtained

./cminus_parser test_cases/test.1.txt

(To make it simple, it can be regarded **./cminus_parser test_cases/test.*.txt)**

```
C-MINUS COMPILATION: test_cases/test.2.txt

Syntax tree:
  Function Declaration: name = main, return type = void
    Void Parameter
    Compound Statement:
      Variable Declaration: name = i, type = int
      Variable Declaration: name = x, type = int[]
        Const: 5
      Assign:
        Variable: name = i
        Const: 0
```

(continued)

```
      While Statement:
        Op: <
          Variable: name = i
          Const: 5
        Compound Statement:
          Assign:
            Variable: name = x
              Variable: name = i
            Call: function name = input
          Assign:
            Variable: name = i
            Op: +
              Variable: name = i
              Const: 1
      Assign:
        Variable: name = i
        Const: 0
      While Statement:
        Op: <=
          Variable: name = i
          Const: 4
        Compound Statement:
          If Statement:
            Op: !=
              Variable: name = x
                Variable: name = i
              Const: 0
            Compound Statement:
              Call: function name = output
                Variable: name = x
                  Variable: name = i
root@b1fbc5679b3c:/workspace/basic#
```

(Test case customize) **(test.k.txt ~ in order to create a test case “if - else syntax test”)**

Screenshot (additional)

```
root@b1fbc5679b3c:/workspace/basic# cat -n test_cases/test.k.txt
     1  int main(){
     2  int a = 1;
     3  int b = 2;
     4  int c; // Declare c outside of the if statement
     5  int d; // Declare d outside of the if statement
     6
     7  if (a) {
     8      if (b) {
     9          c = 0; // Assign a value to c or perform some action
    10      }
    11  } else {
    12      d = 0; // Assign a value to d or perform some action
    13  }
    14  return 0;
    15  }
root@b1fbc5679b3c:/workspace/basic#
```

Parsing result of test.k.txt case

Screenshot

```
root@b1fbc5679b3c:/workspace/basic# ./cminus_parser test_cases/test.k.txt

C-MINUS COMPILATION: test_cases/test.k.txt
Error: syntax error

Syntax tree:
root@b1fbc5679b3c:/workspace/basic#
```

```
root@b1fbc5679b3c:/workspace/basic# cat -n test_cases/test.0.txt && ./cminus_parser test_cases/test.0.txt
     1  int main ( void a[] )
     2  {
     3      void b;
     4      int c;
     5      d[1] = b + c;
     6  }

C-MINUS COMPILATION: test_cases/test.0.txt

Syntax tree:
  Function Declaration: name = main, return type = int
    Parameter: name = a, type = void[]
    Compound Statement:
      Variable Declaration: name = b, type = void
      Variable Declaration: name = c, type = int
      Assign:
        Variable: name = d
          Const: 1
        Op: +
          Variable: name = b
          Variable: name = c
root@b1fbc5679b3c:/workspace/basic#
```