Parser

0. Environment

- WSL Ubuntu 20.04
- gcc (Ubuntu 9.4.0-1ubuntu1~20.04.1) 9.4.0
- flex 2.6.4
- bison (GNU Bison) 3.5.1
- GNU Make 4.2.1(Built for x86_64-pc-linux-gnu)

---

1. Modification

- **main.c**
  - Modify code to print *only* a *s*yntax tree.
  - Set *NO_ANALYZE*, *TraceParse* to TRUE.

```c
/****************************************************/
/* File: main.c                                     */
/* Main program for TINY compiler                   */
/* Compiler Construction: Principles and Practice   */
/* Kenneth C. Louden                                */
/****************************************************/

#include "globals.h"

/* set NO_PARSE to TRUE to get a scanner-only compiler */
#define NO_PARSE FLASE
/* set NO_ANALYZE to TRUE to get a parser-only compiler */
#define NO_ANALYZE TRUE

/* set NO_CODE to TRUE to get a compiler that does not
 * generate code
 */
#define NO_CODE FALSE

#include "util.h"
#if NO_PARSE
#include "scan.h"
#else
#include "parse.h"
#if !NO_ANALYZE
#include "analyze.h"
#if !NO_CODE
#include "cgen.h"
#endif
#endif
#endif

/* allocate global variables */
int lineno = 0;
FILE * source;
FILE * listing;
FILE * code;

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

int Error = FALSE;
```

```c
/* set NO_PARSE to TRUE to ge
#define NO_PARSE FLASE
/* set NO_ANALYZE to TRUE to
#define NO_ANALYZE TRUE
```

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

int Error = FALSE;
```

- **globals.h**
  - Overwrite your *globals.h* with *yacc/globals.h.*
  - “Syntax tree for parsing” should be updated to meet C-Minus Spec.
  - You can define your own AST.
    - You **can** modify/add/remove *NodeKind*, *StmtKind*, *ExpKind*, *ExpType*, and ***TreeNode*.**
      (You only should follow the output AST format specified in project goal slide.
      The Internal implementation is FREE.)
    - FAQ: What is the difference between *StatK* and *ExpK*?
      - It depends on your implementation. (= They are not important in C-Minus implementation)
        You can even remove *NodeKind* (the statement/expression classification) and integrate *StmtKind* and *ExpKind*.
  - *TreeNode** is used to define YYSTYPE in *cminus.y*

1. Modified treeNode structure by adding more NodeKind enum.
2. I declared 4 NodeKind enum, StmtK, ExpK, DclrK and ParamK.
   a. StmtK - StatementKind

      Here are 7 kind of statements to represent purpose of each statement.

      IfK, ElseK, WhileK, ReturnK, NonReturnK, CompK, AssignK.

   b. ExpK - ExpressionKind

      This is a NodeKind representing expressions.

OpK, ConstK, IdK, ArrEK, CallK.

c. DclrK - DeclarationKind

   DclrK is newly added Nodekind used to declaration of variable or function.

   VarK, ArrK, FuncK, TypeK.

d. ParamK - ParameterKind

   This is also added Nodekind for parameters parsed to function.

   NArrK, ArrPK, NullK.

- ***util.c***
  - *printTree*() function should be updated to print C-Minus Syntax Tree.
    - INDENT and UNINDENT macros with *printSpace*() shows tree structure by controlling indentation when printing nodes
    - *printTree*() traverses *child* and *sibling* fields in *TreeNode*
  - *newStmtNode*(), *newExprNode*() or other function should be updated to allocate and initialize new *Node*.
    - This functions are used in *cminus.y*. you can use a raw *malloc*() function instead of *newStmtNode*() and *newExprNode*() functions.

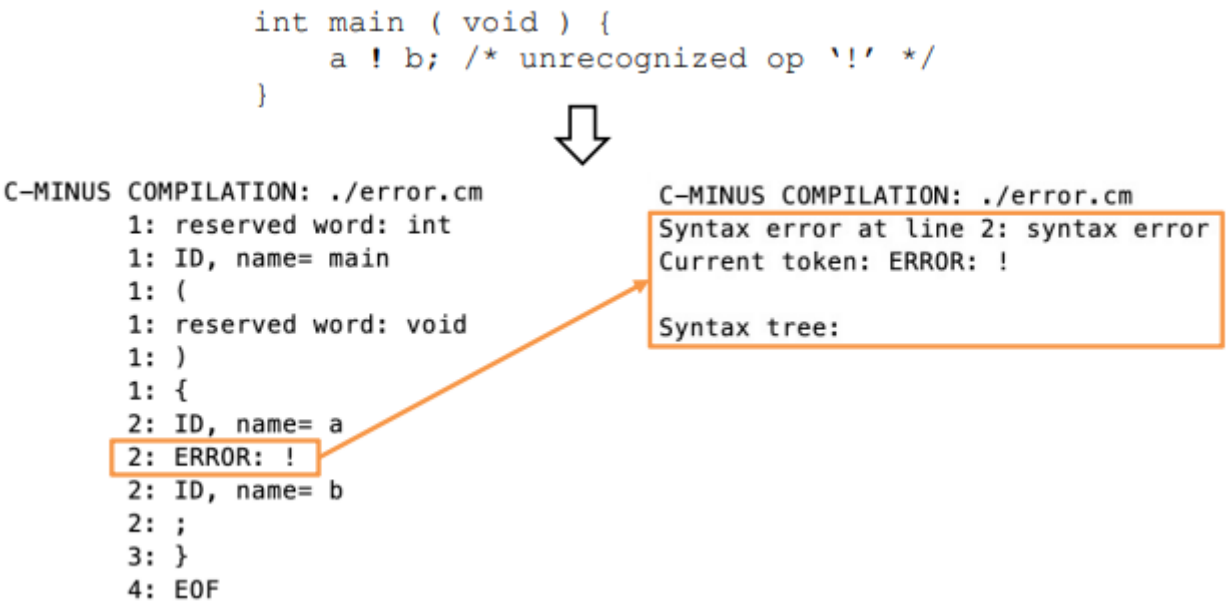
1. I added several functions that make node dynamically.
2. And PrintTree function is modified.

   As nodes nodekind, switch it cases, print strings satisfying output format.

- **cminus.y**
  - Copy *yacc/tiny.y* to *cminus.y*.
  - Write C-Minus tokens in the definition section.
    - Consider priority and associativity.
  - Define a C-Minus grammar and reduce actions for each rules.
    - *YYSTYPE* (the type of *$$*, *$1*, …) is defined as *TreeNode**.

1. In definition section, according to priority and associativity, I write tokens.
2. I modified TINY rules to C-Minus following BNF Grammar.
3. To avoid mid-action rules, I considered about to handle Terminals using Non-Terminals.

2. Error handling



If Error is detected, the “yyerror” function is called, and print ERROR, I put a line that `savedTree=NULL;`
This line makes a program not be able to print AST.

---

3. Consideration

1. Structure

To satisfy BNF Grammar, I had to modify TreeNode structure. By Following BNF 1 to 29, I divided statements into StmtK, ExpK and etc. In etc, I made new K nodes that could not be subset of exist K nodes. Those are DclrK and ParamK.

Using These 4 Kind of Nodes, I could divide every BNF statement 1 to 29.

2. Dangling Else

In BNF rule, there is conflict on 15. In this homework, we set a rule that Associate the else with **nearest if.**

I solve this problem by setting ELSE & RPAREN as “%nonassoc”.

This helps choose which if is associated with else problem because ELSE has no association. So If the ELSE is detected as lookahead token, then the program just do reduction, otherwise shift.

3. Terminals

I considered about how to handle information of Terminals. Because when the program try to reduce, the Terminals are already gone. So I had to save them. The way that I found is making more grammars for Terminals. Making new nodes for terminals, I could handle them as non-Terminal, and save information about that I want to use.

---

4. Test & Result

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

[Input file test.1.txt]

```
root@DESKTOP-UATIUE2:~/gitlab/2022_ele4029_2018008240/2_Parser# ./cminus_parser test.1.txt

C-MINUS COMPILATION: ./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: -
              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@DESKTOP-UATIUE2:~/gitlab/2022_ele4029_2018008240/2_Parser#
```

[Result of ./cminus_parser test.1.txt]

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

[Input file test.2.txt]

```
root@DESKTOP-UATIUE2:~/gitlab/2022_ele4029_2018008240/2_Parser# ./cminus_parser test.2.txt

C-MINUS COMPILATION: ./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
      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@DESKTOP-UATIUE2:~/gitlab/2022_ele4029_2018008240/2_Parser#
```

[Result of ./cminus_parser test.2.txt]