[Project 1 : Scanner]

컴퓨터전공 2013011253 김기원

Project 1 : C-Minus Scanner

0. Compilation tools & Environment

Basic Environment : VMware(Ubuntu 16.04.3 LTS)

Basic Tools : GCC (version 5.4.0) , Flex (version 2.6.0) , Makefile

1. Explanation (step by step)

1-1: Setting up Makefile :

For debugging and basic compilation, Makefile should be the first step.

In this project (scanner) , we just need 6 files for C-Minus scanner.

[globals.h main.c util.h util.c scan.h scan.c] can be more if include object files.

And 2 more files [lex.yy.c cminus.l] for C-Minus scanner using flex.

The method of modifying Makefile for each is include in the PPT, but I suggest to combine them like the code and operation showing below :

```
all: cminus cminus_flex

CC = gcc
FLEX = flex
CFLAGS =
OBJS = main.o util.o scan.o
OBJS_FLEX = main.o util.o lex.yy.o

cminus: $(OBJS)
    $(CC) $(CFLAGS) $(OBJS) -o cminus

cminus_flex: $(OBJS_FLEX)
    $(CC) $(CFLAGS) $(OBJS_FLEX) -o cminus_flex -lfl

lex.yy.c: cminus.l
    $(FLEX) cminus.l

lex.yy.o: lex.yy.c scan.h util.h globals.h
    $(CC) $(CFLAGS) -c lex.yy.c -lfl

main.o: main.c globals.h util.h scan.h
    $(CC) $(CFLAGS) -c main.c

util.o: util.c util.h globals.h
    $(CC) $(CFLAGS) -c util.c

scan.o: scan.c scan.h util.h globals.h
    $(CC) $(CFLAGS) -c scan.c
```

```
kkw@ubuntu:/mnt/hgfs/share/cp1$ sudo make clean
[sudo] password for kkw:
rm cminus
rm cminus_flex
rm lex.yy.c
rm *.o
kkw@ubuntu:/mnt/hgfs/share/cp1$ sudo make
gcc  -c main.c
main.c:48:1: warning: return type defaults to ‘int
 main( int argc, char * argv[] )
 ^
gcc  -c util.c
util.c:109:8: warning: type defaults to ‘int’ in
plicit-int]
 static indentno = 0;
        ^
gcc  -c scan.c
gcc  main.o util.o scan.o -o cminus
flex cminus.l
gcc  -c lex.yy.c -lfl
gcc  main.o util.o lex.yy.o -o cminus_flex -lfl
kkw@ubuntu:/mnt/hgfs/share/cp1$ ./cminus test.cm
```

```
clean:
    -rm cminus
    -rm cminus_flex
    -rm lex.yy.c
    -rm *.o
```

1-2: Modifying code :

In this project, we need to modify TINY compiler's code.

Main.c :

```
#define NO_PARSE TRUE
```

Turn this line to TRUE(1), you can see the next line #if and #else can skip the parser. It can set the compilation to scanner-only through this option.

```
int EchoSource = TRUE;
int TraceScan = TRUE;
```

EchoSource option is about the scanned source code line, and TraceScan is the option that specify every scanned token. For debugging, this option suggest to be TRUE(1). ***** This option showing at scan.c *****

Globals.h :

```
#define MAXRESERVED 6

typedef enum
    /* book-keeping tokens */
   {ENDFILE,ERROR,
    /* reserved words */
    IF,ELSE,WHILE,RETURN,INT,VOID, /* discared */
/*THEN,END,REPEAT,UNTIL,READ,WRITE,*/
    /* multicharacter tokens */
    ID,NUM,
    /* special symbols */

ASSIGN,EQ,NE,LT,LE,GT,GE,PLUS,MINUS,TIMES,OVER,LPAREN,RPAREN,LBRACE,RBRACE,LCU
RLY,RCURLY,SEMI,COMMA
   } TokenType;
```

Set MAXRESERVED to 6. For comparing token that is reserved word or not, project just need 6 reserved word for scanner of C-Minus language. (Textbook P491).

So the old reserved word for TINY can be deleted.

*** Deletion of old reserved word can cause undefined / undeclared in scan.c ***

Continue to add the symbols. scanner need it for modify which TINY lack of identifying token. (EQ, LT, LE, GT, GE .....)

Util.c :

Before modifying scan.c , it is more important to modify util.c. It can print out the token type which scanner identified. Set util.c for every output that can use for debugging and ensure of correctness.

```
void printToken( TokenType token, const char* tokenString )
{ switch (token)
  { case IF: fprintf(listing,"reserved word: %s\n",tokenString); break;
    case ELSE: fprintf(listing,"reserved word: %s\n",tokenString); break;
    case WHILE: fprintf(listing,"reserved word: %s\n",tokenString); break;
    case RETURN: fprintf(listing,"reserved word: %s\n",tokenString); break;
    case INT: fprintf(listing,"reserved word: %s\n",tokenString); break;
    case VOID: fprintf(listing,"reserved word: %s\n",tokenString); break;
    ////////////////////////////////////////
    case ASSIGN: fprintf(listing,"=\n"); break;
    case EQ: fprintf(listing,"==\n"); break;
    case NE: fprintf(listing,"!=\n"); break;
    case LE: fprintf(listing,"<=\n"); break;
    case GE: fprintf(listing,">=\n"); break;
    case LT: fprintf(listing,"<\n"); break;
    case GT: fprintf(listing,">\n"); break;
    ////////////////////////////////////////
    case LPAREN: fprintf(listing,"(\n"); break;
    case RPAREN: fprintf(listing,")\n"); break;
    case LBRACE: fprintf(listing,"[\n"); break;
    case RBRACE: fprintf(listing,"]\n"); break;
    case LCURLY: fprintf(listing,"{\n"); break;
    case RCURLY: fprintf(listing,"}\n"); break;
    ////////////////////////////////////////
    case SEMI: fprintf(listing,";\n"); break;
    case COMMA: fprintf(listing,",\n"); break;
    case PLUS: fprintf(listing,"+\n"); break;
    case MINUS: fprintf(listing,"-\n"); break;
    case TIMES: fprintf(listing,"*\n"); break;
    case OVER: fprintf(listing,"/\n"); break;
    case ENDFILE: fprintf(listing,"EOF\n"); break;
    case NUM: fprintf(listing,"NUM, val= %s\n",tokenString); break;
    case ID: fprintf(listing,"ID, name= %s\n",tokenString); break;
    case ERROR: fprintf(listing,"ERROR: %s\n",tokenString); break;
    default: /* should never happen */
      fprintf(listing,"Unknown token: %d\n",token);
  }
}
```

Add every token type for scanner. Especially the 6 reserved word for C-Minus, and symbols added in globals.h. Old reserved word for TINY can be delete.

1-3: Main of scanner : Scan.c modify :

In this step, too much code must be added for modify. The key problem is the symbols that contain two tokens like : /* , */ , <= , >= , == , !=

1. For the key problem, we need to add the option in to the code. The first step for adding option is define the type of state.

In this part , I separated the symbol which can be combine or not.

/* , */ , <= , >= , == , !=

For these symbols, we need to check the next token and identify it correctly. So it need to be adding option to define type of each state. Like the code below.

```
typedef enum

{ START,INEQ,INCOMMENT,INCOMMENTCONT,INNUM,INID,INLT,INGT,INNOT,INSLASH,DONE }
   StateType;
```

The function getNextChar and ungetNextChar decide to check next tokens or not.

In struct of reserved word, we need to change it to new 6 reserved word for comparing.

```
static struct
    { char* str;
      TokenType tok;
    } reservedWords[MAXRESERVED]
   =
{{"if",IF},{"else",ELSE},{"while",WHILE},{"return",RETURN},{"int",INT},{"void"
,VOID}};
```

This struct use for checking reserved word of current token. Function reservedLookup

```
static TokenType reservedLookup (char * s)
{ int i;
  for (i=0;i<MAXRESERVED;i++)
    if (!strcmp(s,reservedWords[i].str))
      return reservedWords[i].tok;
  return ID;
}
```

If it is reserved word, return it's token and showing the output. Else it will return ID.

Function TokenType getToken(void) is the main function. Through this function, every token for test sample can be identify. And this identification process through the STATE we defined at front (StateType).

In the function for identify token. Can be defined to IF or ELSE IF or ELSE

IF and ELSE IF define the combination symbols (/* */ .....)

ELSE just need to add more cases in the function for identify currentToken :

```
case '+':
  currentToken = PLUS;
  break;
case '-':
  currentToken = MINUS;
  break;
case '*':
  currentToken = TIMES;
  break;
case ';':
  currentToken = SEMI;
  break;
case ',':
  currentToken = COMMA;
  break;
case '(':
  currentToken = LPAREN;
  break;
case ')':
  currentToken = RPAREN;
  break;
case '[':
  currentToken = LBRACE;
  break;
case ']':
  currentToken = RBRACE;
  break;
case '{':
  currentToken = LCURLY;
  break;
case '}':
  currentToken = RCURLY;
  break;
```

For IF and ELSE IF. We need the new state for more process of identifying.

```
        if (isdigit(c))
          state = INNUM;
        else if (isalpha(c))
          state = INID;
        else if ((c == ' ') || (c == '\t') || (c == '\n'))
        {
          save = FALSE;
        }
        else if (c == '<')
          state = INLT;
        else if (c == '>')
          state = INGT;
        else if (c == '=')
          state = INEQ;
        else if (c == '!')
          state = INNOT;
        else if (c == '/')
        { save = FALSE;
          state = INSLASH;
        }
```

First we need to check it is the continued number or characters. Then check for SPACE, ,TAB, ENTER : we just need to ignore it.

For other state, we need new states to check the next token and decide currentToken.

The principle of each function is same, so I just showing one of it. '/* */'

State INSLASH :

```
case INSLASH:
        if (c == '*')
        { state = INCOMMENTCONT;
          save = FALSE;
        }
        else
        { state = DONE;
          currentToken = OVER;
          ungetNextChar();
        }
        break;
```

If the function get the next token, the state will be INSLASH, so the next token check in case INSLASH, for /* , change the state to INCOMMENTCONT, if there is nothing contain, go to else and identify it for OVER. And stop checking next token(for combination symbols).

```
case INCOMMENTCONT:
        save = FALSE;
        if (c == EOF)
        { state = DONE;
          currentToken = ENDFILE;
        }
        else if (c == '/') state = START;
        else state = INCOMMENT;
        break;
```

In to INCOMMENTCONT, the token go to INCOMMENT

```
case INCOMMENT:
        save = FALSE;
        if (c == EOF)
        { state = DONE;
          currentToken = ENDFILE;
        }
        else if (c == '*') state = INCOMMENTCONT;
        break;
```

It continue to check until the * show up, because our option was /* . And turn to state INCOMMENT check to state START, end the ignoring and start check the new token for identify.

```
else if (c == '<')
          state = INLT;
```

The same principle for less then eual function.

```
case INLT:
        state = DONE;
        if (c == '=')
          currentToken = LE;
        else
        {
          currentToken = LT;
          ungetNextChar();
        }
        break;
```

2. Modify using FLEX :

Modifying using flex is simple if the modifying scan.c is complete. Because getToken is automatically for using FLEX.

We just need to modifying **cminus.l**

The method of modifying cminus.l is same with modifying util.c, add identifier, symbols, reserved word 's return.

```
"if"            {return IF;}
"else"          {return ELSE;}
"while"         {return WHILE;}
"return"        {return RETURN;}
"int"           {return INT;}
"void"          {return VOID;}
"="             {return ASSIGN;}
"=="            {return EQ;}
"!="            {return NE;}
">"             {return GT;}
">="            {return GE;}
"<"             {return LT;}
"<="            {return LE;}
"+"             {return PLUS;}
"-"             {return MINUS;}
"*"             {return TIMES;}
"/"             {return OVER;}
"("             {return LPAREN;}
")"             {return RPAREN;}
"["             {return LBRACE;}
"]"             {return RBRACE;}
"{"             {return LCURLY;}
"}"             {return RCURLY;}
";"             {return SEMI;}
","             {return COMMA;}
```

Notice about the combination token can be recognized. That is nice.

We just need to set up /* ~~~~~ */ symbol option. To ignore until */ show up again.

```
"/*"            { char c;
                  while (1){
                   c = input();
                   if (c == EOF) break;
                   if (c == '*')
                   {
                     c = input();
                     if (c == '/' || c == EOF) break;
                   }
                   if (c == '\n') lineno++;
                  }
                }
.               {return ERROR;}
```

3. Result for running program :

cminus :

```
kkw@ubuntu:/mnt/hgfs/share/cp1$ ./cminus test.cm

CMINUS COMPILATION: test.cm
   1: /* A program to perform Euclid's
   2:    Algorithm to computer gcd */
   3:
   4: int gcd (int u, int v)
        4: reserved word: int
        4: ID, name= gcd
        4: (
        4: reserved word: int
        4: ID, name= u
        4: ,
        4: reserved word: int
        4: ID, name= v
        4: )
   5: {
        5: {
   6:     if (v == 0) return u;
        6: reserved word: if
        6: (
        6: ID, name= v
        6: ==
        6: NUM, val= 0
        6: )
        6: reserved word: return
        6: ID, name= u
        6: ;
   7:     else return gcd(v,u-u/v*v);
        7: reserved word: else
        7: reserved word: return
        7: ID, name= gcd
        7: (
        7: ID, name= v
        7: ,
        7: ID, name= u
        7: -
        7: ID, name= u
        7: /
        7: ID, name= v
        7: *
        7: ID, name= v
        7: )
        7: ;
   8:     /* u-u/v*v == u mod v */
   9: }
        9: }
  10:
  11: void main(void)
```

```
11: void main(void)
       11: reserved word: void
       11: ID, name= main
       11: (
       11: reserved word: void
       11: )
12: {
       12: {
13:     int x; int y;
       13: reserved word: int
       13: ID, name= x
       13: ;
       13: reserved word: int
       13: ID, name= y
       13: ;
14:     x = input(); y = input();
       14: ID, name= x
       14: =
       14: ID, name= input
       14: (
       14: )
       14: ;
       14: ID, name= y
       14: =
       14: ID, name= input
       14: (
       14: )
       14: ;
15:     output(gcd(x,y));
       15: ID, name= output
       15: (
       15: ID, name= gcd
       15: (
       15: ID, name= x
       15: ,
       15: ID, name= y
       15: )
       15: )
       15: ;
16: }
       16: }
       17: EOF
```

cminus_flex :

```
kkw@ubuntu:/mnt/hgfs/share/cp1$ ./cminus_flex test.cm

CMINUS COMPILATION: test.cm
        4: reserved word: int
        4: ID, name= gcd
        4: (
        4: reserved word: int
        4: ID, name= u
        4: ,
        4: reserved word: int
        4: ID, name= v
        4: )
        5: {
        6: reserved word: if
        6: (
        6: ID, name= v
        6: ==
        6: NUM, val= 0
        6: )
        6: reserved word: return
        6: ID, name= u
        6: ;
        7: reserved word: else
        7: reserved word: return
        7: ID, name= gcd
        7: (
        7: ID, name= v
        7: ,
        7: ID, name= u
        7: -
        7: ID, name= u
        7: /
        7: ID, name= v
        7: *
        7: ID, name= v
        7: )
        7: ;
        9: }
        11: reserved word: void
        11: ID, name= main
        11: (
        11: reserved word: void
        11: )
        12: {
        13: reserved word: int
        13: ID, name= x
        13: ;
        13: reserved word: int
        13: ID, name= y
        13: ID, name= y
        13: ;
        14: ID, name= x
        14: =
        14: ID, name= input
        14: (
        14: )
        14: ;
        14: ID, name= y
        14: =
        14: ID, name= input
        14: (
        14: )
        14: ;
        15: ID, name= output
        15: (
        15: ID, name= gcd
        15: (
        15: ID, name= x
        15: ,
        15: ID, name= y
        15: )
        15: )
        15: ;
        16: }
        17: EOF
```

It is the end of manual & report.

This report is 10 Pages because of several capture.

The Project code is zip with this report. Unzip and enter dir cp1.

Enter the command in terminal : make