C-Minus Compiler Implementation - 1_Scanner

2018008531 송연주

Implementation Method 1 : C code (`scan.c`)

Compilation method and environment

실행환경 : VMWare Workstation 16 Player, Ubuntu 18.04.5 LTS

컴파일 및 실행 방법

```
$ make
$ ./cminus_cimpl test.cm
```

Explanation about how to implement and how to operate

1. `globals.h`

   pdf에 제시된대로 `MAXRESERVED`를 6으로 수정하고 reserved words와 special symbols을 과제 조건에 맞게 수정하였다.

   ```
   /* MAXRESERVED = the number of reserved words */
   #define MAXRESERVED 6

   typedef enum
       /* book-keeping tokens */
      {ENDFILE,ERROR,
       /* reserved words */
       IF,ELSE,WHILE,RETURN,INT,VOID,
       /* multicharacter tokens */
       ID,NUM,
       /* special symbols */
       ASSIGN,EQ,NE,LT,LE,GT,GE,PLUS,MINUS,TIMES,OVER,LPAREN,RPAREN,
       LBRACE,RBRACE,LCURLY,RCURLY,SEMI,COMMA
      } TokenType;
   ```

2. `main.c`

   pdf에 제시된대로 `NO_PARSE`와 `TraceScan`을 TRUE로 수정하였다.

```
/* set NO_PARSE to TRUE to get a scanner-only compiler */
#define NO_PARSE TRUE
// ...
/* allocate and set tracing flags */
int EchoSource = FALSE;
int TraceScan = TRUE;
int TraceParse = FALSE;
int TraceAnalyze = FALSE;
int TraceCode = FALSE;
```

3. `scan.c`

   reserved word를 변경했으므로 lookup table을 수정하였다.

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

`getToken()`에 기존에 구현되어있지 않던 symbol들에 대한 부분을 추가하였다. `START` case에서 먼저 `getNextChar()`로 받아온 `c`가 2개의 토큰으로 이루어진 symbol(`<=`, `>=`, `==`, `!=`, `/**/`)인지 검사를 해야하기 때문에 `<`, `>`, `=`, `!`, `/`에 대해서는 각각에 해당하는 state로 보내주고, 이외의 경우에는 symbol이 한 개의 토큰으로 이루어져 있으므로 state를 `DONE`으로 바꾸고 각 경우에 맞는 `currentToken`값을 부여한다.

```
case START:
      // ...
      else if (c == '<')
        state = INLT;
      else if (c == '>')
        state = INGT;
      else if (c == '=')
        state = INEQ;
      else if (c == '!')
        state = INNE;
      else if (c == '/')
      {
        save = FALSE;
        state = INOVER;
      }
      else
      {
        state = DONE;
        switch (c)
        {
        // ...
        case '>':
```

```
          currentToken = GT;
          break;
        case '[':
          currentToken = LBRACE;
          break;
        case ']':
          currentToken = RBRACE;
          break;
        case '{':
          currentToken = LCURLY;
          break;
        case '}':
          currentToken = RCURLY;
          break;
        case ',':
          currentToken = COMMA;
          break;
        default:
          currentToken = ERROR;
          break;
        }
      }
      break;
```

state `INLT`, `INGT`, `INEQ`, `INNE`는 비슷한 방법으로 동작한다. 각 `c`에 대응되는 `<`, `>`, `=`, `!` 이 토큰 2개로 이루어진 symbol인지 확인하기 위한 state이다.

각 state에서는 바로 다음에 나오는 토큰이 `=`인지를 검사해서 `=`가 아닐 경우에는 토큰 하나로 이루어진 symbol이므로 `ungetNextChar()`를 통해 토큰을 반환하고 `currentToken`값을 각 경우에 맞게 `LT`, `GT`, `ASSIGN`, `ERROR`(이 경우에는 save를 FALSE로 설정)로 설정한다. 반대로 다음에 나오는 토큰이 '='일 경우에는 각 경우에 맞게 `currentToken`을 `LE`, `GE`, `EQ`, `NE`로 설정한다. 이 과정이 끝나면 state를 `DONE`으로 바꿔 다음 단계로 넘어갈 수 있도록 해준다.

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

```
        currentToken = ASSIGN;
      }
      else
        currentToken = EQ;
      state = DONE;
      break;
    case INNE:
      if (c != '=')
      {
        ungetNextChar();
        save = FALSE;
        currentToken = ERROR;
      }
      else
        currentToken = NE;
      state = DONE;
      break;
```

토큰으로 `/`이 들어왔을 때는 state를 `INOVER`로 바꿔 이것이 나누기(`/`)를 의미하는지 주석(`/* */`)을 의미하는지 확인을 해주어야 한다.

만약 그 다음으로 나오는 토큰이 `*`이라면 이는 주석을 의미하므로 state를 `INCOMMENT`로 바꾸고 save를 FALSE로 설정해준다. 하지만 그것이 아니라면 이는 주석이 아니라 나누기(`/`)를 의미하므로 `ungetNextChar()`로 토큰을 반환해준 뒤 `currentToken`을 `OVER`로 설정하여 나누기임을 명시해준다.

`INCOMMENT` state는 주석과 관련한 내용인데 만약 다음에 들어오는 토큰이 `*`이라면 주석의 끝인 `*/`을 의미하는 것일 수도 있으므로 state를 `INCOMMENT_`로 바꿔 검사해준다. 위의 경우가 아니라면 주석 내부에 있으므로 현 상태를 유지한다.

`INCOMMENT_` 에서는 주석이 끝났는지를 검사한다. 이전에 `*`이 나왔으므로 `/`가 나오면 `*/`이 되어 주석이 끝나므로 state를 `START`로 바꾼다. 만약 `/`가 나오지 않는다면 주석이 계속되므로 state를 `INCOMMENT`로 바꿔준다.

```
    case INOVER: // /* */ 에 대한 주석 처리와 관련된 부분
      if (c == '*') // 주석
      {
        state = INCOMMENT;
        save = FALSE;
      }
      else // 주석이 아님
      {
        state = DONE;
        ungetNextChar();
        currentToken = OVER;
      }
    case INCOMMENT:
      save = FALSE;
      if (c == EOF)
      {
        state = DONE;
        currentToken = ENDFILE;
      }
      else if (c == '*') // 주석
        state = INCOMMENT_;
      break;
    case INCOMMENT_:
```

```
      save = FALSE;
      if (c == EOF)
      {
        state = DONE;
        currentToken = ENDFILE;
      }
      else if (c == '/') // 주석의 마지막
        state = START;
      else // 주석이 끝나지 않고 계속됨
        state = INCOMMENT;
      break;
```

4. `util.c`

   수정된 reserved word와 symbol에 대해서 `printToken()`를 수정해준다.

```
void printToken( TokenType token, const char* tokenString )
{ switch (token)
  { case IF:
    case ELSE:
    case WHILE:
    case RETURN:
    case INT:
    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 LT: fprintf(listing,"<\n"); break;
    case LE: fprintf(listing,"<=\n"); break;
    case GT: fprintf(listing,">\n"); break;
    case GE: 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 COMMA: fprintf(listing,",\n"); break;
    // ...
  }
}
```

Example and Result Screenshot

- example 1. `test.cm`

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

`test.cm`에 대한 output

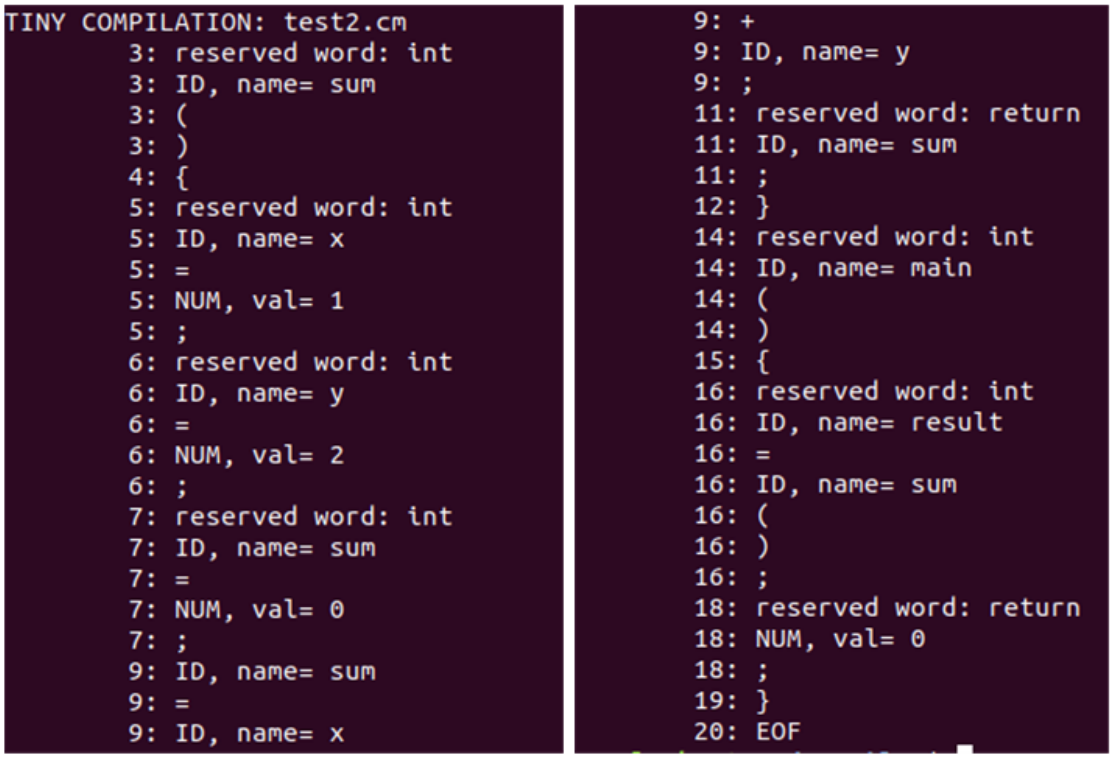
```
TINY COMPILATION: test.cm
        4: reserved word: int
        4: ID, name= gcd
        4: (
        4: reserved word: int
        4: ID, name= u
        4: ,
        4: reserved word: int
        4: ID, name= v
        4: )
        5: {
        6: reserved word: if
        6: (
        6: ID, name= 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: ;
       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
```

- example 2. `test2.cm`

```
/* for testing cminus_lex */

int sum()
{
    int x = 1;
    int y = 2;
    int sum = 0;

    sum = x + y;

    return sum;
```

```
}

int main()
{
    int result = sum();

    return 0;
}
```

`test2.cm`에 대한 output



Implementation Method 2 : Lex(flex) (`cminus.l`)

Compilation method and environment

컴파일 및 실행 방법

```
$ make
$ ./cminus_lex test.cm
```

Explanation about how to implement and how to operate

- `cminus.l`

  `tiny.l`의 기존 내용 중 일부를 수정하여 `cminus.l`을 생성하였다.

  reserved word에 해당하는 if, else, while, return, int, void에 대한 내용을 추가하고 기존 `tiny.l`에 없던 symbol에 대한 내용도 추가한다.

```
"if"            {return IF;}
"else"          {return ELSE;}
"while"         {return WHILE;}
"return"        {return RETURN;}
"int"           {return INT;}
"void"          {return VOID;}
"="             {return ASSIGN;}
"=="            {return EQ;}
"<="            {return LE;}
">"             {return GT;}
">="            {return GE;}
"["             {return LBRACE;}
"]"             {return RBRACE;}
"{"             {return LCURLY;}
"}"             {return RCURLY;}
","             {return COMMA;}
// ...
```

주석의 처리와 관련된 부분이다. 주석이 끝났는지 확인하기 위해서는 토큰 2개(`*/`)를 확인해야하므로 `*`과 `/`가 순서대로 나왔는지 확인하기 위해 `check` 변수를 사용한다.

`check`는 `*`이 나왔을 때 1로 바뀌고, 그 다음 loop에서 `if (c == '/' && check == 1) break;`을 통해 `*/`의 경우인지 확인한다. 만약 이 if문에 걸리지 않는다면 주석이 끝나지 않았음을 의미하고 경우에 따라 `check`의 값을 0 또는 1로 바꿔준다. `*/`이 나왔을 때 do-while문이 break된다.

```
"/*"            { char c;
                  int check = 0;
                  do
                  { c = input();
                    if (c == EOF) break;
                    if (c == '\n') lineno++;
                    if (c == '/' && check == 1) break;
                    if (c == '*') check = 1;
                    else check = 0;
                  } while (1);
                }
.               {return ERROR;}
```

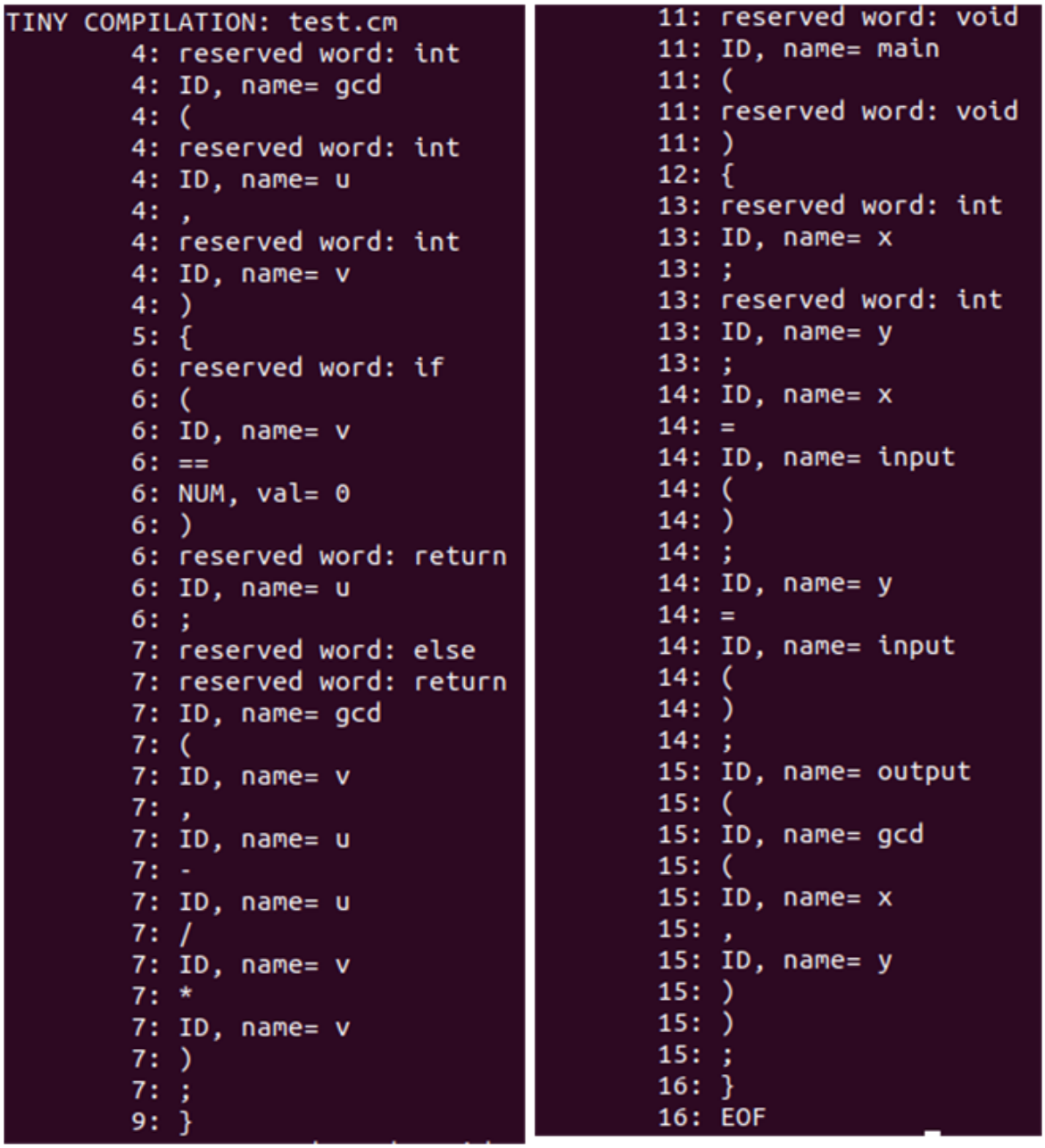
Example and Result Screenshot

- example 1. `test.cm`

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

`test.cm`에 대한 output



- example 2. `test2.cm`

```
/* for testing cminus_lex */

int sum()
{
    int x = 1;
    int y = 2;
    int sum = 0;

    sum = x + y;

    return sum;
```

```
}

int main()
{
    int result = sum();

    return 0;
}
```

`test2.cm`에 대한 output

```
TINY COMPILATION: test2.cm
        3: reserved word: int
        3: ID, name= sum
        3: (
        3: )
        4: {
        5: reserved word: int
        5: ID, name= x
        5: =
        5: NUM, val= 1
        5: ;
        6: reserved word: int
        6: ID, name= y
        6: =
        6: NUM, val= 2
        6: ;
        7: reserved word: int
        7: ID, name= sum
        7: =
        7: NUM, val= 0
        7: ;
        9: ID, name= sum
        9: =
        9: ID, name= x
        9: +
        9: ID, name= y
        9: ;
        11: reserved word: return
        11: ID, name= sum
        11: ;
        12: }
        14: reserved word: int
        14: ID, name= main
        14: (
        14: )
        15: {
        16: reserved word: int
        16: ID, name= result
        16: =
        16: ID, name= sum
        16: (
        16: )
        16: ;
        18: reserved word: return
        18: NUM, val= 0
        18: ;
        19: }
        19: EOF
```