A1-Implementation of Lexical Analyzer using C

Sneha Sriram Kannan 185001157

03-02-2021

1 Problem Statement

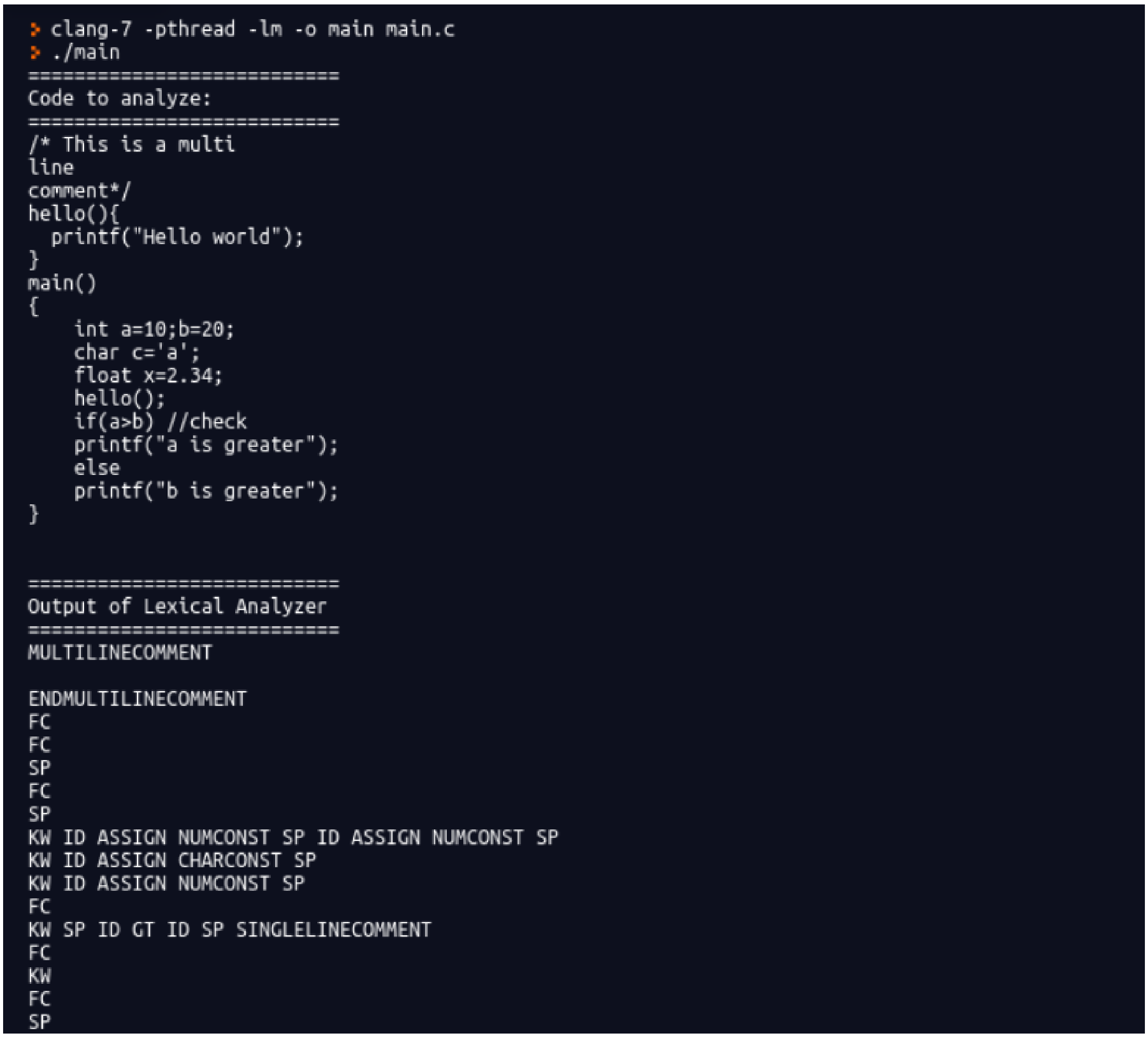
Develop a Lexical analyzer to recognize the patterns namely, identifiers, constants, comments and operators using the following regular expressions.

2 Code

```c
#include <stdio.h>

#include <string.h>

#include <ctype.h>

int substr(char str1[], char str2[]);
int checkFunction(char str[]);
int checkKeyword(char str[]);
int doubleop(char a, char b);
int doublelogicalop(char a, char b);
int LexicalAnalyzer();

int main() {
   LexicalAnalyzer();
   return 0;
}

int LexicalAnalyzer() {
   char file[10][128];

   FILE * fd = fopen("code.txt", "r");
   int i = 0, doubleopIndex;
   //reading code from a file and storing in an array
   while (fgets(file[i], sizeof(file[i]), fd))
      i++;
   int multi = 0;

   printf("==========================\n");
   printf("Code to analyze:\n");
   printf("==========================\n");
   for (int j = 0; j < i; j++) {
      printf("%s", file[j]);
   }
   printf("\n\n\n");
   printf("==========================\n");
   printf("Output of Lexical Analyzer\n");
   printf("==========================\n");
   //traversing through rows of the file
   for (int j = 0; j < i; j++) {
      if (checkFunction(file[j]) == 1) //if it is a function, FC is the only output
      {
         printf("FC\n");
         continue;
      }
      for (int k = 0; k < strlen(file[j]); k++) //traversing through characters in a row of the file
      {
         //checking for end of a multiline comment
         if (multi == 1) {
            if ((strlen(file[j]) - 1) && file[j][k] == '*' && file[j][k + 1] == '/') {
               printf("ENDMULTILINECOMMENT ");
               multi = 0;
               break;
            }
         }
         //not part of multi line comment
         else {
            //checking for start of comments
            if (k != (strlen(file[j]) - 1) && file[j][k] == '/' && file[j][k + 1] == '/') {
               printf("SINGLELINECOMMENT ");
               break;
            } else if ((strlen(file[j]) - 1) && file[j][k] == '/' && file[j][k + 1] == '*') {
               printf("MULTILINECOMMENT ");
               multi = 1;
               break;
            }

            //checking for operators
            else if (file[j][k] == '+' || file[j][k] == '-' || file[j][k] == '*' || file[j][k] == '/' || file[j][k] == '%')
               printf("ARITHOP ");
            else if (file[j][k] == '!')
               printf("LOGICALOP ");
            else if (k != (strlen(file[j]) - 1) && doublelogicalop(file[j][k], file[j][k + 1]) == 1)
               printf("LOGICALOP ");

            //checking for seperators
            else if (file[j][k] == '{' || file[j][k] == '}' || file[j][k] == ';' || file[j][k] == ',' || file[j][k] == ')' || file[j][k] == '(')
               printf("SP ");

            //checking for operators
            else if (k != (strlen(file[j]) - 1) && doubleop(file[j][k], file[j][k + 1]) == 1) {
               //checking if it is an operator with 2 symbols (<>,<= etc)
               if (!(isalpha(file[j][k + 1]) || isdigit(file[j][k + 1])))
                  k++;
               continue;
            } else if (file[j][k] == '=')
               printf("ASSIGN ");

            //searching for numbers
            else if (isdigit(file[j][k]) && k != (strlen(file[j]) - 1)) {
               while ((k + 1) != (strlen(file[j]) - 1)) {
                  if (isdigit(file[j][k + 1]))
                     k++;
                  else if (file[j][k + 1] == '.')
                     k++;
                  else
                     break;
               }
               printf("NUMCONST ");
            }
            //space
            else if (file[j][k] == ' ')
               continue;
            else if (file[j][k] == '\'') {
               while (k != (strlen(file[j]) - 1)) {
                  k++;
                  if (file[j][k] == '\'') {
                     printf("CHARCONST ");
                     break;
                  }
               }
            } else if (file[j][k] == '\"') {
               while (k != (strlen(file[j]) - 1)) {
                  k++;
                  if (file[j][k] == '\"') {
                     printf("STRCONST ");
                     break;
                  }
               }
            }
            //extracting a string checking for keywords and id
            else if (isalpha(file[j][k])) {
               char substring[200];
               int subIndex = 0;
               substring[subIndex++] = file[j][k];
               while ((k + 1) != (strlen(file[j]) - 1)) {
                  if (isalpha(file[j][k + 1])) {
                     substring[subIndex++] = file[j][k + 1];
                     k++;
                  } else
                     break;
               }
               substring[subIndex++] = '\n';
               substring[subIndex] = '\0';
               if (checkKeyword(substring) == 1)
                  printf("KW ");
               else
                  printf("ID ");
            }
         }
      }
      printf("\n");
   }
   return 0;
}

int doublelogicalop(char a, char b) //checks if ab is a logical operator
{
   if ((a == b && b == '&') || (a == b && b == '|'))
      return 1;
   return 0;
}

int doubleop(char a, char b) //checks if ab is a relational operator
{
   if (a == '<') {
      if (b == '>')
         printf("NE ");
      else if (b == '=')
         printf("LE ");
      else
         printf("LT ");
      return 1;
   }
   if (a == '>') {
      if (b == '=')
         printf("GE ");
      else
         printf("GT ");
      return 1;
   }
   if (a == '=' && b == '=') {
      printf("EQ ");
      return 1;
   }
   return 0;
}
```

```
int checkFunction(char str[]) //checks if str is a function
{
   int i = 0;
   int open = 0, close = 0;
   char funcname[200];
   int subIndex = 0;
   while (!isalpha(str[i]))
      i++;
   while (i < strlen(str)) {
      if (str[i] == ' ')
         i++;
      if (isalpha(str[i]))
         funcname[subIndex++] = str[i++];
      else if (str[i] == '(') {
         open = 1;
         i++;
         break;
      } else
         return 0;
   }
   funcname[subIndex++] = '\n';
   funcname[subIndex] = '\0';
   if (checkKeyword(funcname))
      return 0;
   while (i < strlen(str) && open == 1) {
      if (str[i++] == ')')
         return 1;
   }
   return 0;
}

int checkKeyword(char str[]) //checks if str is a keyword
{
   FILE * fd = fopen("keywords.txt", "r");
   char filestr[20];
   while (fgets(filestr, 60, fd) != NULL) {
      if (strcmp(str, filestr) == 0) {
         return 1;
      }
   }
   return 0;
}

int substr(char str1[], char str2[]) //checks if str1 is a substring of str2
{
   int i, j = 0;
   while (i < strlen(str1) && j < strlen(str2)) {
      if (str1[i] == str2[j]) {
         i++;
         j++;
      } else {
         j++;
      }
   }
   if (i == strlen(str1))
      return 1;
   else
      return 0;
}
```

3 Output Screenshot



```
> clang-7 -pthread -lm -o main main.c
> ./main
===========================
Code to analyze:
===========================
/* This is a multi
line
comment*/
hello(){
  printf("Hello world");
}
main()
{
    int a=10;b=20;
    char c='a';
    float x=2.34;
    hello();
    if(a>b) //check
    printf("a is greater");
    else
    printf("b is greater");
}


===========================
Output of Lexical Analyzer
===========================
MULTILINECOMMENT

ENDMULTILINECOMMENT
FC
FC
SP
FC
SP
KW ID ASSIGN NUMCONST SP ID ASSIGN NUMCONST SP
KW ID ASSIGN CHARCONST SP
KW ID ASSIGN NUMCONST SP
FC
KW SP ID GT ID SP SINGLELINECOMMENT
FC
KW
FC
SP
```

Figure 1: Lexical Analyzer Output

4 Learning Outcome

1. I learnt how to identify user defined functions in the code.
2. I understood how to distinguish between functions and parantheses that follow a keyword.
3. I was able to parse the input to get the lexemes and identify the pattern that they matched.
4. The order in which lexemes are identified is important
5. I have realized the role of a lexical analyzer in a compiler.