Lab No 3: Date: 24/01/2026

Construction of token generator

Name: Satyam Singh

Reg no: 230905256

CSE B2 - 37

Lab Exercises:

Q1) Write functions to identify the following tokens.

a. Arithmetic, relational and logical operators.

b. Special symbols, keywords, numerical constants, string literals and identifiers.

Ans)

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>

FILE *fp;
int row = 1, col = 0;
int isKeyword(char *str) {
   char *keywords[] = {
      "int","float","double","char","if","else",
      "while","for","return","void","main"
   };
   int n = 11;
   for (int i = 0; i < n; i++) {
      if (strcmp(str, keywords[i]) == 0)
         return 1;
   }
   return 0;
}
```

```
void getNextToken() {
  char ch, buffer[50];
  int i;

  while ((ch = fgetc(fp)) != EOF) {
    col++;
    if (ch == '\n') {
      row++;
      col = 0;
      continue;
    }
    if (isspace(ch))
      continue;
    if (isalpha(ch) || ch == '_') {
      i = 0;
      buffer[i++] = ch;
      int startCol = col;

      while (isalnum(ch = fgetc(fp)) || ch == '_') {
        buffer[i++] = ch;
        col++;
      }
      buffer[i] = '\0';
      ungetc(ch, fp);

      if (isKeyword(buffer))
        printf("<%s,%d,%d>\n", buffer, row, startCol);
      else
        printf("<id,%d,%d>\n", row, startCol);
    }
    else if (isdigit(ch)) {
      i = 0;
      buffer[i++] = ch;
      int startCol = col;

      while (isdigit(ch = fgetc(fp))) {
        buffer[i++] = ch;
        col++;
      }
      buffer[i] = '\0';
      ungetc(ch, fp);

      printf("<num,%d,%d>\n", row, startCol);
```

```
    }
    else if (ch == '=' || ch == '<' || ch == '>' || ch == '+' || ch == '-') {
      int startCol = col;
      char next = fgetc(fp);

      if ((ch == '=' && next == '=') ||
        (ch == '<' && next == '=') ||
        (ch == '>' && next == '=')) {
        col++;
        printf("<%c%c,%d,%d>\n", ch, next, row, startCol);
      } else {
        ungetc(next, fp);
        printf("<%c,%d,%d>\n", ch, row, startCol);
      }
    }
    else if (ch == ';' || ch == ',' || ch == '(' || ch == ')' ||
         ch == '{' || ch == '}') {
      printf("<%c,%d,%d>\n", ch, row, col);
    }
  }
}
int main() {
  fp = fopen("input.c", "r");
  if (fp == NULL) {
    printf("Cannot open file\n");
    return 1;
  }

  getNextToken();
  fclose(fp);
  return 0;
}
```

Output:

week3 > C input.c > ...

```c
main()
{
int a,b,sum;
a = 1;
b = 1;
sum = a + b;
}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
cdl-6cse-b2@sce-cl11-03:~/Documents/230905256/week3$ gcc -o q1 q1.c
cdl-6cse-b2@sce-cl11-03:~/Documents/230905256/week3$ ./q1
<main,1,1>
<(,1,5>
<),1,6>
<{,2,1>
<int,3,1>
<id,3,5>
<,,3,6>
<id,3,7>
<,,3,8>
<id,3,9>
<;,3,12>
<id,4,1>
<=,4,3>
<num,4,5>
<;,4,6>
<id,5,1>
<=,5,3>
<num,5,5>
<;,5,6>
<id,6,1>
<=,6,5>
<id,6,7>
<+,6,9>
<id,6,11>
<;,6,12>
<},7,1>
```

Q2) Design a lexical analyzer that includes a getNextToken() function for processing a simple C program.The analyzer should construct a token structure containing the row number, column number, and tokentype for each identified token. The getNextToken() function must ignore tokens located within single-line or multi-line comments, as well as those found inside string literals. Additionally, it should strip out preprocessor directives.

Ans)

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>
#define MAX 100
typedef struct {
   int row;
   int col;
   char type[30];
   char lexeme[50];
} Token;
FILE *fp;
int row = 1, col = 0;
char *keywords[] = {
   "int", "float", "char", "double", "if", "else",
   "while", "for", "return", "void", "break", "continue"
};
int keywordCount = sizeof(keywords) / sizeof(keywords[0]);
int isKeyword(char *);
Token getNextToken();
int isKeyword(char *str) {
   for (int i = 0; i < keywordCount; i++) {
      if (strcmp(str, keywords[i]) == 0)
         return 1;
   }
   return 0;
}
Token getNextToken() {
   Token token;
   char ch;
   int i;
```

```
strcpy(token.type, "EOF");
token.lexeme[0] = '\0';
token.row = row;
token.col = col;
while ((ch = fgetc(fp)) != EOF) {
    col++;
    if (ch == '\n') {
        row++;
        col = 0;
        continue;
    }
    if (isspace(ch))
        continue;
    if (ch == '#') {
        while ((ch = fgetc(fp)) != '\n' && ch != EOF);
        row++;
        col = 0;
        continue;
    }
    if (ch == '/') {
        char next = fgetc(fp);
        col++;
        if (next == '/') {
            while ((ch = fgetc(fp)) != '\n' && ch != EOF);
            row++;
            col = 0;
            continue;
        }
        else if (next == '*') {
            while ((ch = fgetc(fp)) != EOF) {
                if (ch == '\n') {
                    row++;
                    col = 0;
                }
                if (ch == '*') {
                    char temp = fgetc(fp);
                    if (temp == '/')
                        break;
                    else
                        ungetc(temp, fp);
                }
            }
            continue;
        }
```

```
        else {
            ungetc(next, fp);
            col--;
        }
    }
    if (ch == '"') {
        while ((ch = fgetc(fp)) != '"' && ch != EOF) {
            if (ch == '\n') {
                row++;
                col = 0;
            }
        }
        continue;
    }
    if (isalpha(ch) || ch == '_') {
        i = 0;
        token.lexeme[i++] = ch;
        token.row = row;
        token.col = col;

        while ((ch = fgetc(fp)) != EOF && (isalnum(ch) || ch == '_')) {
            token.lexeme[i++] = ch;
            col++;
        }
        token.lexeme[i] = '\0';
        if (ch != EOF)
            ungetc(ch, fp);

        if (isKeyword(token.lexeme))
            strcpy(token.type, "KEYWORD");
        else
            strcpy(token.type, "IDENTIFIER");

        return token;
    }
    if (isdigit(ch)) {
        i = 0;
        token.lexeme[i++] = ch;
        token.row = row;
        token.col = col;
        while ((ch = fgetc(fp)) != EOF && isdigit(ch)) {
            token.lexeme[i++] = ch;
            col++;
```

```
            }
            token.lexeme[i] = '\0';
            if (ch != EOF)
                ungetc(ch, fp);
            strcpy(token.type, "NUMBER");
            return token;
        }
        if (strchr("+-*/%=<>!", ch)) {
            token.lexeme[0] = ch;
            token.lexeme[1] = '\0';
            token.row = row;
            token.col = col;
            strcpy(token.type, "OPERATOR");
            return token;
        }
        if (strchr(";(),{}[]", ch)) {
            token.lexeme[0] = ch;
            token.lexeme[1] = '\0';
            token.row = row;
            token.col = col;
            strcpy(token.type, "SPECIAL_SYMBOL");
            return token;
        }
    }
    return token;
}
int main() {
    Token t;
    fp = fopen("q1.c", "r");
    if (!fp) {
        printf("Error opening file\n");
        return 1;
    }
    printf("Row\tCol\tType\t\tLexeme\n");
    printf("---------------------------------------\n");
    while (1) {
        t = getNextToken();
        if (strcmp(t.type, "EOF") == 0)
            break;

        printf("%d\t%d\t%-15s %s\n", t.row, t.col, t.type, t.lexeme);
    }
    fclose(fp);
    return 0;
```

```
}
```

Output:

```
cdl-6cse-b2@sce-cl11-03:~/Documents/230905256/week3$ gcc -o q2 q2.c
cdl-6cse-b2@sce-cl11-03:~/Documents/230905256/week3$ ./q2
Row     Col     Type            Lexeme
------------------------------------
5       1       IDENTIFIER      FILE
5       6       OPERATOR        *
5       7       IDENTIFIER      fp
5       9       SPECIAL_SYMBOL  ;
6       1       KEYWORD         int
6       5       IDENTIFIER      row
6       9       OPERATOR        =
6       11      NUMBER          1
6       12      SPECIAL_SYMBOL  ,
6       14      IDENTIFIER      col
6       18      OPERATOR        =
6       20      NUMBER          0
6       21      SPECIAL_SYMBOL  ;
7       1       KEYWORD         int
7       5       IDENTIFIER      isKeyword
7       14      SPECIAL_SYMBOL  (
7       15      KEYWORD         char
7       20      OPERATOR        *
7       21      IDENTIFIER      str
7       24      SPECIAL_SYMBOL  )
7       26      SPECIAL_SYMBOL  {
8       5       KEYWORD         char
8       10      OPERATOR        *
8       11      IDENTIFIER      keywords
8       19      SPECIAL_SYMBOL  [
8       20      SPECIAL_SYMBOL  ]
8       22      OPERATOR        =
8       24      SPECIAL_SYMBOL  {
9       10      SPECIAL_SYMBOL  ,
9       12      SPECIAL_SYMBOL  ,
9       14      SPECIAL_SYMBOL  ,
9       16      SPECIAL_SYMBOL  ,
9       18      SPECIAL_SYMBOL  ,
9       20      SPECIAL_SYMBOL  ,
```