SAMPLE

// Solved problem

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int main()

{

char ca, cb;

char buff[100];

int k = 0;

FILE \*fa = fopen("sampleInputFile.c", "r");

if (fa == NULL)

{

printf("Cannot open the file\n");

exit(0);

}

ca = getc(fa);

while (ca != EOF)

{

k = 0;

if (ca == '=')

{

buff[k++] = ca;

cb = getc(fa);

if (cb == '=')

{

buff[k++] = cb;

buff[k++] = '\0';

printf("Relational operator %s\n", buff);

}

else

{

buff[k++] = '\0';

printf("Assignment operator %s \n", buff);

}

}

else

{

if (ca == '<' || ca == '>' || ca == '!')

{

buff[k++] = ca;

cb = getc(fa);

if (cb == '=')

{

buff[k++] = cb;

}

buff[k++] = '\0';

printf("Relational operator %s\n", buff);

}

else

{

buff[k++] = '\0';

}

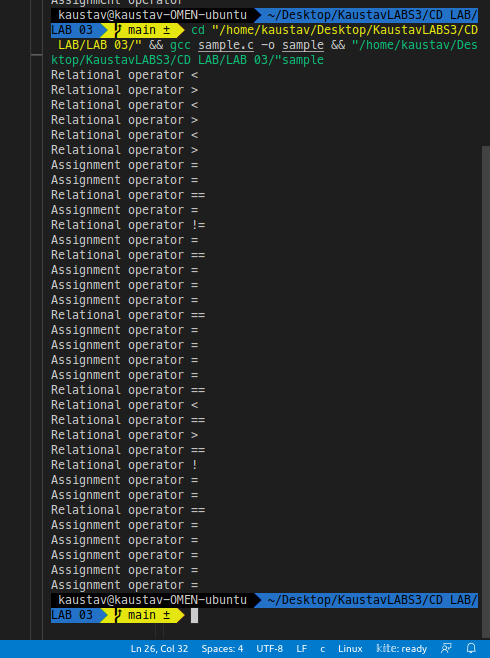
}

ca = getc(fa);

}

return 0;

}



Q1

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

#define FILEINPUT "sample.c"

struct token

{

char lexeme[64];

int row, col;

char type[20];

};

static int row = 1, col = 1;

char buf[2048];

const char specialsymbols[] = {'?', ';', ':', ','};

const char \*keywords[] = {"const", "char", "int", "return", "for", "while", "do",

"switch", "if", "else", "unsigned", "case",

"break"};

const char arithmeticsymbols[] = {'\*'};

int isKeyword(const char \*str)

{

for (int i = 0; i < sizeof(keywords) / sizeof(char \*); i++)

{

if (strcmp(str, keywords[i]) == 0)

{

return 1;

}

}

return 0;

}

int charBelongsTo(int c, const char \*arr)

{

int len;

if (arr == specialsymbols)

{

len = sizeof(specialsymbols) / sizeof(char);

}

else if (arr == arithmeticsymbols)

{

len = sizeof(arithmeticsymbols) / sizeof(char);

}

for (int i = 0; i < len; i++)

{

if (c == arr[i])

{

return 1;

}

}

return 0;

}

void fillToken(struct token \*tkn, char c, int row, int col, char \*type)

{

tkn->row = row;

tkn->col = col;

strcpy(tkn->type, type);

tkn->lexeme[0] = c;

tkn->lexeme[1] = '\0';

}

void newLine()

{

++row;

col = 1;

}

struct token getNextToken(FILE \*f1)

{

int c;

struct token tkn =

{

.row = -1};

int gotToken = 0;

while (!gotToken && (c = fgetc(f1)) != EOF)

{

if (charBelongsTo(c, specialsymbols))

{

fillToken(&tkn, c, row, col, "SS");

gotToken = 1;

++col;

}

else if (charBelongsTo(c, arithmeticsymbols))

{

fillToken(&tkn, c, row, col, "ARITHMETIC OPERATOR");

gotToken = 1;

++col;

}

else if (c == '(')

{

fillToken(&tkn, c, row, col, "LB");

gotToken = 1;

++col;

}

else if (c == ')')

{

fillToken(&tkn, c, row, col, "RB");

gotToken = 1;

++col;

}

else if (c == '{')

{

fillToken(&tkn, c, row, col, "LC");

gotToken = 1;

++col;

}

else if (c == '[')

{

fillToken(&tkn, c, row, col, "LSB");

gotToken = 1;

++col;

}

else if (c == ']')

{

fillToken(&tkn, c, row, col, "RSB");

gotToken = 1;

++col;

}

else if (c == '}')

{

fillToken(&tkn, c, row, col, "RC");

gotToken = 1;

++col;

}

else if (c == '+')

{

int d = fgetc(f1);

if (d != '+')

{

fillToken(&tkn, c, row, col, "ARITHMETICOPERATOR");

gotToken = 1;

++col;

fseek(f1, -1, SEEK\_CUR);

}

else

{

fillToken(&tkn, c, row, col, "UNARYOPERATOR");

strcpy(tkn.lexeme, "++");

gotToken = 1;

col += 2;

}

}

else if (c == '-')

{

int d = fgetc(f1);

if (d != '-')

{

fillToken(&tkn, c, row, col, "ARITHMETICOPERATOR");

gotToken = 1;

++col;

fseek(f1, -1, SEEK\_CUR);

}

else

{

fillToken(&tkn, c, row, col, "UNARYOPERATOR");

strcpy(tkn.lexeme, "--");

gotToken = 1;

col += 2;

}

}

else if (c == '=')

{

int d = fgetc(f1);

if (d != '=')

{

fillToken(&tkn, c, row, col, "ASSIGNMENTOPERATOR");

gotToken = 1;

++col;

fseek(f1, -1, SEEK\_CUR);

}

else

{

fillToken(&tkn, c, row, col, "RELATIONALOPERATOR");

strcpy(tkn.lexeme, "==");

gotToken = 1;

col += 2;

}

}

else if (isdigit(c))

{

tkn.row = row;

tkn.col = col++;

tkn.lexeme[0] = c;

int k = 1;

while ((c = fgetc(f1)) != EOF && isdigit(c))

{

tkn.lexeme[k++] = c;

col++;

}

tkn.lexeme[k] = '\0';

strcpy(tkn.type, "NUMBER");

gotToken = 1;

fseek(f1, -1, SEEK\_CUR);

}

else if (c == '#')

{

while ((c = fgetc(f1)) != EOF && c != '\n')

;

newLine();

}

else if (c == '\n')

{

newLine();

c = fgetc(f1);

if (c == '#')

{

while ((c = fgetc(f1)) != EOF && c != '\n')

;

newLine();

}

else if (c != EOF)

{

fseek(f1, -1, SEEK\_CUR);

}

}

else if (isspace(c))

{

++col;

}

else if (isalpha(c) || c == '\_')

{

tkn.row = row;

tkn.col = col++;

tkn.lexeme[0] = c;

int k = 1;

while ((c = fgetc(f1)) != EOF && isalnum(c))

{

tkn.lexeme[k++] = c;

++col;

}

tkn.lexeme[k] = '\0';

if (isKeyword(tkn.lexeme))

{

strcpy(tkn.type, "KEYWORD");

}

else

{

strcpy(tkn.type, "IDENTIFIER");

}

gotToken = 1;

fseek(f1, -1, SEEK\_CUR);

}

else if (c == '/')

{

int d = fgetc(f1);

++col; //Do we check EOF here?

if (d == '/')

{

while ((c = fgetc(f1)) != EOF && c != '\n')

{

++col;

}

if (c == '\n')

{

newLine();

}

}

else if (d == '\*')

{

do

{

if (d == '\n')

{

newLine();

}

while ((c == fgetc(f1)) != EOF && c != '\*')

{

++col;

if (c == '\n')

{

newLine();

}

}

++col;

} while ((d == fgetc(f1)) != EOF && d != '/' && (++col));

++col;

}

else

{

fillToken(&tkn, c, row, --col, "ARITHMETICOPERATOR");

gotToken = 1;

fseek(f1, -1, SEEK\_CUR);

}

}

else if (c == '"')

{

tkn.row = row;

tkn.col = col;

strcpy(tkn.type, "STRING LITERAL");

int k = 1;

tkn.lexeme[0] = '"';

while ((c = fgetc(f1)) != EOF && c != '"')

{

tkn.lexeme[k++] = c;

++col;

}

tkn.lexeme[k] = '"';

gotToken = 1;

}

else if (c == '<' || c == '>' || c == '!')

{

fillToken(&tkn, c, row, col, "RELATIONAL OPERATOR");

++col;

int d = fgetc(f1);

if (d == '=')

{

++col;

strcat(tkn.lexeme, "=");

}

else

{

if (c == '!')

{

strcpy(tkn.type, "LOGICAL OPERATOR");

}

fseek(f1, -1, SEEK\_CUR);

}

gotToken = 1;

}

else if (c == '&' || c == '|')

{

int d = fgetc(f1);

if (c == d)

{

tkn.lexeme[0] = tkn.lexeme[1] = c;

tkn.lexeme[2] = '\0';

tkn.row = row;

tkn.col = col;

++col;

gotToken = 1;

strcpy(tkn.type, "LOGICAL OPERATOR");

}

else

{

fseek(f1, -1, SEEK\_CUR);

}

++col;

}

else

{

++col;

}

}

return tkn;

}

int main()

{

FILE \*f1 = fopen("prog1InputFile.c", "r");

if (f1 == NULL)

{

printf("Error! File cannot be opened!\n");

return 0;

}

struct token tkn;

while ((tkn = getNextToken(f1)).row != -1)

{

printf("<%s, %d, %d, %s>\n", tkn.lexeme, tkn.row, tkn.col, tkn.type);

}

fclose(f1);

}

