Saivya Singh
CSE D 44
220905370

Lab 3 : Construction of Token Genarator

Q1.Write functions to identify the following tokens.

a. Arithmetic, relational and logical operators.

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

Code:

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

typedef struct token {
char token_name[10];
int row, col;
char type[10];
int index;
} token;

int main()
{
token tok;
char c, buf[10];
int row = 1, col = 1;
FILE *fp = fopen("digit.c", "r");
FILE *out_fp = fopen("q1a_out", "w");

if (fp == NULL)
{
printf("Cannot open input file \n");
exit(0);
}

if (out_fp == NULL)
{
printf("Cannot open output file \n");
exit(0);
}

c = fgetc(fp);
while (c != EOF)
{
int i = 0;
buf[0] = '\0';

if (c == '=')
{
```

```
buf[i++] = c;
c = fgetc(fp);
if (c == '=')
{
buf[i++] = c;
buf[i] = '\0';
strcpy(tok.token_name, buf);
tok.row = row;
tok.col = col;
strcpy(tok.type, "RelOP");
fprintf(out_fp, "< %s, %d, %d, %s >\n", tok.token_name, tok.row, tok.col, tok.type);
}
else
{
buf[i] = '\0';
strcpy(tok.token_name, buf);
tok.row = row;
tok.col = col;
strcpy(tok.type, "AssignOP");
fprintf(out_fp, "< %s, %d, %d, %s >\n", tok.token_name, tok.row, tok.col, tok.type);
}
}
else if (c == '&' || c == '|' || c == '!')
{
buf[i++] = c;
char dup = c;
c = fgetc(fp);

if (c == dup)
{
buf[i++] = c;
}
buf[i] = '\0';
strcpy(tok.token_name, buf);
tok.row = row;
tok.col = col;
strcpy(tok.type, "LogOP");
fprintf(out_fp, "< %s, %d, %d, %s >\n", tok.token_name, tok.row, tok.col, tok.type);
}
else if (c == '<' || c == '>' || c == '!')
{
buf[i++] = c;
c = fgetc(fp);

if (c == '=')
{
buf[i++] = c;
}
buf[i] = '\0';
strcpy(tok.token_name, buf);
tok.row = row;
tok.col = col;
strcpy(tok.type, "RelOP");
fprintf(out_fp, "< %s, %d, %d, %s >\n", tok.token_name, tok.row, tok.col, tok.type);
}
else
{
buf[i] = '\0';
}

c = fgetc(fp);
col++;

if (c == '\n')
{
row++;
col = 1;
}
}

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

B.

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

typedef struct token {
char token_name[20];
int row, col;
char type[20];
int index;
} token;

int main()
{
token tok;
char c, buf[20];
int row = 1, col = 1;
FILE *fp = fopen("digit.c", "r");
FILE *out_fp = fopen("q1b_out", "w");

if (fp == NULL) {
printf("Cannot open file \n");
exit(0);
}
```

```
if (out_fp == NULL) {
printf("Cannot open output file \n");
exit(0);
}

c = fgetc(fp);
while (c != EOF) {
int i = 0;
buf[0] = '\0';

if (c == '=' || c == '<' || c == '>' || c == '+' || c == '-' || c == '*' || c == '/') {
buf[i++] = c;
buf[i] = '\0';
strcpy(tok.token_name, buf);
tok.row = row;
tok.col = col;
strcpy(tok.type, "SpecialSymbol");
fprintf(out_fp, "< %s, %d, %d, %s >\n", tok.token_name, tok.row, tok.col, tok.type); // Write to file
} else if (c == '"' || isalpha(c) || c == '_') {
if (c == '"') {
i = 0;
c = fgetc(fp);
while (c != '"' && c != EOF) {
buf[i++] = c;
c = fgetc(fp);
}
buf[i] = '\0';
strcpy(tok.token_name, buf);
tok.row = row;
tok.col = col;
strcpy(tok.type, "StringLiteral");
fprintf(out_fp, "< %s, %d, %d, %s >\n", tok.token_name, tok.row, tok.col, tok.type); // Write to file
} else if (isalpha(c) || c == '_') {
i = 0;
while (isalnum(c) || c == '_') {
buf[i++] = c;
c = fgetc(fp);
}
buf[i] = '\0';
strcpy(tok.token_name, buf);
tok.row = row;
tok.col = col;
strcpy(tok.type, "Identifier");
fprintf(out_fp, "< %s, %d, %d, %s >\n", tok.token_name, tok.row, tok.col, tok.type); // Write to file
}
} else if (isdigit(c)) {
i = 0;
while (isdigit(c)) {
}
buf[i++] = c;
c = fgetc(fp);
}
if (c == '.') {
buf[i++] = c;
c = fgetc(fp);
while (isdigit(c)) {
buf[i++] = c;
c = fgetc(fp);
}
}
buf[i] = '\0';
strcpy(tok.token_name, buf);
tok.row = row;
tok.col = col;
strcpy(tok.type, "NumericConstant");
fprintf(out_fp, "< %s, %d, %d, %s >\n", tok.token_name, tok.row, tok.col, tok.type); // Write
to file
}

c = fgetc(fp);
col++;

if (c == '\n') {
row++;
col = 1;
}
}

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

Input:

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

Output:

A

```
< =, 4, 4, AssignOP >
< =, 5, 4, AssignOP >
< =, 6, 6, AssignOP >
```

B

```
< main, 1, 1, Identifier >
< int, 3, 2, Identifier >
< a, 3, 3, Identifier >
< b, 3, 4, Identifier >
< sum, 3, 5, Identifier >
< a, 4, 2, Identifier >
< =, 4, 3, SpecialSymbol >
< 1, 4, 5, NumericConstant >
< b, 5, 2, Identifier >
< =, 5, 3, SpecialSymbol >
< 1, 5, 5, NumericConstant >
< sum, 6, 2, Identifier >
< =, 6, 3, SpecialSymbol >
< a, 6, 5, Identifier >
< +, 6, 6, SpecialSymbol >
< b, 6, 8, Identifier >
```

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 token
type 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.

Code:

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

struct token {
char token_name[64];
int row, col;
char type[30];
};

static int row = 1, col = 1;
char specialsymbols[] = {'?', ';', ':', ',', '(', ')', '{', '}', '.'};
char *Keywords[] = {"for", "if", "else", "while", "do", "break", "continue", "return", "int",
"double", "float", "char",
"long", "short", "sizeof", "typedef", "switch", "case", "struct", "const", "void", "exit"};
```

```
char arithmeticsymbols[] = {'*','+','-','/', '%'};

int isKeyword(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, char *arr, int len) {
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->token_name[0] = c;
tkn->token_name[1] = '\0';
}

void newLine() {
++row;
col = 1;
}

struct token getNextToken(FILE *fin) {
int c, d;
struct token tkn = {.row = -1};
int gotToken = 0;

while (!gotToken && (c = getc(fin)) != EOF) {

if (c == '/') {
d = getc(fin);
if (d == '/') {
while ((c = getc(fin)) != EOF && c != '\n') {
++col;
}
if (c == '\n') {
newLine();
}
continue;
} else if (d == '*') {
do {
if (c == '\n') {
```

```
newLine();
}
while ((c = getc(fin)) != EOF && c != '*') {
++col;
}
if (c == '*') {
d = getc(fin);
}
} while (c != EOF && d != '/');
continue;
} else {
fseek(fin, -1, SEEK_CUR);
--col;
}
}

if (c == '\n') {
newLine();
continue;
}

if (charBelongsTo(c, specialsymbols, sizeof(specialsymbols) / sizeof(char))) {
fillToken(&tkn, c, row, col, (char[]){c, '\0'});
gotToken = 1;
++col;
}

else if (charBelongsTo(c, arithmeticsymbols, sizeof(arithmeticsymbols) / sizeof(char))) {
d = getc(fin);
if (d == '=' || (c == '+' || c == '-') && d == c) {
fillToken(&tkn, c, row, col, (char[]){c, c == '=' ? '=' : '\0', '\0'});
col += 2;
} else {
fillToken(&tkn, c, row, col, (char[]){c, '\0'});
++col;
fseek(fin, -1, SEEK_CUR);
}
gotToken = 1;
}

else if (c == '=' || c == '<' || c == '>' || c == '!') {
d = getc(fin);
if (d == '=') {
fillToken(&tkn, c, row, col, (char[]){c, '=', '\0'});
col += 2;
} else {
fillToken(&tkn, c, row, col, (char[]){c, '\0'});
++col;
fseek(fin, -1, SEEK_CUR);
}
gotToken = 1;
```

```
}

else if (isdigit(c)) {
tkn.row = row;
tkn.col = col++;
tkn.token_name[0] = c;
int k = 1;
while ((c = getc(fin)) != EOF && isdigit(c)) {
tkn.token_name[k++] = c;
++col;
}
tkn.token_name[k] = '\0';
strcpy(tkn.type, "Number");
gotToken = 1;
fseek(fin, -1, SEEK_CUR);
}

else if (isspace(c)) {
if (c == '\n') {
newLine();
} else {
++col;
}
continue;
}

else if (isalpha(c) || c == '_') {
tkn.row = row;
tkn.col = col++;
tkn.token_name[0] = c;
int k = 1;
while ((c = getc(fin)) != EOF && isalnum(c)) {
tkn.token_name[k++] = c;
++col;
}
tkn.token_name[k] = '\0';
strcpy(tkn.type, isKeyword(tkn.token_name) ? "Keyword" : "Identifier");
gotToken = 1;
fseek(fin, -1, SEEK_CUR);
}

else if (c == '"') {
tkn.row = row;
tkn.col = col;
strcpy(tkn.type, "StringLiteral");
int k = 1;
tkn.token_name[0] = '"';
while ((c = getc(fin)) != EOF && c != '"') {
tkn.token_name[k++] = c;
++col;
}
```

```
tkn.token_name[k] = '"';
gotToken = 1;
}

else {
++col;
}
}
return tkn;
}

int main() {
FILE *fin = fopen("testy.c", "r");
FILE *out_fp = fopen("q2_out", "w");
if (fin == NULL) {
printf("Unable to open the source file.\n");
return 1;
}

struct token tkn;
while (1) {
tkn = getNextToken(fin);
if (tkn.row == -1) break;

fprintf(out_fp, "<%s, %d, %d, %s>\n", tkn.token_name, tkn.row, tkn.col, tkn.type);
}

fclose(fin);
fclose(out_fp);
return 0;
}
```

Input:

```
#include <stdio.h>

int main() {
int a = 10;
printf("Hello, World!");
return 0;
}
```

Output:

```
<int, 3, 1, Keyword>
<main, 3, 5, Identifier>
<(, 3, 9, (>
<), 3, 10, )>
<{, 3, 12, {>
<int, 4, 5, Keyword>
<a, 4, 9, Identifier>
<=, 4, 11, =>
<10, 4, 13, Number>
<;, 4, 15, ;>
<printf, 5, 5, Identifier>
<(, 5, 11, (>
<"Hello, World!", 5, 12, StringLiteral>
<), 5, 25, )>
<;, 5, 26, ;>
<return, 6, 5, Keyword>
<0, 6, 12, Number>
<;, 6, 13, ;>
<}, 7, 1, }>
```