190905514 **MOHAMMAD TOFIK** **BATCH : C3**
ROLLNO : 62 **SECTION : C** **SEM : FIFTH**

- : CD-LAB-4 :-

1. Using getNextToken( ) implemented in Lab No 3,design a Lexical Analyser to implementlocal and global symbol table to store tokensfor identifiersusingarray of structure.

pgm1.c

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

const char *keywords[] = {"auto", "double", "int",
"struct", "break", "else",
"long", "switch", "case", "enum", "register", "typedef",
"char", "extern", "return",
"union", "continue", "for", "signed", "void", "do", "if",
"static", "while", "default",
"goto", "sizeof", "volatile", "const", "float", "short",
"unsigned", "printf", "scanf",
"true", "false"};

const char *dataTypes[] = {"int", "char", "void", "float",
"bool"};

int isdtype(char *ch)
{
int i;
for (i = 0; i < sizeof(dataTypes) / sizeof(char *); i++)
{
if (strcmp(ch, dataTypes[i]) == 0)
{
return 1;
}
}
return 0;
}
```

```
int isKeyword(char *ch)
{
int i;
for (i = 0; i < sizeof(keywords) / sizeof(char *); i++)
{
if (strcmp(ch, keywords[i]) == 0)
{
return 1;
}
}
return 0;
}

struct token
{
char lexeme[128];
unsigned int row, column;
char type[64];
};

struct sttable
{
int SINO;
char lexeme[128];
char dtype[64];
char type[64];
int size;
};

int findTable(struct sttable *table, char *name, int n)
{
int i = 0;
for (i = 0; i < n; i++)
{
if (strcmp(table[i].lexeme, name) == 0)
{
return 1;
}
}
return 0;
```

```
}

struct sttable fillTable(int SINO, char *lexn, char *dt,
char *t, int s)
{
struct sttable tab;
tab.SINO = SINO;
strcpy(tab.lexeme, lexn);
strcpy(tab.dtype, dt);
strcpy(tab.type, t);
tab.size = s;
return tab;
}

void printTable(struct sttable *table, int n)
{
for (int i = 0; i < n; i++)
{
printf("%d %s %s %s %d\n", table[i].SINO, table[i].lexeme,
table[i].dtype, table[i].type, table[i].size);
}
}

static int row = 1, column = 1;
char buf[2048];
char dbuf[128];
int ind = 0;
const char specialSymbols[] = {'?', ';', ':', ','};
const char arithmeticSymbols[] = {'*'};

int charIs(int c, const char *array)
{
int len;

if (array == specialSymbols)
{
len = sizeof(specialSymbols) / sizeof(char);
}

else if (array == arithmeticSymbols)
```

```
{
len = sizeof(arithmeticSymbols) / sizeof(char);
}

for (int i = 0; i < len; i++)
{
if (c == array[i])
{
return 1;
}
}
return 0;
}

void fillToken(struct token *token, char c, int row, int
col, char *type)
{
token->row = row;
token->column = col;
strcpy(token->type, type);
token->lexeme[0] = c;
token->lexeme[1] = '\0';
}

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

int sz(char *ch)
{
if (strcmp(ch, "int") == 0)
return 4;
if (strcmp(ch, "char") == 0)
return 1;
if (strcmp(ch, "void") == 0)
return 0;
if (strcmp(ch, "float") == 0)
return 8;
```

```
if (strcmp(ch, "bool") == 0)
return 1;
}

struct token getNextToken(FILE *fa)
{
int c;
struct token tkn =
{
.row = -1};

int gotToken = 0;

while (!gotToken && (c = fgetc(fa)) != EOF)
{
if (charIs(c, specialSymbols))
{
fillToken(&tkn, c, row, column, "SS");
gotToken = 1;
++column;
}

else if (charIs(c, arithmeticSymbols))
{
fseek(fa, -1, SEEK_CUR);
c = getc(fa);
if (isalnum(c))
{
fillToken(&tkn, c, row, column, "ARITHMETICOPERATOR");
gotToken = 1;
++column;
}

fseek(fa, 1, SEEK_CUR);
}

else if (c == '(')
{
fillToken(&tkn, c, row, column, "LB");
gotToken = 1;
```

```
column++;
}

else if (c == ')')
{
fillToken(&tkn, c, row, column, "RB");
gotToken = 1;
column++;
}

else if (c == '{')
{
fillToken(&tkn, c, row, column, "LC");
gotToken = 1;
column++;
}

else if (c == '}')
{
fillToken(&tkn, c, row, column, "RC");
gotToken = 1;
column++;
}

else if (c == '[')
{
fillToken(&tkn, c, row, column, "LS");
gotToken = 1;
column++;
}

else if (c == ']')
{
fillToken(&tkn, c, row, column, "RS");
gotToken = 1;
column++;
}

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

```
int x = fgetc(fa);

if (x != '+')
{
fillToken(&tkn, c, row, column, "ARITHMETICOPERATOR");
gotToken = 1;
column++;
fseek(fa, -1, SEEK_CUR);
}

else
{
fillToken(&tkn, c, row, column, "UNARYOPERATOR");
strcpy(tkn.lexeme, "++");
gotToken = 1;
column += 2;
}
}

else if (c == '-')
{
int x = fgetc(fa);

if (x != '-')
{
fillToken(&tkn, c, row, column, "ARITHMETICOPERATOR");
gotToken = 1;
column++;
fseek(fa, -1, SEEK_CUR);
}

else
{
fillToken(&tkn, c, row, column, "UNARYOPERATOR");
strcpy(tkn.lexeme, "++");
gotToken = 1;
column += 2;
}
}
```

```
else if (c == '=')
{
int x = fgetc(fa);

if (x != '=')
{
fillToken(&tkn, c, row, column, "ASSIGNMENTOPERATOR");
gotToken = 1;
column++;
fseek(fa, -1, SEEK_CUR);
}

else
{
fillToken(&tkn, c, row, column, "RELATIONALOPERATOR");
strcpy(tkn.lexeme, "++");
gotToken = 1;
column += 2;
}
}

else if (isdigit(c))
{
fillToken(&tkn, c, row, column++, "NUMBER");
int j = 1;

while ((c = fgetc(fa)) != EOF && isdigit(c))
{
tkn.lexeme[j++] = c;
column++;
}
tkn.lexeme[j] = '\0';
gotToken = 1;
fseek(fa, -1, SEEK_CUR);
}

else if (c == '#')
{
while ((c = fgetc(fa)) != EOF && c != '\n')
;
```

```
newLine();
}

else if (c == '\n')
{
newLine();
c = fgetc(fa);

if (c == '#')
{
while ((c = fgetc(fa)) != EOF && c != '\n')
;
newLine();
}

else if (c != EOF)
{
fseek(fa, -1, SEEK_CUR);
}
}

else if (isspace(c))
{
++column;
}

else if (isalpha(c) || c == '_')
{
tkn.row = row;
tkn.column = column++;
tkn.lexeme[0] = c;
int j = 1;

while ((c = fgetc(fa)) != EOF && isalnum(c))
{
tkn.lexeme[j++] = c;
column++;
}
tkn.lexeme[j] = '\0';
```

```
if (isKeyword(tkn.lexeme))
{
strcpy(tkn.type, "KEYWORD");
}

else
{
strcpy(tkn.type, "IDENTIFIER");
}
gotToken = 1;
fseek(fa, -1, SEEK_CUR);
}

else if (c == '/')
{
int d = fgetc(fa);
++column;
if (d == '/')
{
while ((c = fgetc(fa)) != EOF && c != '\n')
{
++column;
}
if (c == '\n')
{
newLine();
}
}

else if (d == '*')
{
do
{
if (d == '\n')
{
newLine();
}

while ((c == fgetc(fa)) != EOF && c != '*')
{
```

```
++column;
if (c == '\n')
{
newLine();
}
}
++column;
} while ((d == fgetc(fa)) != EOF && d != '/' && (+
+column));
++column;
}

else
{
fillToken(&tkn, c, row, --column, "ARITHMETIC OPERATOR");
gotToken = 1;
fseek(fa, -1, SEEK_CUR);
}
}

else if (c == '"')
{
tkn.row = row;
tkn.column = column;
strcpy(tkn.type, "STRING LITERAL");
int k = 1;
tkn.lexeme[0] = '"';
while ((c = fgetc(fa)) != EOF && c != '"')
{
tkn.lexeme[k++] = c;
++column;
}
tkn.lexeme[k] = '"';
gotToken = 1;
}

else if (c == '<' || c == '>' || c == '!')
{
fillToken(&tkn, c, row, column, "RELATIONALOPERATOR");
++column;
```

```
int d = fgetc(fa);

if (d == '=')
{
++column;
strcat(tkn.lexeme, "=");
}

else
{
if (c == '!')
{
strcpy(tkn.type, "LOGICALOPERATOR");
}
fseek(fa, -1, SEEK_CUR);
}
gotToken = 1;
}

else if (c == '&' || c == '|')
{
int d = fgetc(fa);
if (c == d)
{
tkn.lexeme[0] = tkn.lexeme[1] = c;
tkn.lexeme[2] = '\0';
tkn.row = row;
tkn.column = column;
++column;
gotToken = 1;
strcpy(tkn.type, "LOGICALOPERATOR");
}
else
{
fseek(fa, -1, SEEK_CUR);
}
++column;
}
else
{
```

```
++column;
}
}
return tkn;
}

int main()
{
FILE *fa, *fb;
int ca, cb;
fa = fopen("input.c", "r");

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

fb = fopen("output.c", "w+");
ca = getc(fa);

while (ca != EOF)
{
if (ca == ' ')
{
putc(ca, fb);

while (ca == ' ')
ca = getc(fa);
}

if (ca == '/')
{
cb = getc(fa);

if (cb == '/')
{
while (ca != '\n')
ca = getc(fa);
}
```

```
else if (cb == '*')
{
do
{
while (ca != '*')
ca = getc(fa);
ca = getc(fa);
} while (ca != '/');
}

else
{
putc(ca, fb);
putc(cb, fb);
}
}

else
putc(ca, fb);
ca = getc(fa);
}

fclose(fa);
fclose(fb);
fa = fopen("output.c", "r");

if (fa == NULL)
{
printf("Cannot open file");
return 0;
}

fb = fopen("temp.c", "w+");
ca = getc(fa);

while (ca != EOF)
{
if (ca == '"')
{
```

```
putc(ca, fb);
ca = getc(fa);
while (ca != '"')
{
putc(ca, fb);
ca = getc(fa);
}
}

else if (ca == '#')
{

while (ca != '\n')
{

ca = getc(fa);
}
ca = getc(fa);
}
putc(ca, fb);
ca = getc(fa);
}

fclose(fa);
fclose(fb);

fa = fopen("input.c", "r");
fb = fopen("output.c", "w");
ca = getc(fa);

while (ca != EOF)
{
putc(ca, fb);
ca = getc(fa);
}

fclose(fa);
fclose(fb);
remove("temp.c");
FILE *f1 = fopen("output.c", "r");
```

```
if (f1 == NULL)
{
printf("Error! File cannot be opened!\n");
return 0;
}

struct token tkn;
struct sttable st[10][100];
int flag = 0, i = 0, j = 0;
int tabsz[10];
char w[25];
w[0] = '\0';

while ((tkn = getNextToken(f1)).row != -1)
{
printf("<%s, %d, %d>\n", tkn.lexeme, tkn.row, tkn.column);

if (strcmp(tkn.type, "KEYWORD") == 0)
{
if (isdtype(tkn.lexeme) == 1)
{
strcpy(dbuf, tkn.lexeme);
}
}

else if (strcmp(tkn.type, "IDENTIFIER") == 0)
{
strcpy(w, tkn.lexeme);
tkn = getNextToken(f1);
printf("<%s, %d, %d>\n", tkn.lexeme, tkn.row, tkn.column);

if ((strcmp(tkn.type, "LB")) == 0)
{
if (findTable(st[i], w, j) == 0)
{
ind++;
st[i][j++] = fillTable(ind, w, dbuf, "func", -1);
}
}
```

```
if ((strcmp(tkn.type, "LS")) == 0)
{
if (findTable(st[i], w, j) == 0)
{
tkn = getNextToken(f1);
printf("<%s, %d, %d>\n", tkn.lexeme, tkn.row, tkn.column);

int s = 0;
if (strcmp(tkn.type, "NUMBER") == 0)
{
s = atoi(tkn.lexeme);
}
ind++;
st[i][j++] = fillTable(ind, w, dbuf, "id", sz(dbuf) * s);
}
}

else
{
if (findTable(st[i], w, j) == 0)
{
ind++;
st[i][j++] = fillTable(ind, w, dbuf, "id", sz(dbuf));
}
}
}

else if (strcmp(tkn.type, "LC") == 0)
{
flag++;
}

else if (strcmp(tkn.type, "RC") == 0)
{
flag--;
if (flag == 0)
{
tabsz[i] = j;
i++;
```

```
j = 0;
ind = 0;
}
}
}

int k = 0;

for (k = 0; k < i; k++)
{
printTable(st[k], tabsz[k]);
}
fclose(f1);
}
```

OUTPUT

```
@lplab-ThinkCentre-M71e: ~/Documents/190905514/CD_LAB/LAB4
student@lplab-ThinkCentre-M71e:~/Documents/190905514/CD_LAB/LAB4$ gcc -o lab4 lab4.c
student@lplab-ThinkCentre-M71e:~/Documents/190905514/CD_LAB/LAB4$ ./lab4
<int, 2, 1>
<main, 2, 5>
<(, 2, 9>
<), 2, 10>
<{, 2, 12>
<int, 3, 4>
<a, 3, 8>
<,, 3, 9>
<b, 3, 11>
<,, 3, 12>
<sum, 3, 14>
<;, 3, 17>
<printf, 4, 4>
<(, 4, 10>
<"\nJust a sample program. I am Mohammad tofik.", 4, 11>
<), 4, 56>
<;, 4, 57>
<printf, 5, 4>
<(, 5, 10>
<"\nEnter two no: ", 5, 11>
<), 5, 27>
<;, 5, 28>
<scanf, 6, 4>
<(, 6, 9>
<"%d %d", 6, 10>
<,, 6, 15>
<a, 6, 18>
<,, 6, 19>
<b, 6, 22>
<), 6, 23>
<;, 6, 24>
<sum, 8, 4>
<=, 8, 8>
<a, 8, 10>
<+, 8, 12>
<b, 8, 14>
<;, 8, 15>
<printf, 10, 4>
<(, 10, 10>
<"Sum : %d", 10, 11>
<,, 10, 19>
```

```
@lplab-ThinkCentre-M71e: ~/Documents/190905514/CD_LAB/LAB4
<a, 6, 18>
<,, 6, 19>
<b, 6, 22>
<), 6, 23>
<;, 6, 24>
<sum, 8, 4>
<=, 8, 8>
<a, 8, 10>
<+, 8, 12>
<b, 8, 14>
<;, 8, 15>
<printf, 10, 4>
<(, 10, 10>
<"Sum : %d", 10, 11>
<,, 10, 19>
<sum, 10, 21>
<), 10, 24>
<;, 10, 25>
<return, 12, 4>
<(, 12, 10>
<0, 12, 11>
<), 12, 12>
<;, 12, 13>
<}, 13, 1>
1 main int func -1
2 a int id 4
3 b int id 4
4 sum int id 4
student@lplab-ThinkCentre-M71e:~/Documents/190905514/CD_LAB/LAB4$ cat sample.c
#include<stdio.h>
int main() {
   int a, b, sum;
   printf("\nJust a sample program. I am Mohammad tofik.");
   printf("\nEnter two no: ");
   scanf("%d %d", &a, &b);

   sum = a + b;

   printf("Sum : %d", sum);

   return(0);
}
student@lplab-ThinkCentre-M71e:~/Documents/190905514/CD_LAB/LAB4$
```