CD Lab4

Name: Rhea Adhikari
Class :CSE-D
Batch1
190905156

Q)Using getNextToken( ) implemented in Lab No 3,design a Lexical Analyser to implement local and global symbol table to store tokens for identifiers using array of structure.

first.c

```
// search if lexene already available or not
// if not there then make an entry
// name of lexene,data type,size (use sizeof)
// int,func
// use array of structures

// Output
// tokens generated from input and symbol table

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

int cleanup(){

 char c1, c2;
 FILE * f1, * f2;
 f1 = fopen("sample.c", "r");
 printf("open sample.c\n");
 f2 = fopen("output.c", "w");
  printf("open/create output.c\n");
 if (f1 == NULL || f2 == NULL) {
   printf("Either the input or the output file does not exist \n");
   return 1;
 }

 // remove directives
 c1 = fgetc(f1);
 while (c1 != EOF) {
   if (c1 == '/') {
     c2 = getc(f1);
     if (c2 == '/') {
       putc(c1, f2);
       putc(c2, f2);
       c1 = getc(f1);
       while (c1 != '\n') {
         putc(c1, f2);
         c1 = getc(f1);
```

```
    }
   } else if (c2 == '*') {
    putc(c1, f2);
    putc(c2, f2);
    c1 = getc(f1);
    do {
     while (c1 != '*') {
      putc(c1, f2);
      c1 = getc(f1);
     }
     putc(c1, f2);
     c1 = getc(f1);
    } while (c1 != '/');
   }
  }
  if (c1 == '"') {
   putc(c1, f2);
   c1 = getc(f1);
   while (c1 != '"') {
    putc(c1, f2);
    c1 = getc(f1);
   }
   putc(c1, f2);
   c1 = getc(f1);
  }
  if (c1 == '#') {
   while (c1 != '\n')
    c1 = getc(f1);
  }
  putc(c1, f2);
  c1 = getc(f1);
}

// remove spaces
c1 = fgetc(f1);

while (c1 != EOF) {
  if (c1 == '/') {
   c2 = getc(f1);
   if (c2 == '/') {
    putc(c1, f2);
    putc(c2, f2);
    c1 = getc(f1);
    while (c1 != '\n') {
     putc(c1, f2);
     c1 = getc(f1);
    }
   } else if (c2 == '*') {
    putc(c1, f2);
    putc(c2, f2);
    c1 = getc(f1);
    do {
```

```
        while (c1 != '*') {
          putc(c1, f2);
          c1 = getc(f1);
        }
        putc(c1, f2);
        c1 = getc(f1);
      } while (c1 != '/');
    }
   }
   if (c1 == '"') {
    putc(c1, f2);
    c1 = getc(f1);
    while (c1 != '"') {
     putc(c1, f2);
     c1 = getc(f1);
    }
    putc(c1, f2);
    c1 = getc(f1);
   }
   if (c1 == ' ' || c1 == '\t') {
    putc(' ', f2);
    while (c1 == ' ' || c1 == '\t') {
     c1 = getc(f1);
    }
   }
   putc(c1, f2);
   c1 = getc(f1);
  }

  fclose(f1);
  printf("closed sample.c\n");
  fclose(f2);
  printf("closed output.c\n");
  printf("output.c has cleanup code which can be used further.\n");
}

int counter = 0;
char typeBuffer[20];

int i, row, col;
char arithsymbols[] = {'*', '%'};
char *keywords[] = {"int", "return", "for", "while", "if", "else","printf", "case", "break",
"float",
"const", "bool"};
char spsymbols[] = {':', ',', '.','?', ';'};

struct token {
  int index;
  unsigned int row, col;
  char token_name[];
};
```

```
struct symboltable {
  char name[20];
  char type[20];
  int size;
}
st[20];

void tokenInput(struct token * tok, char c, int row, int col) {
  tok -> token_name[0] = c;
  tok -> index = 0;
  tok -> row = row;
  tok -> col = col;
}

int isSpecialSymbol(char ch) {
  for (i = 0; i < 5; i++) {
    if (spsymbols[i] == ch) {
      return 1;
    }
  }
  return 0;
}

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

int isArithmeticSymbol(char ch) {
  for (i = 0; i < 4; i++) {
    if (arithsymbols[i] == ch) {
      return 1;
    }
  }
  return 0;
}

int isKeyword(char * key) {
  for (i = 0; i < 12; i++) {
    if (strcmp(key, keywords[i]) == 0) {
      return 1;
    }
  }
  return 0;
}

/***************************************/
int search(char * name)
{
  for (int i = 0; i < counter; i++) {
    if (strcmp(name, st[i].name) == 0) {
      return 1;
```

```
    }
  }
  return 0;
}

void insertIntoSymbolTable(char * name, char * type)
{
  if (search(name) == 0)
  {
    strcpy(st[counter].name, name);
    strcpy(st[counter].type, type);
    printf("%s", st[counter].type);
    if (strcmp(type, "func") == 0) {
      st[counter].size = -999;
    } else {
      st[counter].size = 4;
    }
    counter++;
  }
}
/****************************************/

struct token getNextToken(FILE * f, char * name) {
  char c;
  struct token tkn;
  tkn.row = -1;
  int gotToken = 0;
  //if we find the token then out of loop
  //OR
  //if we reach eof then out of loop
  while (!gotToken && (c = fgetc(f)) != EOF) {
    if (isSpecialSymbol(c) || isArithmeticSymbol(c)) {
      tokenInput( & tkn, c, row, col);
      gotToken = 1;
      ++col;
    } else if (c == '(' || c == ')' || c == '}' || c == '{' || c == ']' || c == '[') {
      tokenInput( & tkn, c, row, col);
      gotToken = 1;
      ++col;
    } else if (c == '+') {
      int next = fgetc(f);
      if (next != '+') {
        tokenInput( & tkn, c, row, col);
        gotToken = 1;
        ++col;
        fseek(f, -1, SEEK_CUR);
      } else {
        tokenInput( & tkn, c, row, col);
        strcpy(tkn.token_name, "++");
        gotToken = 1;
        col += 2;
```

```
  }
 } else if (c == '-') {
  int next = fgetc(f);
  if (next != '-') {
   tokenInput( & tkn, c, row, col);
   gotToken = 1;
   ++col;
   fseek(f, -1, SEEK_CUR);
  } else {
   tokenInput( & tkn, c, row, col);
   strcpy(tkn.token_name, "--");
   gotToken = 1;
   col += 2;

  }
 } else if (c == '=') {
  int next = fgetc(f);
  if (next != '=') {
   tokenInput( & tkn, c, row, col);
   gotToken = 1;
   ++col;
   fseek(f, -1, SEEK_CUR);
  } else {
   tokenInput( & tkn, c, row, col);
   strcpy(tkn.token_name, "==");
   gotToken = 1;
   col += 2;

  }
 } else if (isdigit(c)) {
  tkn.row = row;
  tkn.col = col++;
  strcpy(tkn.token_name, "num");
  while ((c = fgetc(f)) != EOF && isdigit(c)) {
   col++;
  }
  gotToken = 1;
  fseek(f, -1, SEEK_CUR);
 } else if (c == '#') {
  while ((c = fgetc(f)) != EOF && c != '\n') {
   continue;
  }
  newLine();
 } else if (c == '\n') {
  newLine();
  c = fgetc(f);
  if (c == '#') {
   while ((c = fgetc(f)) != EOF && c != '\n')
   ;
   newLine();
  } else if (c != EOF)
   fseek(f, -1, SEEK_CUR);
```

```
} else if (isspace(c))
  ++col;
 else if (isalpha(c) || c == '_') {
  tkn.row = row;
  tkn.col = col++;
  tkn.token_name[0] = c;
  int k = 1;
  while ((c = fgetc(f)) != EOF && isalnum(c)) {
    tkn.token_name[k++] = c;
    ++col;
  }
  tkn.token_name[k] = '\0';
  int flag = 1;
  for (int i = 0; i < 12; i++) {
    if (strcmp(tkn.token_name, keywords[i]) == 0) {
      if ((strcmp(tkn.token_name, "int") == 0) || strcmp(tkn.token_name, "float") == 0) {
        strcpy(typeBuffer, tkn.token_name);
      }
      flag = 0;
      break;
    }
  }

  if (flag == 1) {
    if (c == '(') {
      printf("Function Found : %s", tkn.token_name);
      char nm[10];
      strcpy(nm, "func");
      insertIntoSymbolTable(tkn.token_name, nm);
    } else {
      insertIntoSymbolTable(tkn.token_name, typeBuffer);
    }
    strcpy(tkn.token_name, "ID");
  }
  gotToken = 1;
  fseek(f, -1, SEEK_CUR);
 }

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

```
        if (c == '\n')
          newLine();
       }
       ++col;
     } while ((d == fgetc(f)) != EOF && d != '/' && (++col));
     ++col;
   } else {
     tokenInput( & tkn, c, row, --col);
     gotToken = 1;
     fseek(f, -1, SEEK_CUR);
   }
 } else if (c == '"') {
   tkn.row = row;
   tkn.col = col;
   int k = 1;
   tkn.token_name[0] = '"';
   while ((c = fgetc(f)) != EOF && c != '"') {
     tkn.token_name[k++] = c;
     ++col;
   }
   tkn.token_name[k] = '"';
   gotToken = 1;
 }

 else if (c == '<' || c == '>' || c == '!') {
   tokenInput( & tkn, c, row, col);
   ++col;
   int c = fgetc(f);
   if (c == '=') {
     ++col;
     strcat(tkn.token_name, "=");
   } else {
     fseek(f, -1, SEEK_CUR);
   }
   gotToken = 1;
 } else if (c == '&' || c == '|') {
   int d = fgetc(f);
   if (c == d)
   {
     tkn.token_name[0] = tkn.token_name[1] = c;
     tkn.token_name[2] = '\0';
     tkn.row = row;
     tkn.col = col;
     ++col;
     gotToken = 1;
   } else
   {
     tkn.token_name[0] = c;
     tkn.token_name[1] = '\0';
     tkn.row = row;
     tkn.col = col;
     ++col;
```

```
        gotToken = 1;
        fseek(f, -1, SEEK_CUR);
      }
      ++col;
    } else
      ++col;
  }
  strcpy(name, tkn.token_name);
  return tkn;
}
int main() {
  int cleaned=cleanup();
  char c, buf[10];
  // printf("Enter file name: ");
  char input[256]="output.c";
  // scanf("%s", input);
  FILE * fp = fopen(input, "r");
  if (fp == NULL) {
    printf("Cannot open file\n");
    exit(0);
  }
  struct token tok;
  char nm[100];
  while ((tok = getNextToken(fp, nm)).row != -1) {
    printf("<%s, %d, %d>\n", nm, tok.row, tok.col);
  }
  printf("SYMBOL TABLE:\n");
  printf("\nName\t Type\t Size\n");
  printf("********************\n");
  for (int j = 0; j < counter; j++) {
    printf("%s \t %s \t", st[j].name, st[j].type);
    if (st[j].size == -999) {
      printf("NULL\n");
    } else {
      printf(" %d \n", st[j].size);
    }
  }
  fclose(fp);
  return 0;
}
```

OUTPUT SCREENSHOTS (NEXT PAGE)

```
Terminal                                                    Sat 16:59
                                                ugcse@prg28: ~/Documents/190905156/Lab4
File  Edit  View  Search  Terminal  Help
       strcpy(tkn.token_name, "num");
       ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~
first.c:331:9: warning: '__builtin_memcpy' writing 3 bytes into a region of size 0 overflows the destination [-Wstringop-overflow=]
         strcpy(tkn.token_name, "ID");
         ^~~~~~~~~~~~~~~~~~~~~~~~~~~~
ugcse@prg28:~/Documents/190905156/Lab4$ ./first
open sample.c
open/create output.c
closed sample.c
closed output.c
output.c has cleanup code which can be used further.
<int, 3, 1>
Function Found : mainfunc<ID, 3, 5>
<(, 3, 9>
<), 3, 10>
<{, 3, 11>
<int, 4, 3>
int<ID, 4, 7>
<=, 4, 8>
<num, 4, 9>
<;, 4, 11>
<for, 5, 4>
<(, 5, 7>
<int, 5, 8>
int<ID, 5, 12>
<=, 5, 13>
<num, 5, 14>
<;, 5, 15>
<ID, 5, 16>
<<, 5, 17>
<ID, 5, 18>
<;, 5, 19>
<ID, 5, 20>
<++, 5, 21>
<), 5, 23>
<{, 5, 26>
<printf, 6, 2>
<(, 6, 8>
<"%d"[00 0a] 6, 11>
<,, 6, 13>
<ID, 6, 14>
<), 6, 15>
<;, 6, 16>
<}, 7, 3>
<}, 8, 1>
SYMBOL TABLE:

Name     Type    Size
*********************
main     func    NULL
a        int     4
i        int     4
ugcse@prg28:~/Documents/190905156/Lab4$ cat output.c
```

```
ugcse@prg28:~/Documents/190905156/Lab4$ cat output.c
int main() {
  int a = 10;
  for (int i = 1; i < a; i++) {
    printf("%d", i);
  }
}ugcse@prg28:~/Documents/190905156/Lab4$ cat sample.c
#include <stdio.h>
#include <stdlib.h>

int main(){
        int a=10;
                for(int i=1;i<a;i++)            {
 printf(                "%d",i);
        }
ugcse@prg28:~/Documents/190905156/Lab4$ ==
```