LAB 4

Juhi Mehta
190905412
Roll No: 55
Batch B3

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

Code:

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

struct symbolTable{
   char name[20];
   char type[20];
   int size;
};

struct symbolTable st[20];
int counter = 0;

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

void insert(char* name , char* type)
{
   if(search(name) == 0)
   {
      strcpy(st[counter].name,name);
      strcpy(st[counter].type,type);
      if(strcmp(type,"func") == 0)
      {
         st[counter].size = -1;
      }
```

```
        else
            st[counter].size = 4;
        counter++;
    }
}

struct token
{
    char lexeme[64];
    unsigned int row, col;
};

static int row = 1, col = 1;
char buf[2048];

char specialsymbols[] = {'?', ';', ':', ',', '.'};
char latestType[10];

const char *keywords[] = {"int","float", "return", "for", "while", "if", "else","printf"};

char arithmeticsymbols[] = {'*', '%'};

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

int charBelongsTo(int c, char *arr)
{
    int len = 0;
    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 *t, char c, int row, int col)
{
    t->lexeme[0] = c;
    t->lexeme[1] = '\0';
    t->row = row;
    t->col = col;
}

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

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

    int gotToken = 0;
    while (!gotToken && (c = fgetc(f)) != EOF)
    {
        if (charBelongsTo(c, specialsymbols))
        {
            fillToken(&tkn, c, row, col);
            gotToken = 1;
            ++col;
        }
        else if (charBelongsTo(c, arithmeticsymbols))
        {
            fillToken(&tkn, c, row, col);
            gotToken = 1;
            ++col;
        }
        else if (c == '(')
        {
            fillToken(&tkn, c, row, col);
            gotToken = 1;
            ++col;
        }
        else if (c == ')')
        {
            fillToken(&tkn, c, row, col);
            gotToken = 1;
            ++col;
        }
        else if (c == '{')
        {
            fillToken(&tkn, c, row, col);
            gotToken = 1;
            ++col;
        }
        else if (c == '}')
        {
```

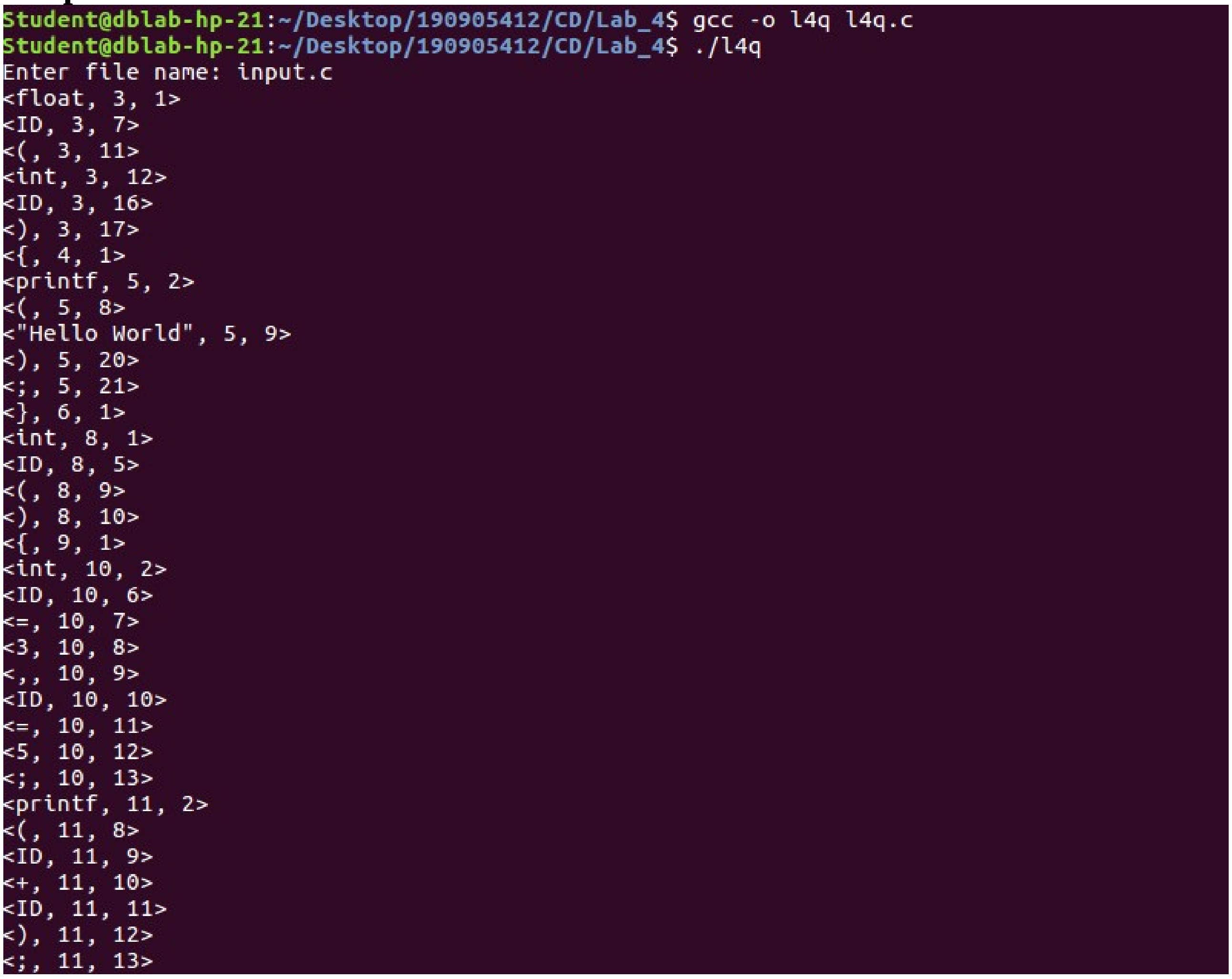
```
    fillToken(&tkn, c, row, col);
    gotToken = 1;
    ++col;
}
else if (c == '[')
{
    fillToken(&tkn, c, row, col);
    gotToken = 1;
    ++col;
}
else if (c == ']')
{
    fillToken(&tkn, c, row, col);
    gotToken = 1;
    ++col;
}
else if (c == '+')
{
    int d = fgetc(f);
    if (d != '+')
    {
        fillToken(&tkn, c, row, col);
        gotToken = 1;
        ++col;
        fseek(f, -1, SEEK_CUR); //go back 1 step *
    }
    else
    {
        fillToken(&tkn, c, row, col);
        strcpy(tkn.lexeme, "++");
        gotToken = 1;
        col += 2; //skip next as it is already included
    }
}
else if (c == '-')
{
    int d = fgetc(f);
    if (d != '-')
    {
        fillToken(&tkn, c, row, col);
        gotToken = 1;
        ++col;
        fseek(f, -1, SEEK_CUR); //go back 1 step *
    }
    else
    {
        fillToken(&tkn, c, row, col);
        strcpy(tkn.lexeme, "--");
        gotToken = 1;
        col += 2; //skip next as it is already included
    }
}
```

```
else if (c == '=')
{
   int d = fgetc(f);
   if (d != '=')
   {
      fillToken(&tkn, c, row, col);
      gotToken = 1;
      ++col;
      fseek(f, -1, SEEK_CUR); //go back 1 step *
   }
   else
   {
      fillToken(&tkn, c, row, col);
      strcpy(tkn.lexeme, "==");
      gotToken = 1;
      col += 2; //skip next as it is already included
   }
}
else if (isdigit(c))
{
   tkn.row = row;
   tkn.col = col++;
   tkn.lexeme[0] = c;
   int k = 1;
   while ((c = fgetc(f)) != EOF && isdigit(c))
   {
      tkn.lexeme[k++] = c;
      col++;
   }
   tkn.lexeme[k] = '\0';
   gotToken = 1;
   fseek(f, -1, SEEK_CUR); //go back 1 step *
}
else if (c == '#')
{
   while ((c = fgetc(f)) != EOF && c != '\n')
      ;
   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);
}
```

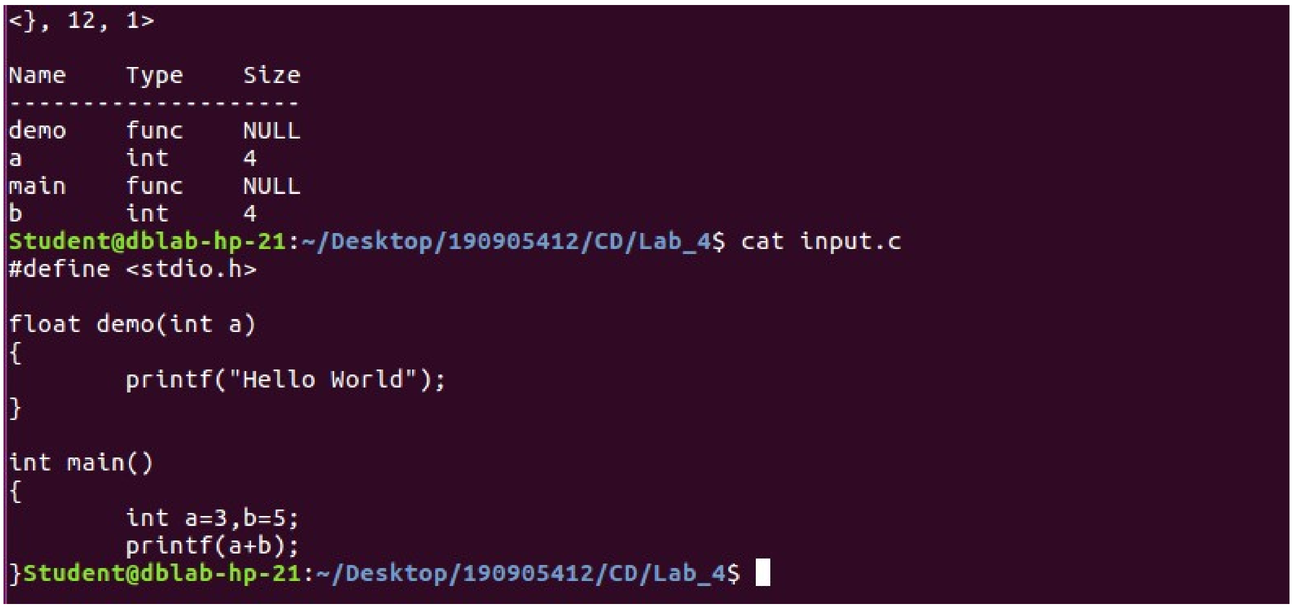
```
else if (isspace(c))
    ++col;
else if (isalpha(c) || c == '_')
{
    tkn.row = row;
    tkn.col = col++;
    tkn.lexeme[0] = c;
    int k = 1;
    while ((c = fgetc(f)) != EOF && isalnum(c))
    {
        tkn.lexeme[k++] = c;
        ++col;
    }
    tkn.lexeme[k] = '\0';
    int check = 1;
    for(int i = 0 ; i < 8 ; i++){
        if(strcmp(tkn.lexeme,keywords[i]) == 0){
            if(strcmp(tkn.lexeme,"int") == 0 || strcmp(tkn.lexeme,"float") == 0){
                strcpy(latestType,tkn.lexeme);
            }
            check = 0;
            break;
        }
    }
    if(check == 1){
        if(c == '('){
            insert(tkn.lexeme,"func");
        }
        else{
            insert(tkn.lexeme,latestType);
        }
        strcpy(tkn.lexeme,"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
    {
        fillToken(&tkn, c, row, --col);
        gotToken = 1;
        fseek(f, -1, SEEK_CUR);
    }
}
else if (c == '"')
{
    tkn.row = row;
    tkn.col = col;
    int k = 1;
    tkn.lexeme[0] = '"';
    while ((c = fgetc(f)) != EOF && c != '"')
    {
        tkn.lexeme[k++] = c;
        ++col;
    }
    tkn.lexeme[k] = '"';
    gotToken = 1;
}
else if (c == '<' || c == '>' || c == '!')
{
    fillToken(&tkn, c, row, col);
    ++col;
    int d = fgetc(f);
    if (d == '=')
    {
        ++col;
        strcat(tkn.lexeme, "=");
    }
    else
    {
        fseek(f, -1, SEEK_CUR);
    }
    gotToken = 1;
}
else if (c == '&' || c == '|')
{
    int d = fgetc(f);
```

```
            if (c == d)
            {
                tkn.lexeme[0] = tkn.lexeme[1] = c;
                tkn.lexeme[2] = '\0';
                tkn.row = row;
                tkn.col = col;
                ++col;
                gotToken = 1;
            }
            else
            {
                tkn.lexeme[0] = c;
                tkn.lexeme[1] = '\0';
                tkn.row = row;
                tkn.col = col;
                ++col;
                gotToken = 1;
                fseek(f, -1, SEEK_CUR);
            }
            ++col;
        }
        else
            ++col;
    }
    return tkn;
}

int main()
{
    printf("Enter file name: ");
    char input[256];
    scanf("%s", input);
    FILE *f = fopen(input, "r");
    if (f == NULL)
    {
        printf("Cannot open file\n");
        exit(0);
    }

    struct token t;
    while ((t = getNextToken(f)).row != -1)
        printf("<%s, %d, %d>\n", t.lexeme, t.row, t.col);
    printf("\nName\tType\tSize\n");
    printf("--------------------\n");
    for(int i = 0 ; i < counter ; i++)
    {
        printf("%s\t%s\t",st[i].name,st[i].type);
        if(st[i].size == -1)
        {
            printf("NULL\n");
        }
        else
        printf("%d\n",st[i].size);
    }
    fclose(f);
    return 0;
}
```

Output:



```
Student@dblab-hp-21:~/Desktop/190905412/CD/Lab_4$ gcc -o l4q l4q.c
Student@dblab-hp-21:~/Desktop/190905412/CD/Lab_4$ ./l4q
Enter file name: input.c
<float, 3, 1>
<ID, 3, 7>
<(, 3, 11>
<int, 3, 12>
<ID, 3, 16>
<), 3, 17>
<{, 4, 1>
<printf, 5, 2>
<(, 5, 8>
<"Hello World", 5, 9>
<), 5, 20>
<;, 5, 21>
<}, 6, 1>
<int, 8, 1>
<ID, 8, 5>
<(, 8, 9>
<), 8, 10>
<{, 9, 1>
<int, 10, 2>
<ID, 10, 6>
<=, 10, 7>
<3, 10, 8>
<,, 10, 9>
<ID, 10, 10>
<=, 10, 11>
<5, 10, 12>
<;, 10, 13>
<printf, 11, 2>
<(, 11, 8>
<ID, 11, 9>
<+, 11, 10>
<ID, 11, 11>
<), 11, 12>
<;, 11, 13>
```



```
<}, 12, 1>

Name    Type    Size
--------------------
demo    func    NULL
a       int     4
main    func    NULL
b       int     4
Student@dblab-hp-21:~/Desktop/190905412/CD/Lab_4$ cat input.c
#define <stdio.h>

float demo(int a)
{
        printf("Hello World");
}

int main()
{
        int a=3,b=5;
        printf(a+b);
}Student@dblab-hp-21:~/Desktop/190905412/CD/Lab_4$ 
```