National University of Sciences and Technology (NUST)
School of Electrical Engineering and Computer Science

Department of Computing

CS 354: Compiler Construction

Class: BSCS-6C

Lab [05]: Lexical Analysis with Flex

Date: 5th Nov, 2019

Time: [2:00pm – 5:00pm]

Group Lab

Abdul Ghaffar Kalhoro 194699

Ahmad Amjad Mughal 121672

BSCS-6C

National University of Sciences and Technology (NUST)
School of Electrical Engineering and Computer Science

Lab [04]: Lexical Analysis with Flex

Introduction

The lexical analyzer is the part of the compiler that reads the source text, it may also perform certain secondary tasks at the user interface. One such task is stripping out comments and white space in the form of blanks, tabs and new line characters, from the source program. Another is correlating error messages from the compiler with the source program i.e. keeping a correspondence between errors and source line numbers.

Objectives

1. Successful understanding/implementation of basic Lexical Analysis using flex

Tools/Software Requirement

1. flex on Linux or Windows platform

Description

Lexical analysis is the process of converting a sequence of characters into a sequence of tokens. A program or function which performs lexical analysis is called a lexical analyzer, lexer or scanner. A lexer often exists as a single function which is called by a parser or another function.

Lab Tasks

- **Flex in a Nutshell (tutorial):** Go through the flex.
- Write a flex program to process a pascal-like toy language with the following specifications:
    - Match integers and floating point constants
    - Match Identifiers, starting with lower-case alphabets and allowing for integers in non-starting locations.
    - Keywords: if, then, begin, end, procedure, function
    - Operators: +, -, *, /
    - Skipping of white-space characters i.e. new-line, tabs and spaces
    - Printing of un-recognized characters

Use the following example code to test your lexical analyzer.

```
procedure compute
begin
   area = 3.141 * radius * radius
end
function main
begin
   compute
end
```

Your output should resemble:

```
A keyword: procedure
An identifier: compute
A keyword: begin
An identifier: area
Unrecognized character: =
A float: 3.141 (3.141)
An operator: *
An identifier: radius
An operator: *
An identifier: radius
A keyword: end
A keyword: function
An identifier: main
A keyword: begin
An identifier: compute
A keyword: end
```

- **Postfix formula evaluation:** Given an input text containing non-negative integers and three operator i.e. +, - and *, evaluate the given postfix formula using flex based lexical analyzer. For example given the following input:

  44 33 22 * + 1 -

  Your output should resemble:

```
44 0 0 0
 33 44 0 0
 22 33 44 0
 726 44 0 0
 770 0 0 0
 1 770 0 0
 769 0 0 0
result = 769
```

Flex Installation

Command:

- **sudo apt-get update**
- **sudo apt-get install flex**

Now the package of flex is installed to your home directory.

To check the flex version use following command.

- **flex --version**

Source Code task1.l

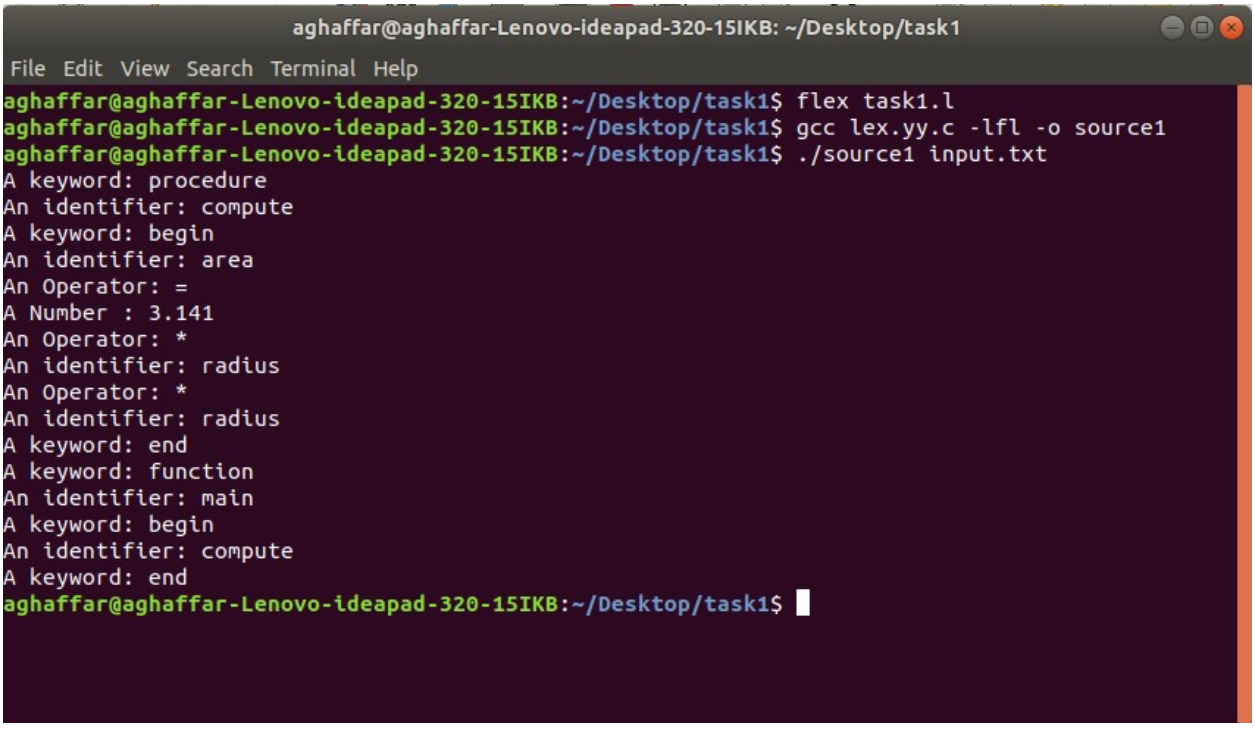
```
/*********************************
Task: 1 ~~ Code for PostFix with
~~input string::
```

```
procedure compute

begin

   area = 3.141 * radius * radius

end

function main

begin

   compute

end

*********************************/



/*Transational rules*/


%%

(("if"|("then")|("begin")|("end")|("procedure")|("function"))) {printf("A keyword: ");
ECHO; printf("\n");}


([+-/*<>=/]|[%$]) {printf("An Operator: "); ECHO; printf("\n");}


(\") {printf("A Qoutation: "); ECHO; printf("\n");}


([()]|[{}]) {printf("A Bracket: "); ECHO; printf("\n");}


("//".*) {printf("A Comment: "); ECHO; printf("\n");}
```

```
(\\n) {printf("A newline operator"); ECHO; printf("\n");}

(;) {printf("A Semicolon: "); ECHO; printf("\n");}

(!) {printf("An Exlamation Mark: "); ECHO; printf("\n");}

([\n\t" "]) {}


([+-]?([0-9]*[.])?[0-9]+) {printf("A Number : "); ECHO; printf("\n");}

([A-Z]) {printf(" Capital Alphabets "); ECHO; printf("\n");}

([a-z]+[0-9a-zA-Z]*) {printf("An identifier: "); ECHO; printf("\n");}

. {printf("Unrecognized character: "); ECHO; printf("\n");}

%%
//driver function
int main(int argc, char **argv)

{
```

```
//condition for the file input

if(argc>1)


yyin=fopen(argv[1],"r");


else


yyin=stdin;

//lexical analyser invoking function.

yylex();
```

OUTPUT



National University of Sciences and Technology (NUST)
School of Electrical Engineering and Computer Science

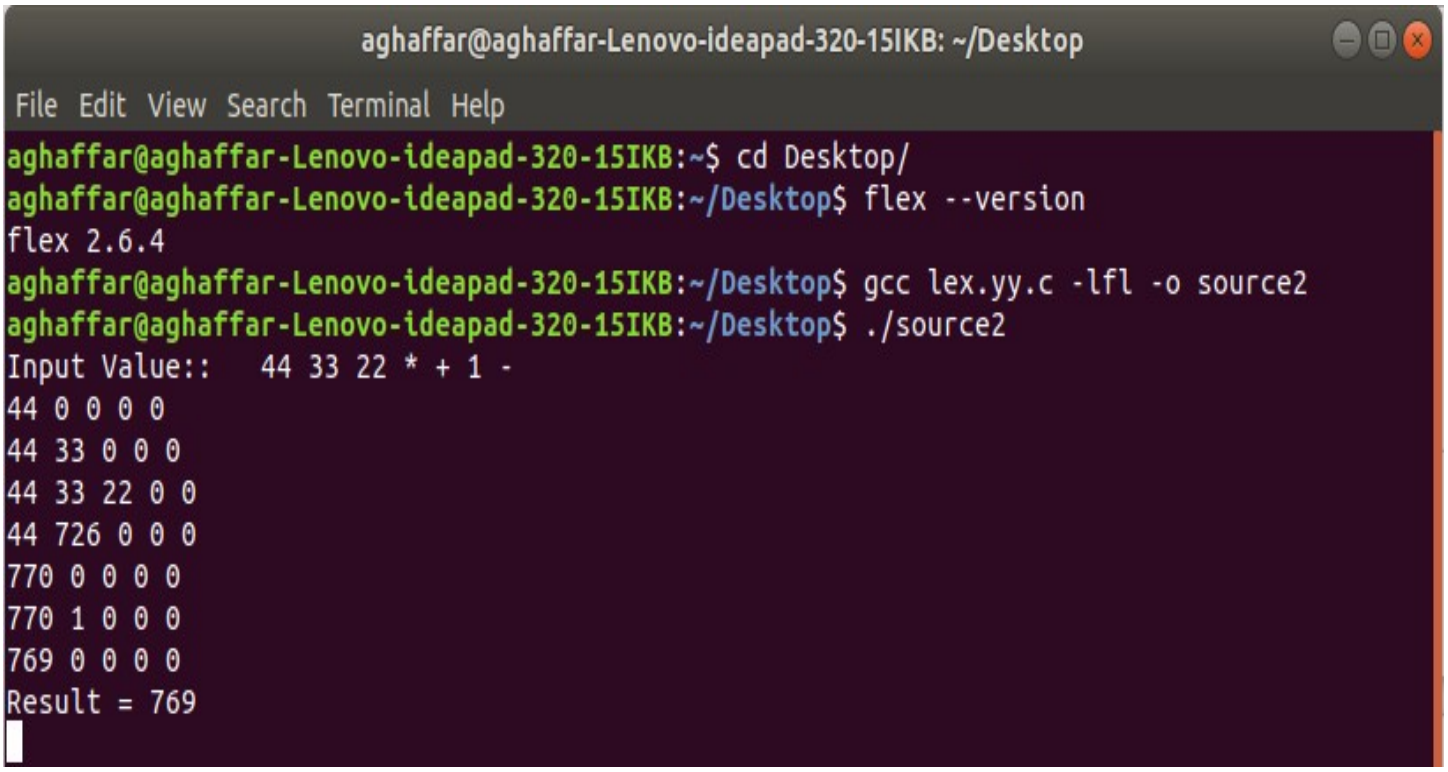
Source Code task2.l

```
/*******************************
Task: 2 ~~ Code for PostFix with
input string::  44 33 22 * + 1 -
*******************************/


%{
// #define stack_size 5
 static int indexStack, stackValue [5];
int counter1 = 0;
//function for pushing values.
 static void func_pushValues (int func_arg) {
//condition for checking stack size to be less than 5
   if (++indexStack<5) {
//insert into stack
stackValue[indexStack]= func_arg;
}
 }
//printing function.
static void print() {
```

```
//loop until less than value.
for(counter1 = 0;counter1<5;counter1++){
printf("%d ",stackValue[counter1]);
}
printf("\n");



}
//pop all the values of the stack function
 static int stackPOP (void) {
   if (indexStack>=0) {
indexStack = indexStack;
int temp = stackValue[indexStack];
stackValue[indexStack] = 0;
indexStack = indexStack -1;
return temp;

}

 }
%}
%%
```

```
[0-9]+                {func_pushValues (atoi(yytext));print();}

"+"                   {func_pushValues (stackPOP() + stackPOP());print();}

"-"                           {int right= stackPOP(); func_pushValues (stackPOP() -
right);print();}

"*"                        {int first = stackPOP();int second = stackPOP(); int product =
first*second; func_pushValues(product);print();}

"/"                           {int right= stackPOP(); func_pushValues (stackPOP() /
right);print();}

"\n"                   {printf ("Result = %d\n", stackPOP());}

[ \t\n]                ;

%%

//drive function

 int main (void) {

indexStack= -1;


yyin = stdin;

printf("Input Value:: \t");


yylex();

}
```

OUTPUT



Deliverables

You are required to upload your task (Sources & PDF document) using the link created on LMS followed by a viva.