**Compilers Laboratory**

**B. Tech. 6th Semester**

**Batch: 2017**



**Department: Computer Science and Engineering**

**Faculty of Engineering & Technology**

**Ramaiah University of Applied Sciences**

|  |  |
| --- | --- |
| Faculty | Engineering & Technology |
| Programme | B. Tech. in Computer Science and Engineering |
| Course | Compilers Laboratory |
| Year/Semester | 2017/6th Semester |
| Course Code | CSC312A |

**Ramaiah University of Applied Sciences**

Private University Established in Karnataka State by Act No. 15 of 2013

List of Experiments

**LEX PROGRAMS**

1. Program to count the number of vowels and consonants in a given string.
2. Program to find the longest word in a given string.
3. Program to count no of:
   1. +positive and –negative integers
   2. +positive and –negative fractions
4. Program to count the number of characters, words, spaces, end of lines in a given input file.
5. Program to count the no of ‘scanf’ and ‘printf’ statements in a C program. Replace them with ‘readf’ and ‘writef’ statements respectively.
6. Program to perform addition, subtraction, multiplication, division and power. Note: Without Precedence.

**YACC & LEX PROGRAMS**

1. Program to recognize a valid variable, which starts with a letter, followed by any number of letters or digits.
2. Program to test the syntax of a simple expression and evaluate an arithmetic expression involving operating +, -, \* and /
3. Program to recognize strings ‘aaab’, ‘abbb’, ‘ab’ and ‘a’ using grammar (a^nb^m, n>=0. m>=0)

# Laboratory 1

Title of the Laboratory Exercise: Program to count the number of vowels and consonants in a given string

1. Introduction and Purpose of Experiment

Students learn to use Lex program to find out vowels and consonants in a given string

.

1. Aim and Objectives

Aim

* To write a program to count the number of vowels and consonants in a given string

Objectives

At the end of this lab, the student will be able to

* Define regular expression for vowels and consonants
* Count the number of vowels and consonants

1. Experimental Procedure

Students are required to carry out the following steps:

* Algorithm
* Write the Lex program
* Compile and execute the program (steps)
* Complete the documentation for the given problem

1. **Presentation of Results**

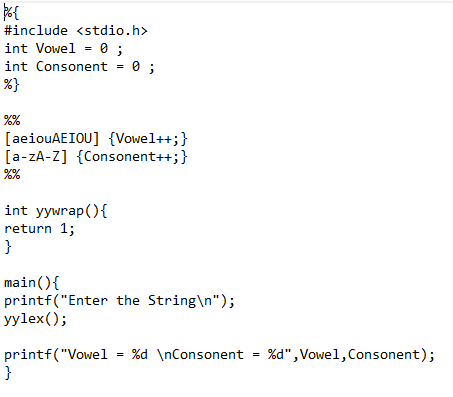
**ALGORITHM:**

1 . first open notepad and save it in the .l extension to make it a lex file.

2 . In the “DEFINITIONS” section of the lex file, we declare and initialise the required variables such as vowels, consonants.

3 . In the “TRANSITION” section of the lex file, we define the rules which has to be followed by the lexical analyser.

4 . In the final section of the lex file, we initialise the lexical analyser via the main function and in that main function, we call the yylex() function which is the lexical analyser itself.



***OUTPUT:***

1. **Analysis and Discussions**

In the above lab, the program is developed to determine or count the number of the vowels or consonants present in a given string using the Lex program.

* + Lex generates C code for a lexical analyzer, or scanner
  + Lex uses patterns that match strings in the input and converts the strings to tokens

Here, in the above lab the vowels and the number of the consonants are to be counted in the given string that is the string input by the user.

* Lex is a program that generates lexical analyzer.
* The lexical analyzer is a program that transforms an input stream into a sequence of tokens.
* It reads the input stream and produces the source code as output through implementing the lexical analyzer in the C program.

The **lex** [utility](https://www.computerhope.com/jargon/u/utility.htm) generates [C](https://www.computerhope.com/jargon/c/c.htm) programs to be used in lexical processing of [character](https://www.computerhope.com/jargon/c/charact.htm) input, and that can be used as an interface to [yacc](https://www.computerhope.com/unix/uyacc.htm). The C programs are generated from **lex** source code and conform to the [ISO](https://www.computerhope.com/jargon/i/iso.htm) C standard. Usually, the **lex** utility writes the program it generates to the file **lex.yy.c**. The state of this file is unspecified if **lex** exits with a non-zero exit status.

1. **Conclusions**

Lex program is developed to count the number of vowels and consonants in the string input by defining the regular expression for the vowel and consonants.

lex cannot be used to recognize nested structures such as parentheses. Nested structures are handled by incorporating a stack. Whenever we encounter a “(” we push it on the stack. When a “)” is encountered we match it with the top of the stack and pop the stack. However, lex only has states and transitions between states. Since it has no stack it is not well suited for parsing nested structures.

No limitation. This program counts and prints number of vowels and consonants.

By doing this lab, learnt to write a Lex program, compile and execute it. I also learnt to write regular expressions for vowels and consonants.

Lex is good at pattern matching.

|  |  |  |
| --- | --- | --- |
| ****Component**** | ****Max Marks**** | ****Marks Obtained**** |
| **Viva** | **6** |  |
| **Results** | **7** |  |
| **Documentation** | **7** |  |
| ****Total**** | ****20**** |  |

# Laboratory 2

Title of the Laboratory Exercise: Program to find the longest word in a given string.

1. Introduction and Purpose of Experiment

Students learn to use Lex program to find out the longest word in a given string.

1. Aim and Objectives

Aim

* To write a program to fine the longest word in a given string

Objectives

At the end of this lab, the student will be able to

* Define regular expression for words
* Find the longest word in a given string

1. Experimental Procedure

Students are required to carry out the following steps:

* Algorithm
* Write the Lex program
* Compile and execute the program (steps)
* Complete the documentation for the given problem

1. **Presentation of Results**

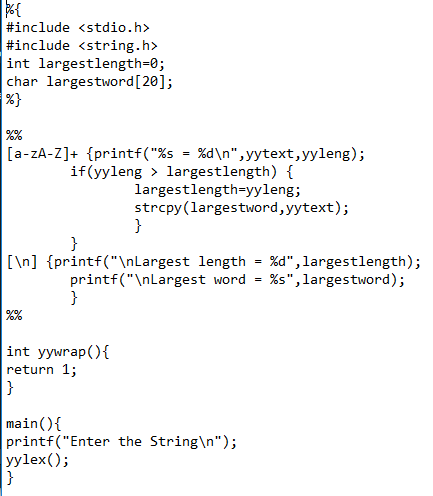
**ALGORITHM:-**

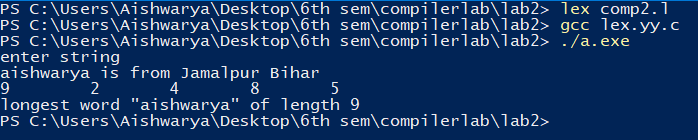
1 . first open notepad and save it in the .l extension to make it a lex file.

2 . In the “DEFINITIONS” section of the lex file, we declare and initialise the required variables such as word and word length.

3 . In the “TRANSITION” section of the lex file, we define the rules which has to be followed by the lexical analyser.

4 . In the final section of the lex file, we initialise the lexical analyser via the main function and in that main function, we call the yylex() function which is the lexical analyser itself.





1. **Analysis and Discussions**

Lex is a computer program that generates lexical analysers. Lex reads an input stream specifying the lexical analyser and outputs source code implementing the Lexar in the C programming language.

Function yywrap is called by lex when input is exhausted. Return 1 if you are done or 0 if more processing is required. Every C program requires a main function. In this case we simply call yylex that is the main entry-point for lex.

### The function of Lex is as follows:

* lexical analyzer creates a program lex.1 in the Lex language. Then Lex compiler runs the lex.1 program and produces a C program lex.yy.c.
* C compiler runs the lex.yy.c program and produces an object program a.out.
* a.exe is lexical analyzer that transforms an input stream into a sequence of tokens.

Lex is a program generator designed for lexical processing of character input streams. It accepts a high-level, problem-oriented specification for character string matching, and produces a program in a general-purpose language which recognizes regular expressions. The regular expressions are specified by the user in the source specifications given to Lex. The Lex written code recognizes these expressions in an input stream and partitions the input stream into strings matching the expressions.

1. **Conclusions**

In the above lab, Lex program is developed where the user input the string and the output obtained is the longest word in the string.

lex cannot be used to recognize nested structures such as parentheses. Nested structures are handled by incorporating a stack. Whenever we encounter a “(” we push it on the stack. When a “)” is encountered we match it with the top of the stack and pop the stack. However, lex only has states and transitions between states. Since it has no stack it is not well suited for parsing nested structures.

No limitation.

By doing this lab, learnt to write a Lex program, compile and execute it. I also learnt to write regular expressions for vowels and consonants.

Lex is good at pattern matching.

|  |  |  |
| --- | --- | --- |
| ****Component**** | ****Max Marks**** | ****Marks Obtained**** |
| **Viva** | **6** |  |
| **Results** | **7** |  |
| **Documentation** | **7** |  |
| ****Total**** | ****20**** |  |

# Laboratory 3

1. Title of the Laboratory Exercise: Program to count no of:
   1. +positive and –negative integers
   2. +positive and –negative fractions
2. Introduction and Purpose of Experiment

.

1. Aim and Objectives

Aim

Objectives

At the end of this lab, the student will be able to

1. Experimental Procedure

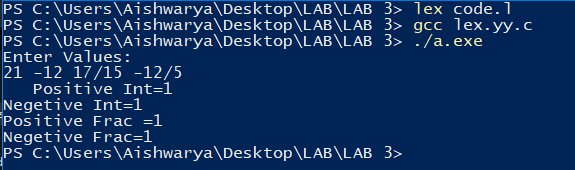
Students are required to carry out the following steps:

* Algorithm
* Write the Lex program
* Compile and execute the program (steps)
* Complete the documentation for the given problem

1. **Presentation of Results**



**OUTPUT:**



1. **Analysis and Discussions**

Lex is not a complete language, but rather a generator representing a new language feature which can be added to different programming languages. During the first phase the compiler reads the input and converts strings in the source to tokens. With regular expressions we can specify patterns to lex so it can generate code that will allow it to scan and match strings in the input. Each pattern specified in the input to lex has an associated action.

### The function of Lex is as follows:

* Firstly, lexical analyzer creates a program lex.1 in the Lex language. Then Lex compiler runs the lex.1 program and produces a C program lex.yy.c.
* Finally, C compiler runs the lex.yy.c program and produces an object program a.out.
* a.exe is lexical analyzer that transforms an input stream into a sequence of tokens.

lex cannot be used to recognize nested structures such as parentheses. Nested structures are handled by incorporating a stack.

1. **Conclusions**

In the above lab, for solving the given problem that is to count the number of +VE and the -VE integers as well as the fractions.

lex cannot be used to recognize nested structures such as parentheses. Nested structures are handled by incorporating a stack. Whenever we encounter a “(” we push it on the stack. When a “)” is encountered we match it with the top of the stack and pop the stack. However, lex only has states and transitions between states. Since it has no stack it is not well suited for parsing nested structures.

By doing this lab, I learnt to write a Lex program, compile and execute it. I also learnt to write regular expressions for vowels and consonants.

Lex is good at pattern matching.

|  |  |  |
| --- | --- | --- |
| ****Component**** | ****Max Marks**** | ****Marks Obtained**** |
| **Viva** | **6** |  |
| **Results** | **7** |  |
| **Documentation** | **7** |  |
| ****Total**** | ****20**** |  |

# Laboratory 4

Title of the Laboratory Exercise: Program to count the number of characters, words, spaces, end of lines in a given input file.

1. Introduction and Purpose of Experiment
2. Aim and Objectives

Aim

* To write a program to

At the end of this lab, the student will be able to

* Define

1. Experimental Procedure

Students are required to carry out the following steps:

* Algorithm
* Write the Lex program
* Compile and execute the program (steps)
* Complete the documentation for the given problem

1. **Presentation of Results**

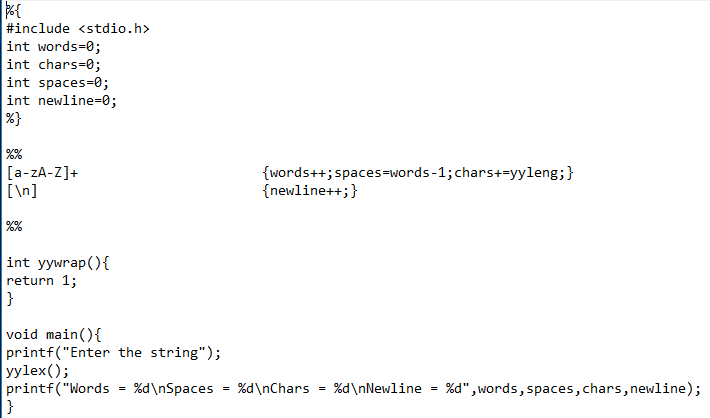
**ALGORITHMS:**

1 . First open NOTEPAD, for lex file save the notepad in .l extension.

2 . In the “DEFINITIONS” section of the lex file, we declare and initialise the required variables as shown in the figures later on.

3 . In the “TRANSITION” section of the lex file, we define the rules which has to be followed by the lexical analyser.

4 . In the final section of the lex file, we initialise the lexical analyser via the main function and in that main function, we call the yylex() function which is the lexical analyser itself.



1. **Analysis and Discussions**

A lex file program in which the user gives a string as an input to the program. The program in-turn analyses the input-string to determine the number of the words, characters, spaces and newline as stated in the problem statement. To find the number of words, we define the RULE in the ‘lex’ file and at the same time we use a function **‘yyleng’** which returns the total length of the concerned word.

### The function of Lex is as follows:

* Firstly lexical analyzer creates a program lex.1 in the Lex language. Then Lex compiler runs the lex.1 program and produces a C program lex.yy.c.
* Finally C compiler runs the lex.yy.c program and produces an object program a.out.
* a.exe is lexical analyzer that transforms an input stream into a sequence of tokens.

1. **Conclusions**

In the above lab, executed the concept of a lexical analyzer. The lexical analyzer takes string as an input from the user and then returns the no. of words, no. of characters, no. of spaces and no. of newlines in the string. Learnt how to use the 3 different sections of a ‘lex-file’.

|  |  |  |
| --- | --- | --- |
| ****Component**** | ****Max Marks**** | ****Marks Obtained**** |
| **Viva** | **6** |  |
| **Results** | **7** |  |
| **Documentation** | **7** |  |
| ****Total**** | ****20**** |  |

# Laboratory 7

Title of the Laboratory Exercise: Program to recognize a valid variable, which starts with a letter, followed by any number of letters or digits.

1. Introduction and Purpose of Experiment

Students learn to

1. Aim and Objectives

Aim

* To write a program

Objectives

At the end of this lab, the student will be able to

* Define

1. Experimental Procedure

Students are required to carry out the following steps:

* Algorithm
* Write the Lex program
* Write yacc program
* Compile and execute the program (steps)
* Complete the documentation for the given problem

1. **Presentation of Results**

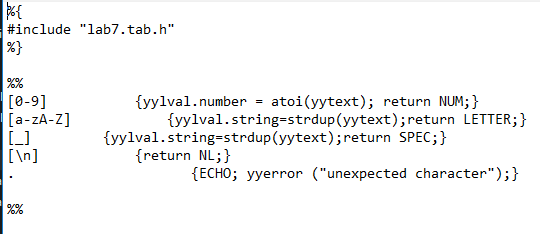
**ALGORITHM:**

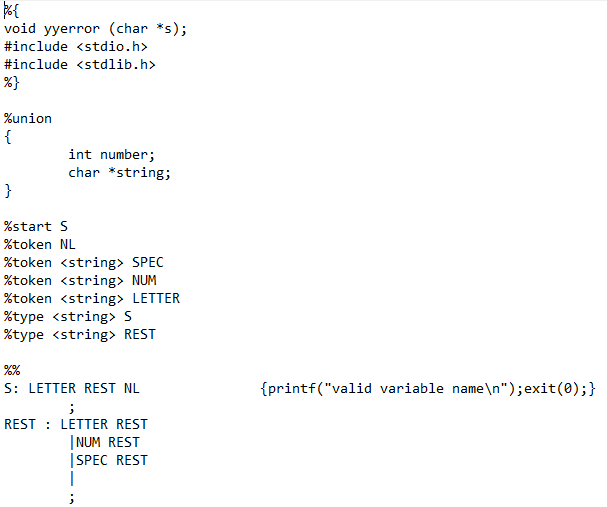
1 . first open NOTEPAD and make two files that is .l extension for lex file anf nest .y extension for yacc file..

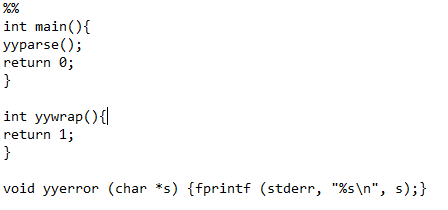
1. In the definition section of the ‘lex’ file we include the ‘yacc’ file, so that the lex file could give the information that the ‘yacc’ file would need while parsing the input string.
2. In the lex file, in the ‘RULES’ section, we define the token that would be returned when a particular lexeme would be encountered in the string given as input by the user.

For the precedence of the operators in the ascending order from top to bottom.

1. In the RULES SECTION, we define the grammar rules which would be used for matching the input taken from the user.
2. In the LAST SECTION, we run the yyparse() function by calling it in the main() function. The user is asked for entering the expression.
3. After main function is completed, yywrap() function is called which returns a value 1.
4. At last, the function yyerror() is defined, to display the error using fprintf method.







1. **Analysis and Discussions**

YACC (Yet Another Compiler Compiler) is a program designed to compile a LALR(1) grammar and to produce the source code of the syntactic analyzer of the language produced by this grammar.

* + Yacc generates C code for syntax analyzer, or parser.
  + Yacc uses grammar rules that allow it to analyze tokens from Lex and create a syntax tree.
* Uses bottom up Shift/Reduce parsing
  + Get a token
  + Push onto stack
  + Can it reduced
    - If yes: Reduce using a rule
    - If no: Get another token
* Yacc cannot look ahead more than one token

1. **Conclusions**

In the above lab, Yacc program is developed to recognize a variable which starts with letter followed by any number of letter or digits or underscore.

|  |  |  |
| --- | --- | --- |
| ****Component**** | ****Max Marks**** | ****Marks Obtained**** |
| **Viva** | **6** |  |
| **Results** | **7** |  |
| **Documentation** | **7** |  |
| ****Total**** | ****20**** |  |

# Laboratory 8

Title of the Laboratory Exercise: Program to test the syntax of a simple expression and evaluate an arithmetic expression involving operating +, -, \* and /

1. Introduction and Purpose of Experiment

Students learn to

1. Aim and Objectives

Aim

* To write a program to

Objectives

At the end of this lab, the student will be able to

* Define

1. Experimental Procedure

Students are required to carry out the following steps:

* Algorithm
* Write the Lex program
* Write yacc program
* Compile and execute the program (steps)
* Complete the documentation for the given problem

1. **Presentation of Results**

**ALGORITHM:**

* + - 1. First open notepad and save a .l extension for lex file and .y extension for yacc file.s
      2. In the DEFINITION SECTION of the lex file, we include the yacc header file which would be processed to a C level program.

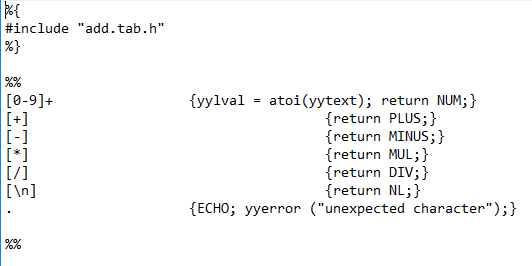
3.In the RULES SECTION we define the tokens that will be returned when a particular lexeme is encountered while scanning the input entered by the user.

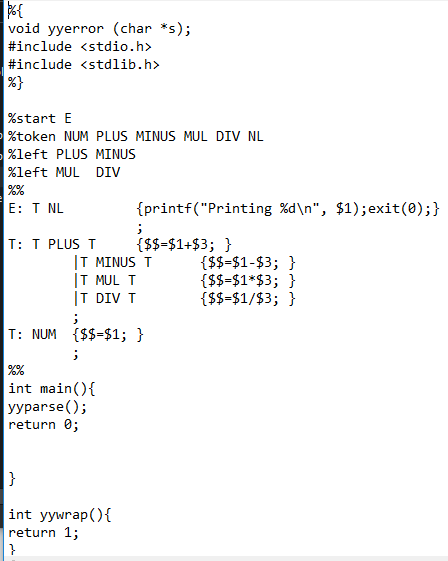
For yacc (parser)-

1 . In the DEFINITION SECTION of the ‘yacc’ file, we include the standard i/o and standard library file. After that, we declare the tokens that can be expected by the yacc file.

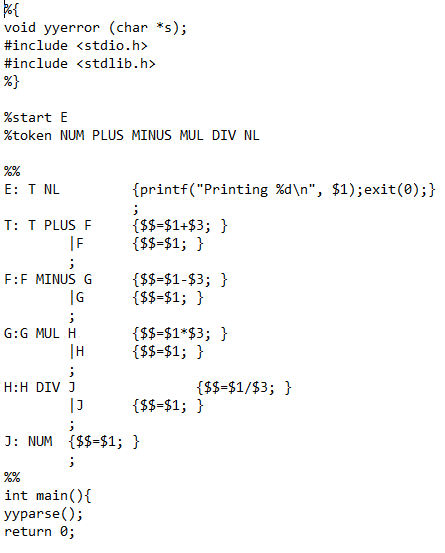
2 . In the RULES SECTION of the ‘yacc’ file, we write the grammar rules which would be used by the yacc file to check the input given by the user.

3 . In the last section of the ‘yacc’ file we run the yyparse() function by calling it in the main() function, and after that yywrap() function is called.





Unambiguous:



1. **Analysis and Discussions**

YACC (Yet Another Compiler Compiler) is a program designed to compile a LALR(1) grammar and to produce the source code of the syntactic analyzer of the language produced by this grammar.

* + Yacc generates C code for syntax analyzer, or parser.
  + Yacc uses grammar rules that allow it to analyze tokens from Lex and create a syntax tree.
* Uses bottom up Shift/Reduce parsing
  + Get a token
  + Push onto stack
  + Can it reduced
    - If yes: Reduce using a rule
    - If no: Get another token
* Yacc cannot look ahead more than one token

1. **Conclusions**

To rectify the ambiguity of the defined grammar, we describe the precedence and the associativity of the operators that are used in the grammar. The operator with higher precedence are described in the lower parts of the list, i.e., the precedence of the operators increases from top to bottom.

|  |  |  |
| --- | --- | --- |
| ****Component**** | ****Max Marks**** | ****Marks Obtained**** |
| **Viva** | **6** |  |
| **Results** | **7** |  |
| **Documentation** | **7** |  |
| ****Total**** | ****20**** |  |