SSBT's College of Engineering & Technology, Bambhori, Jalgaon

Department of Computer Engineering

Name: _	Date of Performance: _ _ /_ _/20_ _
Class: B.E. Computer	Date of Completion: _ _ /_ _/20_ _
Division :	Grade:
Batch:	
Roll No:	Sign. of Teacher with Date:
Subject: Compiler Design Lab	

Experiment No. 2

Aim: To Implement a lexical analyzer of identification of numbers

1. Objective: Implement a lexical analyzer of identification of numbers
(Numbers can be binary, octal, decimal, hexadecimal, float or exponential)

2. Background:

A binary number is a number expressed in the binary numeral system or base-2 numeral system which represents numeric values using two different symbols: typically 0 (zero) and 1 (one). The base-2 system is a positional notation with a radix of 2. Because of its straightforward implementation in digital electronic circuitry using logic gates, the binary system is used internally by almost all modern computers and computer-based devices.

The octal numeral system, or oct for short, is the base-8 number system, and uses the digits 0 to 7. Octal numerals can be made from binary numerals by grouping consecutive binary digits into groups of three (starting from the right). For example, the binary representation for decimal 74 is 1001010, which can be grouped into (00)1 001 010 – so the octal representation is 112

The decimal numeral system (also called base 10 or occasionally denary) has ten as its base. It is the numerical base most widely used by modern civilizations.[1][2]

Decimal notation often refers to a base 10 positional notation such as the Hindu-Arabic numeral system or rod calculus;[3]however, it can also be used more generally to refer to non-positional systems such as Roman or Chinese numerals which are also based on powers of ten.

Hexadecimal (also base 16, or hex) is a positional numeral system with a radix, or base, of 16. It uses sixteen distinct symbols, most often the symbols 0–9 to represent values zero to nine, and A, B, C, D, E, F (or alternatively a,b, c, d, e, f) to represent values ten to fifteen. Hexadecimal numerals are widely used by computer system designers and programmers. Several different notations are used to represent hexadecimal constants in computing languages; the prefix "0x" is widespread due to its use in Unix and C (and related operating systems and languages). Alternatively, some authors denote hexadecimal values using a suffix or subscript. For example, one could write 0x2AF3 or $2AF3_{16}$, depending on the choice of notation.

Floating point is the formulaic representation that approximates a real number so as to support a trade-off between range and precision. A number is, in general, represented approximately to a fixed number of significant digits (the significand) and scaled using an exponent; the base for the scaling is normally two, ten, or sixteen. A number that can be represented exactly is of the following form:

$$\text{significand} \times \text{base}^{\text{exponent}},$$

3. Pre-lab Task:

First, we've already shown regular expression for a "digit":

[O-9]

We can use this to build a regular expression for an integer:

We require at least one digit. This would have allowed no digits at all: Let's add an optional unary minus: We can then expand this to allow decimal numbers. First we will specify a decimal number (for the moment we insist that the last character always be a digit):

[0-9]*\. [0-9]+

Notice the "\" before the period to make it a literal period rather than a wild card character. This pattern matches "0.0", ***"4.5",*** or ".31415". But it won't match *"0"* or "2". We'd like to combine our definitions to match them as well. Leaving out our unary minus, we could use: We use the grouping symbols *"0*t*"*o specify what the regular expressions are for the " I " operation. Now let's add the unary minus:

-?(([0-9]+) 1 ([0-9]*\.[0-9]+))

We can expand this further by allowing a float-style exponent to be specified as well. First, let's write a regular expression for an exponent: This matches an upper- or lowercase letter E, then an optional plus or minus sign, then a string of digits. For instance, this will match "e12" or "E-3". We can then use this expression to build our final expression, one that specifies a real number:

-?((to-91+) I (10-91*\. 10-9]+) ([a - +] ? t o - 9 1 +) ?)

Our expression makes the exponent part optional. Let's write a real lexer that uses this expression. It examines the input and tells us each time it matches a number according to our regular expression.

4. In-lab Task:

```
Template :
#include<math.h>
    FILE *fp;
%}
Binary [0-1]+
…
…..
…..
```

```
%%
{Binary} {printf("this is an binary number : %s", yytext);}
…..
…..
%%

int main(int argc,char *argv[])
{
   fp=fopen(argv[1],"r");

   if(fp!=NULL)
   {
      yyin=fp;
      yylex();
   }
   return(0);
}
```

5. Post-lab Task:

Outcomes:

....
....
....
....

Questions:

Q1. Write regular expression for

a. Binary number

b. Octal number

c. Decimal number

d. Hexadecimal number

e. Floating point number

f. Exponential number