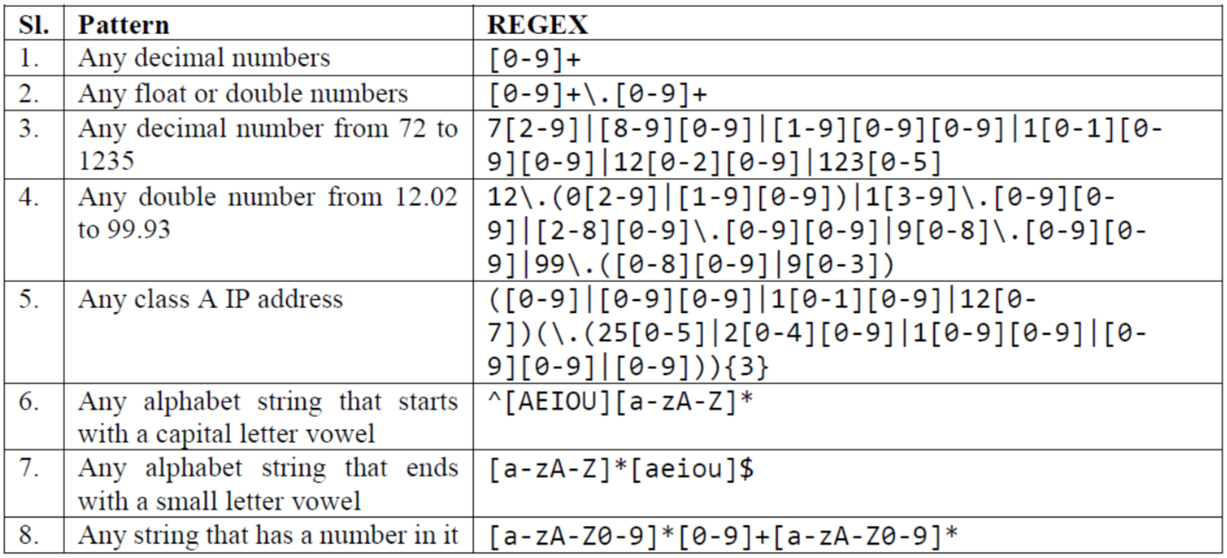
Lab No. 02 – Advanced Lexical Analyzer

Creating patterns using regular expressions:

Let’s dig deeper into building regular expressions. Here are some samples and their corresponding regex.



<table>
<tr><th>Sl.</th><th>Pattern</th><th>REGEX</th></tr>
<tr><td>1.</td><td>Any decimal numbers</td><td><code>[0-9]+</code></td></tr>
<tr><td>2.</td><td>Any float or double numbers</td><td><code>[0-9]+\.[0-9]+</code></td></tr>
<tr><td>3.</td><td>Any decimal number from 72 to 1235</td><td><code>7[2-9]|[8-9][0-9]|[1-9][0-9][0-9]|1[0-1][0-9][0-9]|12[0-2][0-9]|123[0-5]</code></td></tr>
<tr><td>4.</td><td>Any double number from 12.02 to 99.93</td><td><code>12\.(0[2-9]|[1-9][0-9])|1[3-9]\.[0-9][0-9]|[2-8][0-9]\.[0-9][0-9]|9[0-8]\.[0-9][0-9]|99\.([0-8][0-9]|9[0-3])</code></td></tr>
<tr><td>5.</td><td>Any class A IP address</td><td><code>([0-9]|[0-9][0-9]|1[0-1][0-9]|12[0-7])(\.(25[0-5]|2[0-4][0-9]|1[0-9][0-9]|[0-9][0-9]|[0-9])){3}</code></td></tr>
<tr><td>6.</td><td>Any alphabet string that starts with a capital letter vowel</td><td><code>^[AEIOU][a-zA-Z]*</code></td></tr>
<tr><td>7.</td><td>Any alphabet string that ends with a small letter vowel</td><td><code>[a-zA-Z]*[aeiou]$</code></td></tr>
<tr><td>8.</td><td>Any string that has a number in it</td><td><code>[a-zA-Z0-9]*[0-9]+[a-zA-Z0-9]*</code></td></tr>
</table>

Explanation of the patterns:

- **Pattern 1 - Decimal numbers:** As all decimal numbers are composed of the character 0-9 so a simple pattern of repeating 0-9 for 1 or infinite times is enough to recognize any decimal numbers.
- **Pattern 2 - Double/Float Numbers:** Double numbers vary from decimal numbers by a (.) between the numbers. So on both sides we have the same pattern except a (.) in the middle.
- **Pattern 3 - Decimal number range:** We need to treat the numbers as characters so we need to break down the numbers into recognizable ranges. See that, we have divided the numbers from 72 – 79, 80 – 99, 100 – 999, 1000 – 1199, 1200 – 1229 and 1230 – 1235, and then just created REGEX for each of the ranges and used an OR operation among them.
- **Pattern 4 - Double numbers range:** Same principle as decimal numbered ranges except we need an extra (.) between the numbers.
- **Pattern 5 - IP addresses:** As we are using the class A IP address i.e., 0.0.0.0 – 127.255.255.255, so the 4th Quadrant has the pattern 0-127, and the rest of the three quadrants have the same pattern of 0-255. So, we repeated the pattern for 3 quadrants using the {value} operator.
- **Pattern 6 - Vowel letter starting:** To match at the start of a string we used a (^) sign. The rest are only for all possible alphabets.

- **Pattern 7 - Vowel letter ending:** Same principle as above except for matching at the end of the string we used a ($) operator.
- **Pattern 8 - Number within a string:** As the number can be either at the front or middle or end of a string we enclosed it with possible pattern of a string and repeated the string patterns for 0 or infinite times.

Complete Code:

Now, let’s combine the given patterns into a complete program.

```
%{
//For today we do not need any variables
%}
%%

[0-9]+ {printf("%s - Found a decimal number.\n",yytext);}
[0-9]+\.[0-9]+ {printf("%s - Found a double number.\n",yytext);}
123[0-5]|12[0-2][0-9]|1[0-1][0-9][0-9]|[1-9][0-9][0-9]|[8-9][0-9]|7[2-9] {printf("%s - Found a decimal number within 72 - 1235.\n",yytext);}
12\.(0[2-9]|[1-9][0-9])|1[3-9]\.[0-9][0-9]|[2-8][0-9]\.[0-9][0-9]|9[0-8]\.[0-9][0-9]|99\.([0-8][0-9]|9[0-3]) {printf("%s - Found a double number within 12.02 - 99.93.\n",yytext);}
([0-9]|[0-9][0-9]|1[0-1][0-9]|12[0-7])(\.(25[0-5]|2[0-4][0-9]|1[0-9][0-9]|[0-9][0-9]|[0-9])){3} {printf("%s - Found a class A IP address.\n",yytext);}

^[AEIOU][a-zA-Z]* {printf("%s - Found an alphabet string that starts with a capital letter vowel.\n",yytext);}
[a-zA-Z]*[aeiou]$ {printf("%s - Found an alphabet string that ends with a small letter vowel.\n",yytext);}
[a-zA-Z0-9]*[0-9]+[a-zA-Z0-9]* {printf("%s - Found a string that has a number in it.\n",yytext);}
%%

int main(){
FILE *file;
file = fopen("code.c", "r") ;
	if (!file) {
	printf("couldnot open file");
	exit (1);
	}
	else {
	yyin = file;
	}
yylex();
}
```

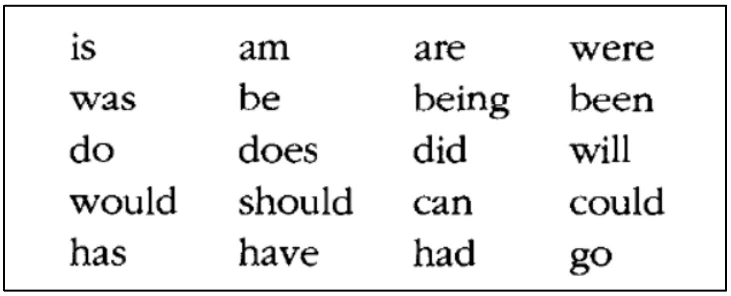
Tasks for LAB 02:

1. Create regular expressions for accepting the following strings:
   a. Any double number from -12.02 to +5699.93
   b. Any alphanumeric strings that either starts with or ends with a digit.
   c. Valid email address
   d. Valid student IDs from batches 61 – 70 of CSE department.
   e. Valid C identifier

f. Valid C include statements

2. Find the character which has the highest frequency in a given paragraph that is written in code1.c file

3. Create a lexical analyzer that can identify the following verbs and their frequency in a paragraph written in the English language. The paragraph is written in code1.c file



is	am	are	were
was	be	being	been
do	does	did	will
would	should	can	could
has	have	had	go

4. Create a lex program that will count the number of variables, variable types, operators and digits separately.