# EA872 Laboratório de Programação de Software Básico
# Atividade 3


**Vinícius Esperança Mantovani** **RA 247395**

Entrega (limite): 16/08/2023, durante a aula

**Exercício 1)**

a) O analisador léxico em questão funciona como um contador de caracteres, palavras e linhas de um arquivo de input. Essa é sua finalidade.

b) A seguir, está um exemplo de execução do analisador:

```
[pininsu@fedora arquivos_de_apoio_ativ_3]$ ./ex1
olá, professor, tudo bem com o senhor?
Este é um teste de funcionamento do
analisador!c=88 p=15    l=2
```

c) A variável yyleng armazena o comprimento da sequência de caracteres (string) capturada pelo analisador em uma determinada regra.

d) O padrão exibido, “ [^ \t\n]+ ”, é uma expressão regular para uma sequência de caracteres diferentes de “\t” e “\n”, ou seja, é uma sequência de caracteres sem espaçamento ou quebras de linha entre si.

e) Caso removessemos o “\n”, teriamos um número menor de palavras contadas do que realmente existe, uma vez que, ao final de uma linha, não existe um espaço, mas sim uma quebra de linha. Logo, caso não houvesse “\n”, a cada linha terminada, perder-se-ia uma palavra na contagem, ou seja, para um texto com 20 palavras, ter-se-ia uma resposta de 19 na contagem (variável).

**Exercício 2)**

a) A linha #include “ex2.tab.h” é responsável por importar o arquivo “ex2.tab.h” para o arquivo “ex2.l”, ou seja, importar a tabela de símbolos para o analisador léxico.

b) A variável “yylval” armazena o valor numérico associado ao token capturado pelo analisador léxico. Enquanto que, “yytext” armazena a string capturada pelo analisador léxico.

c) Os parâmetros $1 e $2 indicam os dois primeiros elementos ao lado direito de uma produção da gramática exposta no “.y”. No caso de “ex2.y”, são usados na impressão

dos valores dos seus respectivos parâmetros referenciados. Além disso, na produção INTEIRO, o $$ passa a assumir o valor de $1 e, vale destacar que $$ é uma pseudo-variável usada para guardar o valor de uma determinada produção e, no caso de ex2.y, é responsável por armazenar o inteiro capturado no momento atual pelo analisador sintático. Por fim, notemos que, na existência de um segundo parâmetro, a impressão destoa daquela feita para o primeiro parâmetro passado pelo input, da seguinte forma: Na passagem do segundo parâmetro, temos "...(produção composta)…", enquanto que na passagem do primeiro, temos "...(produção simples)...".

d) Inicialmente, o valor digitado deve ser algum número de inteiros maior que um, por exemplo, dois inteiros. Desse modo, teremos a captura de dois "INTEIRO", conforme "INTEIRO INTEIRO", o primeiro inteiro (da esquerda) caminhará no seguinte sentido "INTEIRO" -> "linha" -> "linhas". Sendo assim, em "INTEIRO", temos a impressão de seu valor, então, convertendo-se o elemento em "linha", saimos da produção "linha" e entramos na produção "linhas", na opção "linha", nesse ponto é impresso novamente seu valor, mas com a mensagem diferente, expondo "...(produção simples)...". Assim, o primeiro "INTEIRO" é analisado com sucesso e satisfaz a gramática. Continuando, o segundo "INTEIRO" é analisado da mesma forma, mas, como temos o elemento "INTEIRO" anterior, temos na produção "linhas" a opção "linhas linha" a usar, logo, o valor do segundo "INTEIRO" é impresso conforme buscamos, acompanhado pela mensagem "...(produção composta)...".
e) Caso não houvesse o "\n" em "return(\n)", o analisador sintático teria problemas ao interpretar a sequência de tokens, uma vez que, apesar do único token definido ser o "INTEIRO", ocorre que, na produção de "linha", temos "INTEIRO '\n'". Desse modo, caso não houvesse esse retorno de "\n", esse "\n" da produção não haveria e, a sentença não seria reconhecida como parte da gramática definida pelo ".y" e ocorreria um erro de sintaxe.

**Exercício 3)**

a) Os padrões que casam com [0-9]*\.[0-9]*, são sequências de um número qualquer de caracteres numéricos sucedidos por um "." que é, por sua vez, sucedido por outra sequência de um número arbitrário de caracteres numéricos. O que se entende por um ponto flutuante (float point), um número fracionário.
b) O problema da expressão é o uso de "*", que impõe um número arbitrário de algarismos, incluindo a inexistência de algarismos. Isso implica na possibilidade de sequências como ".81", "93." ou mesmo "." serem capturadas pela expressão regular, sendo, portanto, tratadas pelo analisador como um fracionário, apesar de não serem. Sendo assim, o modo mais simples de se corrigir isso seria trocar "*" por "+", que casa com um número arbitrário de algarismos, excetuando-se a inexistência de algarismos, ou seja, não poderiam, no caso do uso do "+", haver casos em que não há casas decimais ou mesmo casas antes do ponto, havendo ao menos um algarismo em cada lado do ponto.
c) Para tornar o analisador compatível com números hexadecimais, bastou que se adicionasse a linha "hex [0][Xx][0-9]+" na seção de definições do ".l" e, as linhas:

Unset

```
{hex}   {
           sscanf(yytext, "%x", &yylval.valor_inteiro);
           return INTEIRO;
        }
```

na seção de regras do “.l” também. Isso porque, tendo definido uma expressão regular para um hexadecimal, basta que se utilize o “sscanf” para guardá-lo como inteiro na base hexadecimal na variável valor_inteiro que é um token no “.y”. Assim, retornando o “INTEIRO”, trata-se, no analisador sintático, do caso do hexadecimal do mesmo modo que se trata de um decimal, como um “int”.