

Atividade em aula 04

Vinícius de Oliveira Peixoto Rodrigues (245294)

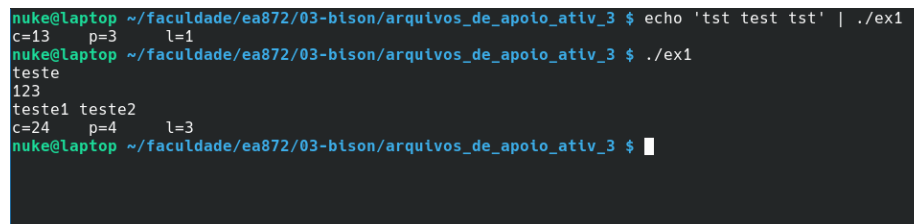
Agosto de 2022

Exercício 01

Item (a)

A especificação constrói um lexer que conta o número de palavras (**p**), linhas (**l**) e caracteres (**c**) do input.

Item (b)



```
nuke@laptop ~/faculdade/ea872/03-bison/arquivos_de_apolo_ativ_3 $ echo 'tst test tst' | ./ex1
c=13  p=3  l=1
nuke@laptop ~/faculdade/ea872/03-bison/arquivos_de_apolo_ativ_3 $ ./ex1
teste
123
teste1 teste2
c=24  p=4  l=3
nuke@laptop ~/faculdade/ea872/03-bison/arquivos_de_apolo_ativ_3 $
```

Figura 1: Exemplo de funcionamento do lexer **ex1**

Item (c)

Ela guarda o comprimento da string na qual o **flex** deu match de acordo com alguma das regras definidas. Está sendo usada para contar o número de caracteres (incluindo espaços em branco, tabs, newlines e EOF).

Item (d)

Serve para dar match em caracteres que não são um whitespace, um `"\n"` (newline) ou um `"\t"` (tab).

Item (e)

Porque ela permite levar em conta os caracteres do grupo do item (c) na contagem de caracteres total do input.

Exercício 2

Item (a)

Ela inclui o header gerado automaticamente pelo bison (`ex2.tab.h`) que contém declarações de variáveis e funções do bison que o lexer faz uso (em particular o `yylval`).

Item (b)

- `yytext`: é onde o lexer guarda os tokens (como strings) que são extraídos da entrada
- `yylval`: é a variável através da qual o lexer informa o valor **semântico** de um token para o parser (nesse caso o valor numérico do número hexadecimal lido)

Item (c)

- `$$`: serve para guardar o resultado da resolução de uma regra
- `$1`: serve para se referir ao valor do elemento de posição 1 em uma regra
- `$2`: serve para se referir ao valor do elemento de posição 2 em uma regra

Item (d)

O lexer passa o valor do número hexadecimal lido para a global `yylval`, que em seguida é usado para resolver o `linha` na regra `linhas: linhas linha`, e só então ele é impresso.

Item (e)

É necessário porque a regra de resolução `linha : INTEIRO '\n'` exige o `newline` após o token. Se a `linha` não estivesse presente, o parser não resolveria a regra `linha` e os números não seriam parseados, gerando erro de sintaxe.

Exercício 3

Item (a)

Uma sequência de dígitos com tamanho maior ou igual a 0, seguida por um ponto, seguida por uma sequência de dígitos de tamanho maior ou igual a 0.

Item (b)

A sequência que contém só um ponto final, ".", é aceita como um número float válido. Para corrigir isso, basta colocar a restrição de que é necessário ter dígitos de pelo menos um dos dois lados:

```
frac    [0-9]*([0-9]\.?\.[0-9])[0-9]*
```

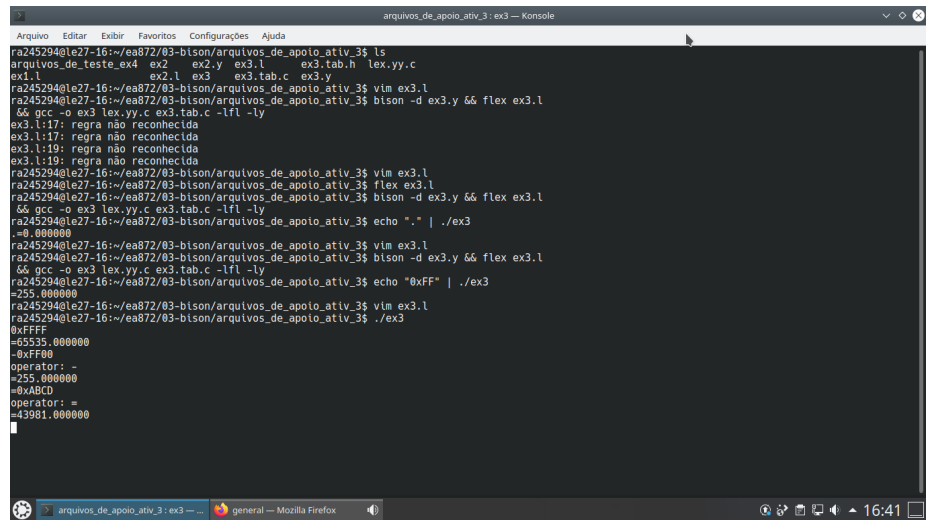
Item (c)

Basta criar uma nova definição:

```
hex      0x[0-9a-zA-Z]+
```

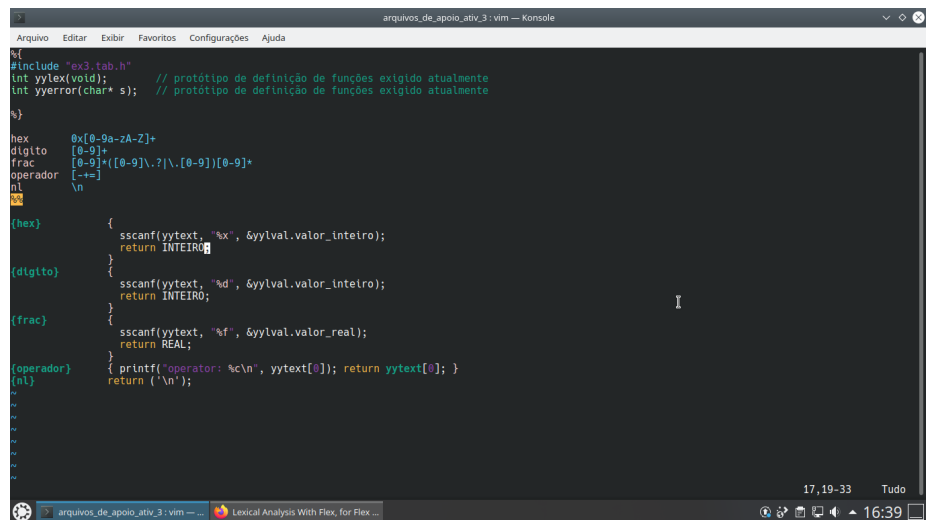
e então criar uma regra do lexer:

```
{hex}    {  
            sscanf(yytext, "%x", &yy1val.valor_inteiro);  
            return INTEIRO;  
        }
```



```
ra245294@le27-16:~/ea872/83-bison/arquivos_de_apoio_ativ_3$ ls
arquivos_de_teste_ex4  ex2  ex2.y  ex3.l  ex3.tab.h  lex.yy.c
ex1.l  ex2.l  ex3  ex3.tab.c  ex3.y
ra245294@le27-16:~/ea872/83-bison/arquivos_de_apoio_ativ_3$ vim ex3.l
ra245294@le27-16:~/ea872/83-bison/arquivos_de_apoio_ativ_3$ bison -d ex3.y && flex ex3.l
&& gcc -o ex3 lex.yy.c ex3.tab.c -lfl -ly
ex3.l:17: regra não reconhecida
ex3.l:17: regra não reconhecida
ex3.l:19: regra não reconhecida
ex3.l:19: regra não reconhecida
ra245294@le27-16:~/ea872/83-bison/arquivos_de_apoio_ativ_3$ vim ex3.l
ra245294@le27-16:~/ea872/83-bison/arquivos_de_apoio_ativ_3$ flex ex3.l
ra245294@le27-16:~/ea872/83-bison/arquivos_de_apoio_ativ_3$ bison -d ex3.y && flex ex3.l
&& gcc -o ex3 lex.yy.c ex3.tab.c -lfl -ly
ra245294@le27-16:~/ea872/83-bison/arquivos_de_apoio_ativ_3$ echo "." | ./ex3
.=0.000000
ra245294@le27-16:~/ea872/83-bison/arquivos_de_apoio_ativ_3$ vim ex3.l
ra245294@le27-16:~/ea872/83-bison/arquivos_de_apoio_ativ_3$ bison -d ex3.y && flex ex3.l
&& gcc -o ex3 lex.yy.c ex3.tab.c -lfl -ly
ra245294@le27-16:~/ea872/83-bison/arquivos_de_apoio_ativ_3$ echo "0xFF" | ./ex3
=255.000000
ra245294@le27-16:~/ea872/83-bison/arquivos_de_apoio_ativ_3$ vim ex3.l
ra245294@le27-16:~/ea872/83-bison/arquivos_de_apoio_ativ_3$ ./ex3
0xFFFF
=65535.000000
=0xFFFF
operator: =
=255.000000
=0xABCD
operator: =
=43901.000000
```

Figura 2: Exemplo de funcionamento do lexer `ex3`



```
%{
#include "ex3.tab.h"
int yylex(void); // protótipo de definição de funções exigido atualmente
int yyerror(char* s); // protótipo de definição de funções exigido atualmente
}%

hex      0x[0-9a-zA-Z]+
dglto    [0-9]+
frac     [0-9]*([0-9]\.[0-9]*|\.0-9)*
operator [-+*=]
nl       \n

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

{dglto}
{
    sscanf(yytext, "%d", &yyval.valor_inteiro);
    return INTEIRO;
}

{frac}
{
    sscanf(yytext, "%f", &yyval.valor_real);
    return REAL;
}

{operator}
{
    printf("operator: %c\n", yytext[0]); return yytext[0];
}

{nl}
{
    return ('\n');
}
```

Figura 3: Exemplo de funcionamento do lexer `ex3`