Instituto Tecnológico de Aeronáutica

Laboratório de CES-41

Laboratório I:
Exercícios básicos de programação em *Flex*

Gianluigi Dal Toso

Professor:
Fábio Carneiro Mokarzel

03 de Maio de 2020

Sobre os códigos e arquivos de dados

Anexos à este relatório está um arquivo comprimido em formato `.zip` contendo todos os códigos-fonte desenvolvidos para este laboratório, bem como o arquivo de dados utilizado para testar a implementação de cada tarefa.

Tarefa 1

Escrever um programa em *Flex* reconhecedor de cadeias sobre o alfabeto {0, 1}, tais que o número de dígitos 0 seja par ou o número de dígitos 1 seja par.

Para a resolução desta tarefa, utilizou-se de duas expressões regulares distintas que resultam na aceitação da cadeia, o que é equivalente à operação lógica "ou". Dessa forma, pela expressão `par0` são aceitas as cadeias do alfabeto com número par de dígitos 0 e pela expressão `par1` são aceitas as cadeias do alfabeto com número par de dígitos 1. O Quadro 2 contém a implementação da tarefa. Tomou-se a liberdade de verificar ainda quais cadeias inválidas pertencem e quais não pertencem ao alfabeto e mostrar essa informação na tela. Isto também será realizado nas demais tarefas.

Além disso, também tomei a liberdade para criar uma regra especial para cadeias de caracteres '-'. Quando uma cadeia com qualquer número deste caractere for encontrada, imprime-se uma linha na tela. Este artifício foi inserido somente para deixar o *output* mais didático. Os arquivos de entrada então vai ser construídos da seguinte maneira: Inicialmente terão cadeias válidas, em seguida uma cadeia de caracteres '-' para servir como separador, cadeias inválidas, outro separados e então cadeias que não pertencem ao alfabeto. O Quadro 1 contém um exemplo de arquivo de entrada para a Tarefa 1.

Quadro 1: Exemplo de arquivo de entrada

```
110
1111110
00011
----
10
01
----
211111
asdasj
001999
```

De fato, quem irá dizer se as cadeias são de fato válidas ou inválidas é a execução do programa implementado. Esse artifício utilizei somente para que eu pudesse validar com mais facilidade o programa, ao construir os arquivos de entrada de forma a facilitar a minha comparação com a impressão na saída. Este artifício será implementado nas demais Tarefas também.

Quadro 2: `tarefa1/tarefa1.l`

```
%{
#define ACEITA      1
#define REJEITA     2
#define FORA        3
%}
delim       [ \t\n\r]
ws          {delim}+
par0        1*(1*01*0)*1*
par1        0*(0*10*1)*0*
alfabeto    [01]*
separador   \-+
```

```
string      [^ \t\n\r]+

%%
{ws}        {;}
{par0}      {return ACEITA;}
{par1}      {return ACEITA;}
{alfabeto}  {return REJEITA;}
{separador} {printf("--------------------------------------\n");}
{string}    {return FORA;}
%%

int main(int argc, char* argv[])
{
    int i;

    while(i = yylex())
    {
        switch(i)
        {
            case ACEITA:
                printf("%-30s: Aceita \n", yytext);
                break;
            case REJEITA:
                printf("%-30s: Rejeita \n", yytext);
                break;
            case FORA:
                printf("%-30s: Rejeita (fora do alfabeto) \n", yytext);
                break;
        }
    }
    return 0;
}
```

O Quadro 3 contém a saída obtida da execução da Tarefa 1.

Quadro 3: `Resultado da Tarefa 1`

```
0                              : Aceita
1                              : Aceita
00                             : Aceita
11                             : Aceita
000                            : Aceita
111                            : Aceita
0101                           : Aceita
0011                           : Aceita
00110011001100110011001100110011  : Aceita
01010101010101010101010101010101  : Aceita
001                            : Aceita
110                            : Aceita
1111110                        : Aceita
00011                          : Aceita
---------------------------------------
10                             : Rejeita
01                             : Rejeita
0111                           : Rejeita
1011                           : Rejeita
0010                           : Rejeita
000111                         : Rejeita
101010                         : Rejeita
```

```
010110                          : Rejeita
--------------------------------------
211111                          : Rejeita (fora do alfabeto)
212121                          : Rejeita (fora do alfabeto)
asdasj                          : Rejeita (fora do alfabeto)
asd.mhasd1                      : Rejeita (fora do alfabeto)
00a11kwd11j                     : Rejeita (fora do alfabeto)
2213124124                      : Rejeita (fora do alfabeto)
001999                          : Rejeita (fora do alfabeto)
```

Tarefa 2

Escrever um programa em *Flex* reconhecedor de cadeias sobre o alfabeto {0, 1}, tais que o número de dígitos 0 seja ímpar e o número de dígitos 1 seja ímpar.

Para resolver este exercício utilizei-me do fato de as cadeias aceitas agora serem a negação da condição de aceitação da Tarefa 1. Desta forma, simplesmente alterou-se a condição de aceitação e rejeição das expressões e do alfabeto. O Quadro 4 contém o código da implementação da Tarefa e o Quadro 5 contém o resultado obtido.

Quadro 4: `tarefa2/tarefa2.l`

```
%{
#define ACEITA      1
#define REJEITA     2
#define FORA        3
%}
delim       [ \t\n\r]
ws          {delim}+
par0        1*(1*01*0)*1*
par1        0*(0*10*1)*0*
alfabeto    [01]*
separador   \-+
string      [^ \t\n\r]+

%%
{ws}        {;}
{par0}      {return REJEITA;}
{par1}      {return REJEITA;}
{alfabeto}  {return ACEITA;}
{separador} {printf("--------------------------------------\n");}
{string}    {return FORA;}
%%

int main(int argc, char* argv[])
{
    int i;

    while(i = yylex())
    {
        switch(i)
        {
            case ACEITA:
                printf("%-30s: Aceita \n", yytext);
                break;
```

```
            case REJEITA:
                printf("%-30s: Rejeita \n", yytext);
                break;
            case FORA:
                printf("%-30s: Rejeita (fora do alfabeto) \n", yytext);
                break;
        }
    }
    return 0;
}
```

Quadro 5: Resultado da Tarefa 2

```
01                             : Aceita
10                             : Aceita
1011                           : Aceita
0010                           : Aceita
000010                         : Aceita
1010101010                     : Aceita
0000011111                     : Aceita
---------------------------------------
0                              : Rejeita
1                              : Rejeita
000                            : Rejeita
111                            : Rejeita
00                             : Rejeita
0000                           : Rejeita
00000                          : Rejeita
11                             : Rejeita
1111                           : Rejeita
111111                         : Rejeita
001                            : Rejeita
011                            : Rejeita
110                            : Rejeita
0011                           : Rejeita
0011                           : Rejeita
0101                           : Rejeita
01001                          : Rejeita
01010                          : Rejeita
10101                          : Rejeita
---------------------------------------
23109278912781                 : Rejeita (fora do alfabeto)
231827812                      : Rejeita (fora do alfabeto)
adhyb189h1                     : Rejeita (fora do alfabeto)
219901                         : Rejeita (fora do alfabeto)
00a11                          : Rejeita (fora do alfabeto)
hello                          : Rejeita (fora do alfabeto)
world                          : Rejeita (fora do alfabeto)
```

Tarefa 3

Escrever um programa em *Flex* reconhecedor de cadeias sobre o alfabeto {0, 1, 2}, tais que o número de dígitos 2 seja divisível por 5 (Obs: zero é divisível por 5).

Para esse exercício agora inclui o dígito 2 no alfabeto. De qualquer forma, a implementação realizada seguiu o mesmo padrão anterior. O Quadro 6 contém o código da implementação da Tarefa e o Quadro 7 contém o resultado obtido.

Quadro 6: `tarefa3/tarefa3.l`

```
%{
#define ACEITA      1
#define REJEITA     2
#define FORA        3
%}
delim       [ \t\n\r]
ws          {delim}+
aceita      [01]*([01]*2[01]*2[01]*2[01]*2[01]*2)*[01]*
alfabeto    [012]*
separador   \-+
string      [^ \t\n\r]+

%%
{ws}        {;}
{aceita}    {return ACEITA;}
{alfabeto}  {return REJEITA;}
{separador} {printf("--------------------------------------\n");}
{string}    {return FORA;}
%%

int main(int argc, char* argv[])
{
    int i;

    while(i = yylex())
    {
        switch(i)
        {
            case ACEITA:
                printf("%-30s: Aceita \n", yytext);
                break;
            case REJEITA:
                printf("%-30s: Rejeita \n", yytext);
                break;
            case FORA:
                printf("%-30s: Rejeita (fora do alfabeto) \n", yytext);
                break;
        }
    }
    return 0;
}
```

Quadro 7: `Resultado da Tarefa 3`

```
0                              : Aceita
1                              : Aceita
010                            : Aceita
01010101                       : Aceita
101010011010                   : Aceita
22222                          : Aceita
0222221                        : Aceita
020120120120120000000          : Aceita
00000222222222211110           : Aceita
```

```
02220222122202                : Aceita
--------------------------------------
2                             : Rejeita
22                            : Rejeita
222                           : Rejeita
2222                          : Rejeita
22220                         : Rejeita
222222                        : Rejeita
00001112220200000             : Rejeita
022202221222                  : Rejeita
--------------------------------------
0222302221222                 : Rejeita (fora do alfabeto)
djajvj211290u1                : Rejeita (fora do alfabeto)
hello                         : Rejeita (fora do alfabeto)
world                         : Rejeita (fora do alfabeto)
11299809124                   : Rejeita (fora do alfabeto)
991241989124                  : Rejeita (fora do alfabeto)
3020120120120120000000        : Rejeita (fora do alfabeto)
300000222222222211110         : Rejeita (fora do alfabeto)
```

Tarefa 4

Escrever um programa em *Flex* reconhecedor de cadeias sobre o alfabeto {0, 1}, com no mínimo cinco caracteres, tais que qualquer bloco de cinco caracteres consecutivos contenha no mínimo três dígitos 1.

Para resolver esta tarefa, pareceu-me mais fácil pensar no caso complementar. Rejeitar as cadeias do alfabeto que contenham algum bloco de cinco caracteres consecutivos com menos de três digito 1. A implementação foi feita negando-se todas as possíveis ocorrências desse padrão. Talvez essa não seja a implementação mais elegante, mas foi uma implementação bastante direta e resolveu o problema. O Quadro 8 contém o código da implementação da Tarefa e o Quadro 9 contém o resultado obtido.

Quadro 8: `tarefa4/tarefa4.l`

```
%{
#define ACEITA      1
#define REJEITA     2
#define FORA        3
%}
delim       [ \t\n\r]
ws          {delim}+
rejeita     [01]*(00000|10000|01000|00100|00010|00001|11000|10100|10010|10001|01100|01010|01001|00110|00101|00011)[01]*
alfabeto    [01]{5,}
separador   \-+
string      [^ \t\n\r]+

%%
{ws}        {;}
{rejeita}   {return REJEITA;}
{alfabeto}  {return ACEITA;}
{separador} {printf("------------------------------------------------------------\n");}
{string}    {return FORA;}
%%

int main(int argc, char* argv[])
```

```
{
    int i;

    while(i = yylex())
    {
        switch(i)
        {
            case ACEITA:
                printf("%-50s: Aceita \n", yytext);
                break;
            case REJEITA:
                printf("%-50s: Rejeita \n", yytext);
                break;
            case FORA:
                printf("%-50s: Rejeita (fora do alfabeto) \n", yytext);
                break;
        }
    }
    return 0;
}
```

Quadro 9: `Resultado da Tarefa 4`

```
11100                                                   : Aceita
111011111011101110111                                   : Aceita
11111111                                                : Aceita
10101                                                   : Aceita
00111001111011110101111011100111001101110111           : Aceita
10011                                                   : Aceita
-------------------------------------------------------------
000000000                                               : Rejeita
0001000100                                              : Rejeita
0000010001010                                           : Rejeita
1000110101111111111100000111101111                      : Rejeita
0010110010000111001                                     : Rejeita
111111111110000111111111                                : Rejeita
-------------------------------------------------------------
0222302221222                                           : Rejeita (fora do alfabeto)
djajvj211290u1                                          : Rejeita (fora do alfabeto)
hello                                                   : Rejeita (fora do alfabeto)
world                                                   : Rejeita (fora do alfabeto)
11299809124                                             : Rejeita (fora do alfabeto)
991241989124                                            : Rejeita (fora do alfabeto)
3020120120120120000000                                  : Rejeita (fora do alfabeto)
300000222222222211110                                   : Rejeita (fora do alfabeto)
```

Tarefa 5

Escrever um programa em *Flex* para fazer análise léxica de uma mini-linguagem que contenha os seguintes átomos: identificadores (ID), constantes inteiras (CTINT), constantes reais (CTREAL), operadores aditivos (OPAD), operadores multiplicativos (OPMULT), abre e fecha-parentesis (ABPAR e FPAR), abre e fecha-chaves (ABCHAV e FCHAV), sinal de atribuição (ATRIB), vírgula e ponto-e-vírgula (VIRG e PVIRG) e ainda as palavras reservadas `program`, `var`, `int` e `real`.

O programa implementado para a resolução da Tarefa 5 seguiu as ideias utilizadas nos slides de explicação para os Programas 1.8 e 1.9 das aulas de prática. O Quadro 8 contém o código da implementação da Tarefa. Para o arquivo de entrada, tentei simular a construção de um programa nessa linguagem construída e o Quadro 12 contém o arquivo de entrada utilizado para esta Tarefa. Além disso, o Quadro **??** contém o resultado da execução do programa.

Quadro 10: `tarefa5/tarefa5.l`

```
%{
#include <string.h>
#define ID          1
#define CTINT       2
#define CTREAL      3
#define OPAD        4
#define MAIS        5
#define MENOS       6
#define OPMULT      7
#define VEZES       8
#define DIV         9
#define ABPAR       10
#define FPAR        11
#define ABCHAV      12
#define FCHAV       13
#define ATRIB       14
#define VIRG        15
#define PVIRG       16
#define PROGRAM     17
#define VAR         18
#define INT         19
#define REAL        20
#define INVAL       21
union {
    char string[50];
    int atr, valor;
    float valreal;
    char carac;
} yylval;
%}
delim               [ \t\n\r]
ws                  {delim}+
digito              [0-9]
letra               [A-Za-z]
ctint               {digito}+
ctreal              {digito}+\.{digito}*
id                  {letra}({letra}|{digito})*

%%
{ws}                {;}
program             {return PROGRAM;}
var                 {return VAR;}
int                 {return INT;}
real                {return REAL;}
{id}                {strcpy(yylval.string, yytext); return ID;}
{ctint}             {yylval.valor = atoi(yytext); return CTINT;}
{ctreal}            {yylval.valreal = atof(yytext); return CTREAL;}
"+"                 {yylval.atr = MAIS; return OPAD;}
"-"                 {yylval.atr = MENOS; return OPAD;}
"*"                 {yylval.atr = VEZES; return OPMULT;}
"/"                 {yylval.atr = DIV; return OPMULT;}
"("                 {return ABPAR;}
")"                 {return FPAR;}
```

```
"{"                 {return ABCHAV;}
"}"                 {return FCHAV;}
"="                 {return ATRIB;}
","                 {return VIRG;}
";"                 {return PVIRG;}
.                   {yylval.carac = yytext[0]; yylval.atr = INVAL; return INVAL;}
%%

int main (int argc, char* argv[]) {
    int i;
    printf ("\n   texto  |   tipo   | atributo  \n");
    printf ("-------------------------------- \n");

    while (i = yylex ()) {
        printf ("%10s|%10d|", yytext, i);
        switch (i) {
            case ID:
                printf ("%10s", yylval.string); break;
            case CTINT:
                printf ("%10d", yylval.valor); break;
            case CTREAL:
                printf ("%10f", yylval.valreal); break;
            case OPAD:
                printf ("%10d", yylval.atr); break;
            case OPMULT:
                printf ("%10d", yylval.atr); break;
            case INVAL:
                printf ("%10c", yylval.carac); break;
        }
        printf ("\n");
    }
    return 0;
}
```

Quadro 11: `tarefa5/tarefa5.dat`

```
int program() {
    var int dia, idade;
    var real saldo;

    idade = 31;
    dia = 0;
    saldo = 31415.21;

    dia = dia + 1;
    saldo = saldo - 1000;

    dia = dia + 1;
    saldo = saldo / 2;

    dia = dia + 1;
    saldo = saldo + 15000.0;

    dia = dia + 21;
    saldo = saldo * 42;
}
```

Quadro 12: Resultado da Tarefa 5

```
   texto  |   tipo   | atributo
--------------------------------
       int|        19|
   program|        17|
         (|        10|
         )|        11|
         {|        12|
       var|        18|
       int|        19|
       dia|         1|       dia
         ,|        15|
     idade|         1|     idade
         ;|        16|
       var|        18|
      real|        20|
     saldo|         1|     saldo
         ;|        16|
     idade|         1|     idade
         =|        14|
        31|         2|        31
         ;|        16|
       dia|         1|       dia
         =|        14|
         0|         2|         0
         ;|        16|
     saldo|         1|     saldo
         =|        14|
  31415.21|         3|31415.210938
         ;|        16|
       dia|         1|       dia
         =|        14|
       dia|         1|       dia
         +|         4|         5
         1|         2|         1
         ;|        16|
     saldo|         1|     saldo
         =|        14|
     saldo|         1|     saldo
         -|         4|         6
      1000|         2|      1000
         ;|        16|
       dia|         1|       dia
         =|        14|
       dia|         1|       dia
         +|         4|         5
         1|         2|         1
         ;|        16|
     saldo|         1|     saldo
         =|        14|
     saldo|         1|     saldo
         /|         7|         9
         2|         2|         2
         ;|        16|
       dia|         1|       dia
         =|        14|
       dia|         1|       dia
         +|         4|         5
         1|         2|         1
         ;|        16|
```

```
60      saldo|         1|     saldo
61          =|        14|
62      saldo|         1|     saldo
63          +|         4|         5
64   15000.0|         3|15000.000000
65          ;|        16|
66        dia|         1|       dia
67          =|        14|
68        dia|         1|       dia
69          +|         4|         5
70         21|         2|        21
71          ;|        16|
72      saldo|         1|     saldo
73          =|        14|
74      saldo|         1|     saldo
75          *|         7|         8
76         42|         2|        42
77          ;|        16|
78          }|        13|
```

Curiosidade: Linguagem `rockstar`

Navegando pela internet acabei esbarrando em uma linguagem de programação chamada rockstar. A premissa dessa linguagem é que o código-fonte se pareça com a letra de uma música de rock. Claro que trata-se de uma linguagem feita mais com o inuito te entreter do que realmente servir para fins práticos. Mas o interessante é que diversas pessoas utilizaram essa linguagem para fins didáticos e criaram compiladores, interpretadores e transpiladores para essa linguagem. É possível encontrar projetos até mesmo utilizando *Flex* & *Yacc*. Quando eu tiver um tempo, pretendo dar uma olhada com mais calma na documentação da linguagem e quem sabe começar um projeto nesse sentido (só por diversão mesmo, sem compromisso de concluir o projeto de fato). De qualquer forma achei bem interessante e achei válido compartilhar. Deixarei abaixo alguns links interessantes sobre essa linguagem.

`Site Oficial:` https://codewithrockstar.com/
`GitHub:` https://github.com/RockstarLang/rockstar
(Na descrição tem o link de alguns projetos de compiladores/interpretadores/transpiladores)