MAC211 – Laboratório de Programação I
Segunda Prova — 20 de junho de 2013

Gabarito

Questão 1 (valor: 1 ponto)

Escreva a saída que o processador de macros `m4` gera para a entrada a seguir. Justifique sua resposta mostrando as expansões que são feitas.

```
define(`Func',`ifelse(eval($1<=1),1,1,`eval($1*Func(eval($1-1)))')')
Func(3)
```

Obs.: `eval` é uma macro pré-definida cuja a expansão dá o resultado da expressão numérica ou lógica passada como parâmetro. Ela aceita os principais operadores numéricos e lógicos de C.

Resposta:
A expansão de `Func(3)` resulta no número 6. A macro `Func` implementa a função fatorial.

Func(3)
> ifelse(**eval(3<=1)**,1,1,`eval(3*Func(**eval(3-1)**))')
> ifelse(0,1,1,`eval(3*Func(**eval(3-1)**))')
> eval(3* Func(**eval(3-1)**))
> eval(3* **Func(2)**)
> eval(3* ifelse(**eval(2<=1)**,1,1,`eval(2*Func(**eval(2-1)**))'))
> eval(3* ifelse(0,1,1,`**eval(2*Func(eval(2-1)))**'))
> eval(3* eval(2*Func(**eval(2-1)**)))
> eval(3* eval(2***Func(1)**))
> eval(3* eval(2* ifelse(**eval(1<=1)**,1,1,`eval(1*Func(eval(1-1)))')))
> eval(3* eval(2* ifelse(1,1,**1**,`eval(1*Func(eval(1-1)))')))
> eval(3* eval(2*1)))
> eval(3*2)
> 6

Questão 2 (valor: 2 pontos)

Escreva uma expressão regular (usando metacaracteres das expressões estendidas) que reconheça **somente**:

a) *strings* constantes, ou seja, qualquer cadeia de caracteres delimitada por aspas.
Uma *string* constante também pode conter dentro dela aspas, contanto que essas aspas sejam precedidas por '\'. Por exemplo, a *string* `‘‘exemplo \‘‘ de cadeia’’` é válida.

```
‘‘([^’’]|[\]’’)*’’
```

b) *strings* que contenham exatamente 5 vogais sem repetições que apareçam ordenadas alfabeticamente. As vogais não precisam aparecer uma ao lado da outra.

```
[^AaEeIiOoUu]*[Aa][^AaEeIiOoUu]*[Ee][^AaEeIiOoUu]*[Ii][^AaEeIiOoUu]*[Oo][^AaEeIiOoUu]*[Uu][^AaEeIiOoUu]*
```

c) números binários cuja quantidade de dígitos seja par.

```
([01][01])*
```

d) números em ponto flutuante. Exemplos de números válidos: `-0.3e10`, `1.342E+89`, `+.25E-7`, `2e48`, `e-783`.

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

e) as horas fornecidas por um relógio digital de 24 horas. Exemplos de horas válidas: `0:00` ou `00:00`, `7:18` ou `07:18`, `12:04`, `19:45`, `23:59`.

```
(2[0-3]|[0-1]?[0-9]):[0-5][0-9]
```

Questão 3 (valor: 2,5 pontos)

Crie um *script* em `awk` que gera um relatório a partir de dados de vendas armazenados em um arquivo texto em que cada linha contém os seguintes campos (separados por espaços):

[data da venda] [cod. produto] [categoria] [preço unitário] [quantidade]

Um exemplo de conteúdo para um arquivo desse tipo é o mostrado a seguir:

```
24/05/2013    1234245    livro      34.90      2
19/02/2013    8293828    cd         19.90      1
02/05/2013    2781020    celular    699.90     3
12/04/2013    1232245    livro      29.90      2
07/05/2013    9823298    dvd        25.90      1
23/04/2013    1234245    livro      33.90      5
28/03/2013    9823298    dvd        27.90      1
```

O arquivo pode conter outras categorias além das que aparecem no exemplo.

O relatório deve considerar somente dados de vendas realizadas nos meses de abril e maio de 2013. O relatório deve conter:

- o valor total das vendas por categoria;
- o código do produto mais vendido;
- o preço médio dos livros.

Resposta:

```
/^[0-9][0-9]\/0[45]\/2013/  {
        total_vendas[$3] += $4 * $5
        cont_produto[$2] += $5
        if ($3 ~ /livro/) {
            preco_livro += $4
            cont_livro++
        }
    }

END    {
        print ">>> Total de vendas por categoria"
        for (categoria in total_vendas) {
            print "\t", categoria,"\t: ",total_vendas[categoria]
        }
        for (produto in cont_produto) {
            if (cont_produto[produto] > mais_vendido) {
                mais_vendido = cont_produto[produto]
                prod = produto
            }
        }
        print ">>> O produto mais vendido foi: ", prod, "["mais_vendido" unidades]"
        preco_livro /= cont_livro
        print ">>> Preco medio do livro: ", preco_livro
    }
```

Questão 4 (valor: 4 pontos)

Considere um arquivo contendo anúncios de venda de carros e casas apresentados como a seguir:

```
Casa 34m2 Vila Indiana R$180900.90
Carro Kombi 1977 R$4500.00
Carro Exo Sport 2010 R$52790.00
Casa 62m2 Consolacao R$439000.90
Casa 230m2 Jardim Europa R$1780230.90
Carro Conda Hivic Sedan 2012 R$68507.30
```

Cada linha do arquivo contém um anúncio de venda. A primeira palavra da linha indica o tipo da venda: de carro ou de casa. Para uma casa, as demais informações que aparecem na linha são:

- a metragem da casa (um número inteiro seguido por "m2", de metros quadrados);
- o bairro onde a casa está localizada (uma ou mais palavras separadas por espaço);
- o preço de venda da casa (um número decimal precedido por "R$").

No caso de uma venda de carro, as demais informações que aparecem na linha são:

- o modelo do carro (uma ou mais palavras separadas por espaço);
- o ano do carro;

- o preço de venda do carro (um número decimal precedido por "R$").

a) Escreva uma especificação em `bison` que gere um *parser* (analisador sintático) para um arquivo de anúncios como o descrito acima. Para um arquivo de entrada no formato correto, o seu *parser* deve gerar como saída:
   - a metragem média e o preço médio das casas
   - o ano do carro mais velho

   Considere que os símbolos terminais (tipos de *tokens*) usados nas regras gramaticais que você definir serão reconhecidos por um analisador léxico gerado com o `flex` (veja o item (b)).

Resposta:

```
%{  #include <stdio.h>
    #define YYSTYPE double

    double metragem_total = 0, valor_casas = 0, menor_ano = 3000;
    int cont_casas = 0;                                          %}

%token TCARRO TCASA VALOR METRAGEM ANO TEXTO PALAVRA

%%

arquivo         : /* vazio */ | arquivo linha;

linha           : '\n' | anuncio '\n';

anuncio         : anuncio_carro | anuncio_casa ;

anuncio_carro   : TCARRO ' ' texto ' ' ANO ' ' VALOR
            {    if ($5 < menor_ano)
                    menor_ano = $5;        };

anuncio_casa    : TCASA ' ' METRAGEM ' ' texto ' ' VALOR
            {   cont_casas++;
                valor_casas += $7;
                metragem_total += $3;      };

texto           : PALAVRA | texto ' ' PALAVRA;

%%
int main(int argc, char** argv){
   yyparse();

   if (menor_ano != 3000)
       printf ("Menor ano de carro = %d \n", (int)menor_ano);

   if (cont_casas > 0) {
       metragem_total /= cont_casas;
       valor_casas /= cont_casas;
       printf ("Casas: valor medio = R$%f e metragem media = %fm2 \n",
                                          valor_casas, metragem_total);
   }
}

yyerror (char * s) {
   fprintf(stderr,"%s\n", s);
}
```

b) Escreva uma especificação em `flex` que gere um analisador léxico para o *parser* do item (a). O seu analisador deve reconhecer *tokens* tais como: as palavras "reservadas" `Carro` e `Casa`, uma metragem, um preço, um ano, uma palavra qualquer (entre outros).
Na sua especificação, na ação correspondente a cada regra léxica criada, não se esqueça de devolver o tipo do *token* e também atribuir a ele um valor semântico (caso esse *token* influencie os resultados gerados como saída no *parser* do item (a)).

Resposta:

```
%{  #include <stdlib.h>
    #include "parser.tab.h"      %}

NUM_INT     [0-9]+
NUM_REAL    [0-9]+[[.][0-9]{1,2}]?
BRANCO      [ \t]*

%%
Casa            { return TCASA; };

Carro           { return TCARRO; }

R${NUM_REAL}    { sscanf (yytext+2, "%lf", &yylval); return VALOR;  }

{NUM_INT}m2     { yytext[strlen(yytext)-2] = '\0';
                  sscanf (yytext, "%lf", &yylval); return METRAGEM; }

[0-9]{4}        { sscanf(yytext, "%lf", &yylval); return ANO; }

[A-Za-z]+       { return PALAVRA; }

{BRANCO}        { return ' '; }

\n|.            { return yytext[0]; }

%%
```

Questão 5 (valor: 1,0 ponto)

Considere a função `dobro`, definida a seguir como uma macro em C.

```
#define  dobro(x)    x + x
```

Escreva dois trechos de código em C com exemplos de uso da função `dobro` que geram erros (resultados diferentes do esperado). Explique os seus exemplos e os respectivos erros decorrentes. Cada trecho de código deve ilustrar um tipo de erro diferente.

Resposta:

```
int y = 3 * dobro(2+1);
```

Problema: `y` receberá o valor `7 = 3 * 2 + 1`, enquanto o esperado era o valor `9 = 3 * (2 + 1)`.

```
int a, b = 10;
a = dobro(b++);
printf(‘‘%d’’, b);
```

Problema: O comando printf exibirá o valor 12, enquanto o esperado era 11 (a substituição de `dobro(b++)` por `b++ + b++`, embora resulte no valor correto para soma, causa um incremento adicional da variável `b`).