MAC211 – Laboratório de Programação I

Prova Substitutiva — 27 de junho de 2013 — Gabarito (parcial)

Questão 1 (valor: 2,5 pontos) – Linguagem de Montagem

Dizemos que um inteiro positivo n é *palíndromo* se o 1º algarismo de n é igual ao seu último algarismo, o 2º algarismo de n é igual ao penúltimo algarismo, e assim sucessivamente.
Exemplos: 567765 e 32423 são palíndromos; 567675 não é palíndromo.
Implemente uma função em linguagem de montagem que recebe como parâmetro um número inteiro positivo, verifica se o número é palíndromo e imprime na tela uma mensagem com o resultado.
Sua função deve poder ser chamada a partir de um programa em C, ou seja, sua implementação deve respeitar as convenções usadas pela linguagem C sobre como a passagem de parâmetros deve ser feita e sobre como a pilha de dados deve ser usada pela função.

Observações: Seu código deve funcionar em uma arquitetura de 32 bits; considere que um número inteiro nessa arquitetura ocupa 2 bytes. Para imprimir a mensagem na tela, você pode usar:

- a chamada ao sistema `write(int arq_num, char* msg, int tamanho_msg)`, sendo que o número da saída padrão (`stdout`) é 1 **ou**
- a função da biblioteca padrão de C `printf(char* formato [, outros argumentos])`

Resolução (uma das várias possíveis):

```
global palindromo
extern printf

section .data
mensagem1: db 'Eh palindromo!',10,0     ; msg sucesso
mensagem2: db 'Nao eh palindromo!',10,0 ; msg falha
tamanho1: equ $ - mensagem1
tamanho2: equ $ - mensagem2

n:       equ  +8  ; deslocamento p/ o param. da funcao
reverso: equ -14  ; deslocamento p/ a variavel local

section .text
palindromo:
    ; define o frame da pilha da funcao
    push    ebp
    mov    ebp, esp
    ; salva registradores
    push     bx
    push     cx
    push     dx
    ; aloca espaco para a variavel local
    sub esp, 2

    mov    [ebp+reverso], word 0   ; reverso <- 0
    mov ax, word [ebp+n]           ; ax <- n
    mov bx, 10                     ; bx <- 10

loop:   ; laco para "descascar" n
    ; obtem o ultimo digito do numero em ax
    xor dx, dx     ; zera dx
    div bx
    mov cx, dx     ; cx armazena o digito
    push ax        ; empilha o numero "descascado"
    ; inclui o digito no reverso
    mov ax, word [ebp+reverso]
    mul bx
    add ax, cx
    mov [ebp+reverso], ax

    pop ax        ; recupera o numero "descascado"
    cmp ax,0      ; se ele nao eh zero, ainda ha mais
    jnz loop      ;     digitos para descascar

    ; compara n com o seu reverso
    mov cx, [ebp+n]
    cmp cx, word[ebp+reverso]
    je    igual
    mov ecx, mensagem2  ; endereço da msg de falha
    mov edx, tamanho2   ; tamanho da string de msg

    jmp fim

igual:
    mov ecx, mensagem1  ; endereço da msg de sucesso
    mov edx, tamanho1   ; tamanho da string de msg

fim:
    ;  mostra o resultado na tela
    ; sys_write(stdout, mensagem, tamanho)
    mov eax, 4 ; eax <- num da chamada a sys_write
    mov ebx, 1 ; ebx <- numero do stdout
    int 80h    ; faz a chamada ao nucleo (kernel)

    ; desaloca espaço das vars locais
    add esp, 2
    ; recupera registradores
    pop dx
    pop cx
    pop bx
    ; recupera base da pilha do chamador
    pop    ebp
    ret
```

Questão 2 (valor: 2,5 pontos) – Interfaces e Bibliotecas

Faça a interface (arquivo `.h`) e a implementação (arquivo `.c`) de uma biblioteca em C que gerencie listas de tarefas (*to-do lists*). As informações básicas de uma tarefa são a descrição do que tem que ser feito e uma data limite (prazo) para a realização da mesma.
Sua biblioteca pode oferecer funcionalidades tais como: inclusão, remoção e alteração de tarefas; listagem das tarefas com prazo em uma dada data; listagem ordenada por prazo das tarefas ainda não concluídas; exportação das tarefas para um arquivo texto. Esses são apenas alguns exemplos de funcionalidades possíveis; você é quem deve definir a interface e as estruturas de dados mais apropriadas para lidar com o problema proposto.
Lembre-se de considerar as boas práticas de projeto de interfaces e implementação de bibliotecas.

Dica: a ordenação de datas fica mais fácil quando elas estão no formato "yyyy-mm-dd" (por exemplo, 25/12/2013 $\simeq$ "2013-12-25").

Questão 3 (valor: 2 pontos) – Awk

Um jornalista do fictício JIME (Jornal do IME) tem o hábito de "colecionar" em um arquivo texto mensagens que coleta da internet sobre tópicos da atualidade. Nesse arquivo, há dois tipos de entradas (linhas): *tweets* e *posts* de *blogs*. Um exemplo de um trecho desse arquivo é mostrado a seguir (com 2 *posts* de *blog* e 6 *tweets*).

```
31/05/2013 http://comitepopularsp.wordpress.com/ A força das ideias contra a ideia da força: Copa pra Quem?
23/06/2013 05:28 @planeta_diario #Brasil : País terá mais de 50 #manifestações até domingo; veja lista\
 em http://tinyurl.com/19rs06S #OGiganteAcordou
23/06/2013 13:52 @clark_kent A vitória das #manifestações leva mais gente às ruas;  O movimento convoca\
 a sua maior passeata #Brasil
25/06/2013 04:09 @biro_biro Amanhã tem o primeiro jogo da semifinal! Brasil x Uruguai, direto do Mineirão,\
 às 16:00! #VemPraRuaBrasil
25/06/2013 http://googlebrasilblog.blogspot.com.br/ Transparency Report: Por uma web mais segura
25/06/2013 22:17 @lois_lane #PEC37 é derrubada na câmara! Vencemos mais uma! Essa é uma vitória do povo\
 brasileiro! #Brasil #OGiganteAcordou
26/06/2013 20:45 @falcao Avante #Brasil ! É a seleção canarinha detonando na #CopaDasConfederações !
27/06/2013 10:39 @climatemponews O dia ficará #chuvoso na maior parte do estado de #SP .
```

Um *tweet* possui a data e hora de postagem, o usuário que fez o *post* e a mensagem. A mensagem pode conter *hashtags*, ou seja, palavras-chave antecedidas pelo símbolo "#", que geralmente designam o assunto a que se refere a mensagem.
Um *post* de *blog* possui a data de postagem, a *url* do *blog* e o título da mensagem.
Crie um *script* em `awk` que processe a coleção de mensagens do jornalista, gerando um relatório que contenha as seguintes informações:

- Considerando apenas os *tweets* que contêm a palavra "Brasil":
  - as 10 *hashtags* mais usadas (*top 10 trending topics*)
  - o período do dia (diurno ou noturno) em que a maioria dos *tweets* é feita. Considere diurno qualquer horário entre 06:00 e 17:59. De forma complementar, qualquer horário entre 18:00 e 05:59 deve ser considerado noturno.
- Considerando todos os *posts* de *blogs*:
  - a data com a maior quantidade de *posts*;
  - o tamanho médio (em caracteres) do título de um *post*.

Observações: Assuma que uma mensagem ou um título de mensagem nunca possui quebras de linha. Em outras palavras, cada linha do arquivo ou é um *tweet* ou é um *post* de *blog*.

Resolução:

```
($3 ~ /^@[A-Za-z][_A-Za-z0-9]*$/) && ($0 ~ /Brasil/) {
                for (i=4; i <= NF; i++)
                        if ($i ~ /^[#]/)
                                hashtags[$i]++
                if ($2 ~ /(0[6-9]|1[0-7]):[0-5][0-9]/)
                        diurno++
                else
                        noturno++
        }

($2 ~ /^http:/) {
                for (i=3; i <= NF; i++)
                        tam += length($i)

                contp_data[$1]++
                if (contp_data[$1] > contp_data[melhor_data])
                        melhor_data = $1

                contp++
        }

END {
        print "Tweets"
        print " >>> Trending topics: "
        for (i=1; i <= 10; i++) {
                for (tag in hashtags)
                        if (hashtags[tag] > hashtags[maior])
                                maior = tag
                print " \t ", maior
                hashtags[maior] = 0
          }

        if (diurno > noturno)
                print " >>> O período do dia que tem mais tweets é diurno"
        else
                print " >>> O período do dia que tem mais tweets é noturno"

        printf "\nPosts\n"
        print " >>> Tamanho médio do título: ", tam/contp, " caracteres"
        print " >>> Data com mais posts: ", melhor_data
    }
```

Questão 4 (valor: 2 pontos) Flex + Bison

Escreva uma especificação para o `bison` e outra para o `flex` que, quando combinadas, gerem um programa que funcione como uma calculadora. Uma entrada válida para esse programa são expressões numéricas na notação infixa separadas por quebras de linha (‘\n’). Uma expressão numérica pode conter parênteses e as operações de soma (‘+’) e subtração (‘-’). Uma peculiaridade dessa calculadora é que ela deve aceitar dois tipos de expressões: (i) expressões envolvendo somente operandos que são números na base decimal e (ii) expressões envolvendo somente operandos que são números na base hexadecimal. Considere que números na base hexadecimal sempre possuem o prefixo “0x”.
Para cada expressão válida lida da entrada padrão, seu programa deve imprimir na saída padrão o valor resultante da avaliação da expressão. Exemplos de expressões e seus resultados:

```
((1.5 + 3) - 2.75)
>>> RESULTADO: 1.75
```

```
0xFFF - (0x1EF + 0x2A.4)
>>> RESULTADO: 0xDE5.C
```

Para facilitar a resolução da questão, considere que as seguintes funções estão disponíveis para uso:

- `double ConverteParaDecimal(char* s)` – devolve o número na base decimal correspondente ao número hexadecimal armazenado na *string* `s`.
- `char* ConverteParaHexadecimal(double d)` – devolve uma *string* contendo o número hexadecimal correspondente ao número na base decimal `d`.

Note que você não precisa implementar essas funções, basta chamá-las quando necessário.

Resolução:
- Especificação para o Flex (*scanner*):

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

%%

0x[0-9a-fA-F]+(\.[0-9a-fA-F]+)?  { yylval = converte_dec(&yytext[2]); return HEXADECIMAL; }
[0-9]+(\.[0-9]+)?                { sscanf(yytext,"%lf", &yylval); return DECIMAL; }
[[:blank:]]*[\n]                 { return '\n'; }
[[:blank:]]*                     {  }
.                                { return yytext[0]; }
```

- Especificação para o Bison (*parser*):

```
%{
   #include <stdio.h>
   #define YYSTYPE double
   void yyerror (char *s);
%}

%token DECIMAL HEXADECIMAL
%left '+' '-'

%%

input:
      /* empty */
     | input line ;

line:
     '\n'
     | exp_dec '\n'  { printf(">>> RESULTADO: %f\n", $1); }
     | exp_hex '\n'  { printf(">>> RESULTADO: 0x%s\n", ConverteParaHexadecimal($1)); } ;

exp_dec: DECIMAL
        | exp_dec '+' exp_dec   { $$ = $1 + $3; }
        | exp_dec '-' exp_dec   { $$ = $1 - $3; }
        | '(' exp_dec ')'       { $$ = $2; } ;

exp_hex: HEXADECIMAL
        | exp_hex '+' exp_hex   { $$ = $1 + $3; }
        | exp_hex '-' exp_hex   { $$ = $1 - $3; }
        | '(' exp_hex ')'       { $$ = $2; } ;
%%

int main (void) {
   return yyparse ();
}
```

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

Questão 5 (valor: 1 ponto) – Sistemas de Controle de Versão

- Qual a principal diferença entre a arquitetura dos sistemas de controle de versão centralizados e a dos distribuídos?

  Resolução:

  O controle de versão centralizado possui 1 só repositório por projeto, enquanto o controle de versão distribuído possui múltiplos repositórios por projeto.

- Cite ao menos três vantagens dos sistemas de controle de versão distribuídos quando comparados aos centralizados.

  Resolução:

  1. É possível fazer *commits* (gravar versões dos arquivos) sem disponibilizá-las para os demais membros do projeto
  2. É possível fazer *commits* mesmo quando o acesso a internet não está disponível
  3. Operações comuns (como *commits*, consulta ao histórico e reversão de alterações) são feitas de forma rápida, já que não requerem a comunicação com um servidor central. A comunicação com os outros repositórios só é necessária quando se quer disponibilizar (*push*) ou obter (*pull*) as alterações para/de os outros membros do projeto
  4. Cada repositório local funciona como uma cópia do projeto, o que é uma proteção "natural" contra perda de dados