MAC211 – Laboratório de Programação 1 – Primeira Prova — 07/05/13

Gabarito

Questão 1 (valor: 2,5 pontos)

Dizemos que um inteiro positivo n é perfeito se for igual à soma de seus divisores positivos diferentes de n. Exemplo: 6 é perfeito, pois $1 + 2 + 3 = 6$.
Implemente uma função em linguagem de montagem que recebe como parâmetro um número inteiro positivo e devolve 1 se ele for perfeito e 0 no caso contrário.
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 e do valor de retorno deve ser feita
- como a pilha 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.

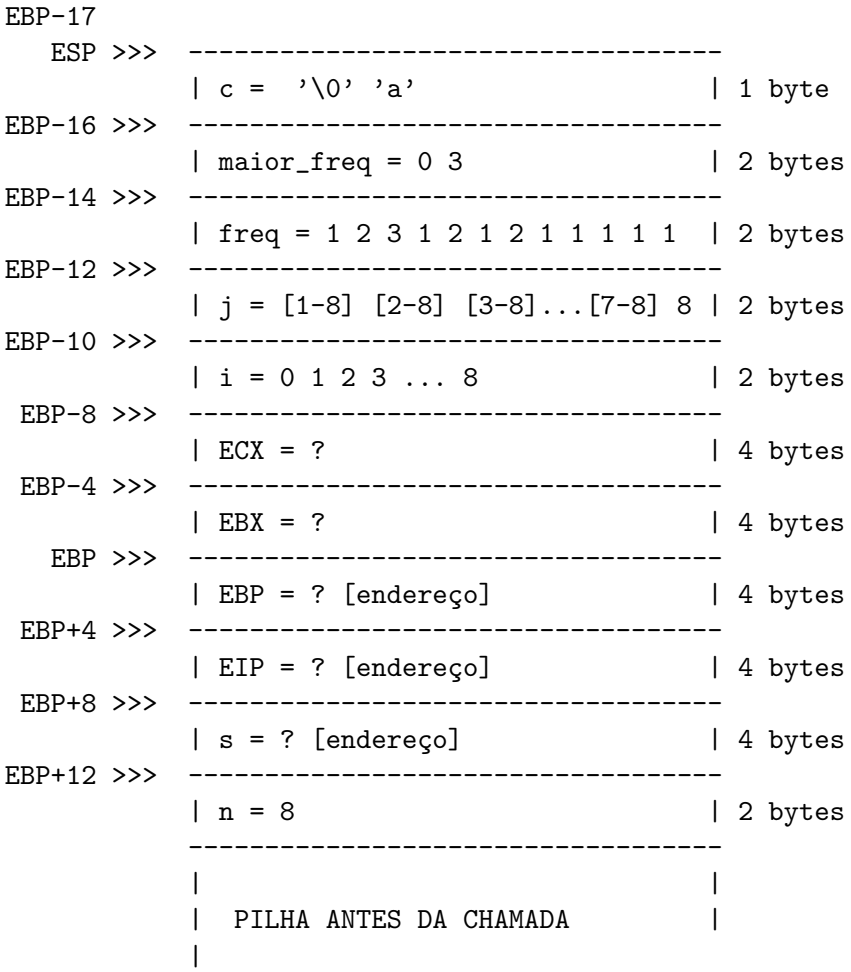
```
global seg_max
section .data

n:    equ    +8    ; deslocamento para o parametro da funcao

section .text

seg_max:
    push    ebp                ; define o frame da pilha da funcao
    mov     ebp, esp

    push    bx                 ; salva registradores
    push    cx
    push    dx

    mov     cx, 0              ; cx armazenar a soma
    mov     bx, word [ebp+n]   ; candidato a divisor de n

loop:
    dec     bx             ; vai para o proximo candidato a divisor
    jz      compara

    mov     ax, word [ebp+n]   ; armazena n em eax
    mov     dx, 0

    div     bx       ; divide dx:ax por bx, o resultado esta em ax
    cmp     dx, 0
    jne     loop     ; se bx eh divisor de n, inclui-o no soma

    add     cx, bx
    jmp     loop

compara:
    cmp     cx, [ebp+n]      ; compara n com a soma
    jne     nao
    mov     eax, 1           ; eh igual, devolve em eax o valor 1
    jmp     fim

nao:
    mov     eax, 0           ; nao eh igual, devolve em eax o valor 0

fim:
    pop     dx               ; recupera registradores
    pop     cx
    pop     bx
    pop     ebp              ; recupera base da pilha do chamador
    ret
```

Questão 2 (valor: 2 pontos)

O código em linguagem C a seguir é transformado em linguagem de montagem pelo `gcc` no Linux.
Mostre a situação completa da pilha resultante da execução das operações que vão desde a chamada à função `MaisFrequente` (linha 25 do código) até a última iteração do laço das linhas 8–18 (ou seja, a operação anterior ao `return` da função).
Observações: Considere que o programa foi gerado para uma arquitetura de 32 bits; um número inteiro nessa arquitetura ocupa 2 bytes. Considere também que a versão em linguagem de montagem da função mexe explicitamente apenas nos registradores EAX, EBX e ECX.

```
1  /* Funcao que recebe como entrada uma string e o seu tamanho e
2     devolve o caracter que aparece com maior frequencia na string */
3
4  char MaisFrequente(char *s, int n){
5      int i, j, freq, maior_freq = 0;
6      char c = '\0';
7
8      for(i = 0; i < n; i++) {
9          freq = 1;
10         for(j = i+1; j < n; j++) {
11             if (s[i] == s[j])
12                 freq++;
13         }
14         if (freq > maior_freq) {
15             maior_freq = freq;
16             c = s[i];
17         }
18     }
19     return c;
20  }
21
22 int main(){
23    char c;
24
25    c = MaisFrequente(``ararinha'', 8);    /* Funcao devolvera o caracter 'a' */
26    ...
27 }
```

Solução:



Questão 3 (valor: 2,5 pontos)

(a) Escreva em linguagem C o filtro `semcomentario`, que é um programa que recebe um arquivo texto na sua entrada padrão e que o joga para a saída padrão removendo os comentários. O arquivo de entrada pode conter dois tipos de comentários:

- comentário de linha: iniciado pelo caracter '#'. A ocorrência de um '#' em uma linha indica que todos os caracteres que aparecem depois dele na linha são comentários
- comentário de bloco: delimitado pelo par '[' e ']'. Todos os caracteres que aparecem entre um par '[' e ']' são comentários. Os comentários de bloco podem abranger mais de uma linha.

Exemplo do funcionamento de `semcomentario`:
Entrada

```
Este eh um exemplo de texto #para a prova de mac211.
O exemplo eh [simples e ]curto
O texto contem comentarios [de linha (#) e de
bloco ]interessantes
```

Saída

```
 Este eh um exemplo de texto
 O exemplo eh curto
 O texto contem comentarios interessantes
```

Solução: (vale 1.5)

```
#include <stdio.h>

int main(){
    int c;

    while ((c=getchar()) != EOF) {
        if (c == '#') {
            while ((c=getchar()) != EOF && c != '\n');
            if (c != EOF)
                putchar(c);
        }
        else if (c == '[') {
            while ((c=getchar()) != EOF && c != ']');
        }
        else
            putchar(c);
    }

    return 0;
}
```

(b) Usando o programa filtro criado no item anterior, escreva um *pipeline* UNIX (ou seja, uma sequência de comandos separados pelo caractere *pipe* '|') para selecionar de um arquivo texto chamado `entrada.txt` somente as linhas que contêm a *string* `''MAC211''` desconsiderando os comentários existentes no arquivo. O resultado da seleção deve ser gravado em um arquivo chamado `saida.txt`. No seu *pipeline*, além de programas utilitários do Linux, você também pode usar comandos de redirecionamento.

Solução: (vale 1)

```
cat entrada.txt | ./semcomentario | grep MAC211 > saida.txt
```

Questão 4 (valor: 1,5 pontos)

(a) Caracterize as bibliotecas estáticas e dinâmicas quanto à forma como são ligadas ao seu programa invocador.

Solução: (vale 0.5)

- **Bibliotecas estáticas** – são as que bibliotecas cujo código é acessado durante a construção (*build*) do executável do programa invocador
- **Bibliotecas dinâmicas** – são as que bibliotecas que são integradas (= ligadas) a um outro programa depois que a execução do mesmo já tiver sido invocada. A ligação das bibliotecas dinâmicas pode ocorrer de duas maneiras:
  - **na inicialização**: quando o programa é carregado para execução, todas as bibliotecas dinâmicas que ele referencia também são carregadas com ele
  - **carga preguiçosa** (mais comum): a biblioteca só é carregada quando ela for necessária para a continuidade da execução do programa

(b) Cite as principais vantagens e desvantagens das bibliotecas estáticas.

Solução: (vale 0.5)

- Vantagens
  - Não há o risco de não se localizar uma biblioteca em tempo de execução (já que o código das funções da biblioteca que um programa utiliza são adicionados ao seu executável final)
  - A versão das bibliotecas é fixa, portanto, não há o perigo de incompatibilidade de versões
  - Mudanças no comportamento das bibliotecas novas não afetam a corretude dos programas antigos
  - A carga de programas com bibliotecas estáticas é mais rápida
- Desvantagens
  - Tamanho do arquivo executável fica maior
  - O código da biblioteca é adicionados ao código executável mesmo que em tempo de execução ele não seja usado

(c) Cite as principais vantagens e desvantagens das bibliotecas dinâmicas.

Solução: (vale 0.5)

- Vantagens
  - Compartilhamento entre os vários programas que as usam (tanto na memória quanto no disco)
  - Menor uso de memória na máquina como um todo (imagine o que aconteceria se a libc fosse estática...)
  - Novas versões (com melhorias ou correções) das bibliotecas são aproveitadas mesmo pelos programas mais antigos
- Desvantagens
  - A carga de programas com bibliotecas dinâmicas é mais lenta
  - *DLL hell* – complicações frenquente relacionadas ao uso de bibliotecas dinâmicas nas versões mais antigas do Windows

Questão 5 (valor: 1,5 pontos)

A seguir, encontram-se os arquivos `fila.h` e `fila.c` contendo, respectivamente, a interface e a implementação de uma pequena biblioteca para a manipulação de filas de números inteiros.

Arquivo `fila.h`

```
1  #ifndef _FILA_H_
2  #define _FILA_H_
3
4  #define TAM_MAX_FILA 1000
5
6  typedef struct {
7    int elementos[TAM_MAX_FILA];
8    int tam;
9  } TipoFila;
10
11 TipoFila fila;  /* Mantem a fila */
12
13 /* Inicializa ou esvazia a fila,
14    atribuindo 0 ao seu tamanho   */
15 void LimpaFila();
16
17 /* Insere o elemento passado como
18    paramentro no final da fila   */
19 void InsereNaFila(int);
20
21 /* Devolve o primeiro elemento
22    da fila, removendo-o da mesma */
23 int FilaRemove();
24
25 /* Le os numeros contidos no
26    arquivo passado como parametro
27    e os inclui na lista          */
28 void CarregaFilaDeArquivo(FILE*);
29
30 #endif
```

Arquivo `fila.c`

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "fila.h"
4
5  void LimpaFila() {
6    fila.tam = 0;
7  }
8
9  void InsereNaFila(int x) {
10   if (fila.tam == TAM_MAX_FILA) {
11     printf("Fila cheia! \n");
12     exit(0);
13   }
14   fila.elementos[fila.tam++] = x;
15 }
16
17 int FilaRemove() {
18   int i, x;
19
20   if (fila.tam == 0) {
21     printf("Fila vazia! \n");
22     exit(0);
23   }
24
25   x = fila.elementos[0];
26   fila.tam--;
27   for (i=1; i <= fila.tam; i++)
28     fila.elementos[i-1] = fila.elementos[i];
29
30   return x;
31 }
32
33 void CarregaFilaDeArquivo(FILE* arq) {
34   while (!feof(arq) &&
35                fila.tam < TAM_MAX_FILA) {
36     fscanf(arq, "%d",
37                  &fila.elementos[fila.tam]);
38     fila.tam++;
39   }
40
41   fclose(arq);
42 }
```

Aponte características presentes no código acima que ferem os princípios para o desenvolvimento de boas interfaces e bibliotecas discutidos em aula. Justifique sua resposta.

Solução:
Principais problemas

1. **Ocultação dos detalhes de implementação** – problema: variável global `fila`

   - Evite o uso de variáveis globais; sempre que possível, é melhor passar os dados por meio de parâmetros para funções.
   - Não use dados que estão sempre "visíveis"; é difícil manter a consistência dos valores quando usuários podem alterar variáveis de forma indiscriminada.

2. **Consistência e regularidade** – problema: falta de consistência nos nomes das funções. Por exemplo, *InsereNaFila* e *FilaRemove*. Seria melhor ter, por exemplo, *RemoveDaFila*.

3. **Gerenciamento de recursos** – problema: a função *CarregaFilaDeArquivo* recebe como parâmetro um arquivo aberto, mas fecha-o dentro da função.

   - A liberação de um recurso deve ser feita na mesma camada em que ele foi alocado.

4. **Tratamento de erros** – problema: as funções *InsereNaFila* e *FilaRemove*, em caso de erro, encerram a execução do programa.

   - Desejável: detectar erros num nível baixo; lidar com eles num nível alto.
   - Ideia geral: o chamador é quem deve determinar a forma como o erro deve ser tratado.
   - As rotinas da biblioteca precisam colaborar com essa ideia: em casos de erro, devem falhar de forma "graciosa", ou seja, não abortando o código e retornando detalhes suficientes sobre o erro, para que o chamador possa fazer um tratamento apropriado.