PUC-Rio – Software Básico – INF1018
Prova 1 – 1/10/13

1. (2,5 pontos) Considere o código abaixo: Considere o programa C a seguir:

```
#include <stdlib.h>
#include <stdio.h>
void dump (void *p, int n) {
  unsigned char *p1 = (unsigned char *) p;
  while (n--) {
    printf("%p - %02x\n", p1, *p1);
    p1++;
  }
}

struct art {
  int y;
  char rep;
  struct art *prox;
  short x;
};

int main (void) {
  struct art A0 = {30,  'c', NULL, -8};
  struct art A1 = {-1027, 'c'+1, &A0, 5<<3};
  dump ((void *)&A1, sizeof(struct art));  return 0;
}
```

   Considerando que `A0` seja alocado na posição de memória 0xbfd98e64 e `A1` na posição de memória 0xbfd98e54, diga o que esse programa irá imprimir quando executado, explicando como você chegou aos valores exibidos (ATENÇÃO: valores sem contas e explicações NÃO valem ponto!!!). Coloque PP nas posições correspondentes a *padding*. Considere que o valor do caracter 'a' na tabela ASCII é 97, e suponha que a máquina de execução é *little-endian* e que as convenções de alinhamento são as do Linux no IA-32.

2. Traduza as funções `foo` e `boo` abaixo para assembly IA-32 (o assembly visto em sala), utilizando as regras usuais de alinhamento, passagem de parâmetros, salvamento de registradores e resultados em C. Traduza o mais diretamente possível o código de C para assembly. *Comente seu código!*

   (a) (2,5 pontos)

```
int foo (int* nums, int qtos) {
  if (qtos==0)
    return 0;
  else
    return *nums + foo(nums+1,qtos-1);
}
```

   (b) (2,5 pontos)

```
int boo (int *nums, int tam) {
  int i, indmaior, maior = -1;
  for (i=0; i<tam; i++)
    if  (nums[i] > maior) {
      maior = nums[i];
      indmaior = i;
    }
  return indmaior;
}
```

3. (2,5 pontos) No formato UTF-8, usado para representar caracteres UNICODE, caracteres têm tamanho variável. Um caractere tem tamanho mínimo de 8 bits, porém, se seu código necessitar de mais espaço para ser representado, o formato UTF-8 pode utilizar mais de um byte para representá-lo (até um máximo de quatro bytes). Os bits iniciais do primeiro (ou único) byte indicam quantos bytes são usados para a representação do caractere, conforme a seguinte convenção:

```
0xxxxxxx  - 1 byte
110xxxxx xxxxxxxx  - 2 bytes
1110xxxx xxxxxxxx xxxxxxxx  - 3 bytes
11110xxx xxxxxxxx xxxxxxxx xxxxxxxx  - 4 bytes
```

Escreva, em C, uma função *utf8strlen*, com o protótipo:

```
int utf8len (unsigned char* c);
```

Essa função recebe um ponteiro para o primeiro (ou único) byte de uma representação UTF-8 e retorna o número de bytes que essa representação ocupa.