PUC-Rio – Software Básico – INF1018
Prova 1 – Turma 3wb – 04/05/2017

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

```
#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 S1 {
  char *val_string;
  int   val_int;
};
struct S2 {
  int num;
  char cod;
  struct S1 *ps;
};
struct S1 s1[2];
struct S2 s2[2] = {{-254, 'd' << 3, &s1[0]}, {2050, 0xaa & 0xc0, &s1[1]}};
int main (void) {
  dump (s2, sizeof(s2));
  return 0;
}
```

Sabendo das informações abaixo e supondo que a máquina de execução é *little-endian* com as convenções de alinhamento do Linux no IA-64 (vistas em sala), mostre o que esse programa irá imprimir quando executado. Coloque **PP** nas posições correspondentes a *padding*.

endereço de `s1` na memória	0x6010a0
endereço de `s2` na memória	0x601060
valor do caractere 'a' na tabela ASCII	97 (decimal)

(Mostre como você chegou aos valores exibidos. Valores sem contas **NÃO** valem ponto!).

2. Traduza as funções `foo` e `boo` abaixo para assembly IA-64, utilizando as regras usuais de alinhamento, passagem de parâmetros, salvamento de registradores e resultados em C/Linux. (Não se preocupe em entender o que as funções fazem, apenas traduza-as literalmente.)

   Atenção! **Traduza o mais diretamente possível o código de C para assembly**.

   (a) (2,5 pontos)

```
struct X {
  short val0;
  int val1;
  int val2;
};

int foo (struct X vals[], int x, int y) {
  if (x < y) {
    return vals[x].val1;
  }
  else
    return vals[x].val2;
}
```

(b) (3,0 pontos)

```
#define NULL 0

struct Y {
  int n;
  struct Y *esq;
  struct Y *dir;
};

int f(int x);

int boo (struct Y *p, int i) {
  if (p == NULL)
    return 0;
  else
    if (p->n > i)
      return f(p->n) + boo(p->dir, i);
    else
      return boo(p->dir, i);
}
```

3. (2,0 pontos) No formato UTF-8, a codificação de caracteres UNICODE tem tamanho variável, de 1 a 4 bytes. A tabela abaixo mostra o número de bytes necessários para representar cada faixa de valores de códigos UNICODE, e a codificação usada para cada uma dessas faixas:

Código UNICODE	Representação em UTF-8
U+0000 - U+007F	0xxxxxxx
U+0080 - U+07FF	110xxxxx 10xxxxxx
U+0800 - U+FFFF	1110xxxx 10xxxxxx 10xxxxxx
U+10000 - U+10FFFF	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx

Escreva, em C, uma função chamada *nchars* que receba um ponteiro para uma sequência de caracteres UNICODE codificados em UTF-8 e retorne **o número de caracteres UNICODE** representados nessa sequência. O final da sequência dada como entrada para a função é indicado por um byte nulo (0x00), que não é computado no tamanho.

Por exemplo, se a sequência fornecida corresponder ao conteúdo do nosso arquivo **utf8_peq** (acrescido do byte nulo no final):

```
43 4c 41 56 45 20 f0 9d 84 9e 20 41 47 55 41 20 e6 b0 b4 20 2e 00
```

a função *nchars* deverá retornar o valor **16**.

O protótipo da função a ser implementada deve ser:

```
int nchars(char *utf8);
```

Boa Prova!