PUC-Rio – Software Básico – INF1018
Prova 1 – 30/09/2010 – Turma 3WB

1. (2,5 pontos) No formato UTF-8, caracteres tem tamanho variável. Um caracter UNICODE tem tamanho mínimo de 8 bits, porém se seu código necessitar de mais espaço para ser representado, o formato UTF-8 pode usar mais de um byte para representá-lo (até um máximo de 4 bytes). A tabela abaixo apresenta o número de bytes necessários para representar cada faixa de valores de códigos UNICODE:

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 ou assembly IA-32 usado nos laboratórios, uma função *numchars* com o protótipo:

```
void numchars(char *utf8, int n[]);
```

Essa função recebe uma string com uma sequência de caracteres UTF-8 e retorna, no array n, o número de caracteres UNICODE em cada uma das faixas encontrados nessa sequência. Ou seja, após a execução da função, `n[0]` conterá o número de caracteres UNICODE na faixa U+0000 a U+007F, `n[1]` conterá o número de caracteres UNICODE na faixa U+0080 a U+07FF, `n[2]` conterá o número de caracteres UNICODE na faixa U+0800 a U+FFFF, e, finalmente, `n[3]` conterá o número de caracteres UNICODE na faixa U+10000 a U+10FFFF.

O final da string dada como entrada para a função é indicado por um byte nulo (0x00).

2. (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++;
  }
}

char str[] = "alo";

struct X {
  char c1;
  short y;
  int x;
  char *p;
  unsigned char c2;
} x[2] = {{-12, 20, -1023, NULL, 'b' & 0xA5}, {'6'+ 2, 130 >> 3, 8192, str, 300}};

int main(void) {
  dump(x, sizeof(x));
  return 0;
}
```

Considerando que x seja armazenado no endereço de memória 0x80494e4, e que *str* no endereço 0x0x80494e0, diga o que o programa irá imprimir quando executado, explicando como você chegou

a esses valores (valores sem contas e explicações NÃO SERÃO CONSIDERADOS!). Considere que o valor do caracter 'a' na tabela ASCII é 97, e que o caracter '0' é 0x30. Suponha que a máquina de execução é *little-endian*, e que as convenções de alinhamento são as do Linux no IA-32.

3. Traduza as funções `f` e `g` abaixo para assembly IA-32 do gcc/Linux (visto em sala), utilizando as regras básicas de passagem de parâmetros e retorno de resultado em C. Comente o seu código!

   (Não se preocupe se você não entender o que as funções fazem, apenas traduza-as literalmente).

   (a) (2,5 pontos)

```
void f(int a[], int b[], int n) {
 for (i = 0; i < n; i++)
   b[i] = a[i] * 2;
}
```

   (b) (2,5 pontos)

```
struct L {
  char v;
  struct L *next;
};

int g(struct L *l, char c) {
  if (l == NULL)
    return 0;
  else {
    if (l->v == c)
      return 1;
    else
      return g(l->next, c);
  }
}
```