PUC-Rio – Software Básico – INF1018
Prova 1 – 07/05/2013 – Turma 3WB

Importante: Não esqueça de colocar o seu nome e número de matrícula na(s) folha(s) de respostas. **Leia com atenção os enunciados das questões e faça exatamente o que é pedido!**

1. (2,5 pontos) Considere o programa C a seguir (as reticências na inicialização dos elementos de **x** são propositais):

```
#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 X {
  short s;
  int i;
  char c;
} x[2] = {{...}, {...}};

int main(void) {
  dump(x, 28);
  return 0;
}
```

Ao ser executado em uma máquina de arquitetura IA-32 (*little-endian*), obedecendo as convenções de alinhamento do Linux, esse programa imprimiu a saída a seguir:

```
0x100001070 - 04
0x100001071 - 01
0x100001072 - 00
0x100001073 - 00
0x100001074 - 00
0x100001075 - f0
0x100001076 - ff
0x100001077 - ff
0x100001078 - 41
0x100001079 - 00
0x10000107a - 00
0x10000107b - 00
0x10000107c - ff
0x10000107d - fd
0x10000107e - 00
0x10000107f - 00
0x100001080 - 21
0x100001081 - 04
0x100001082 - 00
0x100001083 - 00
0x100001084 - 82
0x100001085 - 00
0x100001086 - 00
0x100001087 - 00
0x100001088 - 00
0x100001089 - ff
0x10000108a - 00
0x10000108b - ff
```

Analisando a saída do programa e a declaração de `x` (um *array* de 2 estruturas), substitua as "reticências" na inicialização dos elementos de `x` com os valores dos campos respectivos. **Expresse esses valores em notação decimal**. Mostre como você chegou a esses valores (isto é, mostre suas contas e a posição dos valores na saída do dump). Valores sem explicação não serão considerados!

ATENÇÃO: O programa pede que "dump" imprima 28 bytes. Esse número é **suficiente** para mostrar todo o conteúdo do *array*, mas **podem estar sendo mostrados bytes a mais!**

2. (2,5 pontos) No formato UTF-16, caracteres UNICODE são codificados em um ou dois *code units* de 16 bits. Códigos UNICODE na faixa **U+0000 a U+FFFF** são representados em UTF-16 pelo seu próprio valor numérico (em um único *code unit*, com 16 bits).

   Escreva, em C, uma função *unicode_utf16*, que recebe um *array* de códigos UNICODE na faixa U+0000 a U+FFFF e preenche um *buffer* com a representação desses códigos em UTF-16, na ordenação especificada. Assuma que o *array* de códigos UNICODE contém apenas códigos válidos e que o *buffer* fornecido tem espaço suficiente. Sua função deve obedecer o seguinte protótipo:

   ```
   void unicode_utf16(unsigned int *codes, int n, unsigned char *buffer, int ordem);
   ```

   Os parâmetros *codes* e *n* fornecem, respectivamente, o endereço do *array* de códigos UNICODE e seu número de elementos. O terceiro parâmetro é o endereço do *buffer* que deverá ser preenchido. O último parâmetro (*ordem*) especifica a ordenação desejada: 0 indica ordenação *little-endian* e 1 indica ordenação *big-endian*.

   Atenção! Sua função deve ser **independente** da ordenação da memória da máquina onde ela será executada!

3. Traduza as funções `f` e `g` abaixo para assembly IA-32 do gcc/Linux (visto em sala), utilizando as regras básicas de alinhamento, passagem de parâmetros, retorno de resultado e uso/salvamento de registradores. **Importante: comente o seu código!**

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

   (a) (2,5 pontos)

   ```
   #define NUM 5
   int foo(int x);
   int boo(int *a, int n);

   int f(void) {
     int i, *pa;
     int a[NUM];
     for (i = 0, pa = a; i < NUM; i++) {
       *pa = foo(i);
       pa++;
     }
     return boo(a, NUM);
   }
   ```

   (b) (2,5 pontos)

   ```
   struct N {
     int val;
     struct N *next;
   };

   int g(struct N *n, int k) {
     if (n == NULL)
       return 0;
     if (n->val == k)
         return 1;
     return g(n->next, k);
   }
   ```