PUC-Rio – Software Básico – INF1612
Prova 1 – 5/5/2010 – Turma 3WB

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 quadrado {
  void *proximo;
  char cor;
  short xcentro;
  int lado;
  short ycentro;
};

int main (void) {
  struct quadrado quadrado1 = { NULL, 'p', -5, 1040, 20};
  struct quadrado quadrado2= { (void *) &quadrado1, 'b', -2080, 580, 4100};
  dump ((void *)&quadrado2, sizeof(quadrado2));
  return 0;
}
```

   Considerando que `quadrado1` seja alocado na posição de memória 0xbfffb8b0 e `quadrado2` na posição de memória 0xbfffb9fc, 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!!!). 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 do gcc/Linux (o assembly visto em sala), utilizando as regras usuais de passagem de parâmetros e resultados em C. Use registradores para abrigar as variáveis locais que aparecem nessas funções. Comente seu código.

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

   (a) (2,5 pontos)

```
int foo (int *l, int tam) {
  if (tam == 0)
    return 0;
  else return l[0] + foo(l+1, tam-1);
}
```

   (b) (2,5 pontos)

```
struct X {
  int v;
  struct X *prox;
};

int boo (struct X *x){
  int acc = 0;
```

```
  while (x!=NULL) {
    acc += acc + x->v;
    x = x->prox;
  }
  return acc;
}
```

3. (2,5 pontos) Implemente uma função C, chamada `descompacta`, que receba como parâmetro um vetor de bytes, e escreva os dados desse vetor em um outro vetor de structs. Os dados no vetor de bytes estão guardados de forma corrida, sem *padding*, e devem ser colocados nas posições corretas do vetor de structs, seguindo as regras de alinhamento padrão da plataforma Pentium com sistema operacional Linux. Não é necessário considerar se os dados são little ou big endian.

   A sua função deve ter o seguinte protótipo:

```
struct Dado {
   char x;
   int y;
   short z;
};
void compacta (char * entrada, int tamanho, struct Dado* saida);
```

   Onde:

   - `entrada` é o endereço do vetor de bytes original, onde os dados estão armazenados sem alinhamento;
   - `tamanho` é o tamanho (número de elementos) do vetor de saída;
   - `saida` é o endereço do vetor de saída, isto é, o vetor onde devem ser escritos os *structs* bytes de alinhamento. Considere que o vetor de entrada é grande o suficiente para alimentar todos os dados que serão armazenados no vetor de saída.