PUC-Rio – Software Básico – INF1018
Prova 1 – 8/10/2009

1. (2,5 pontos) 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 coisas {
  int y;
  char rep;
  struct coisas *prox;
  short x;
};

int main (void) {
  struct coisas Coisa0 = {30,  'c', 0, -8};
  struct coisas Coisa1 = {-130, 'e', &Coisa0, 2090};
  dump ((void *)&Coisa1, sizeof(Coisa1));  return 0;
}
```

   Considerando que `Ponto0` seja alocado na posição de memória 0xbffff7a8 e `Ponto1` na posição de memória 0xbffff7b8, 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 (o assembly visto em sala), utilizando as regras usuais de alinhamento, 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! Também não se preocupe com o que a função f faz!)

   (a) (2,5 pontos)

```
int foo (char *a) {
  if (*a==0) return 0;
  else return *a - '0' + foo(a+1);
}
```

   (b) (2,5 pontos)

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

void boo (struct X *x, int sub){
  while (x!=NULL) {
    if (x->v > sub)
      x->v = sub;
    x = x->nxt;
  }
}
```

3. (2,5 pontos) Suponha as declarações usadas no trabalho:

```
#define NUM_BYTES 16
typedef unsigned char BigInt[NUM_BYTES]; /* little-endian!!! */
```

Implemente em C uma operação `bi_addbyte` com a seguinte assinatura:

```
void bi_addbyte (BigInt res, Bigint num, char b, int n);
```

A operação deve retornar em `res` o BigInt obtido somando-se o byte `b` na posição `n` de `num` (naturalmente considerando que a soma nessa posição pode ter consequências nos bytes mais significativos).

Por exemplo, se `a` for um `BigInt` com o valor:

```
BigInt a = {0xff, 0xff, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
            0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 };
```

e fizermos:

```
bi_addbyte (res, a, 0x01, 0);
```

o valor final de `a` deve ser:

```
a[0] = 0x00, 0x00, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00,
       0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
```