**PUC-Rio – Software Básico – INF1018**
**Prova 1 – Turma 3wb – 10/10/2017**

1. (2,0 pontos) No nosso primeiro trabalho, implementamos o armazenamento compactado de um array de estruturas, gerando um arquivo onde **os campos de cada estrutura** são armazenados em sequência, no seguinte formato:

   - o primeiro byte é o *cabeçalho* do campo
   - os bytes seguintes representam o conteúdo compactado do campo, em *big endian*.

   Para campos do tipo inteiro (com e sem sinal), o cabeçalho tem o seguinte formato:

| BITS: | 7 | 6-5 | 4-0 |
|---|---|---|---|
| | cont | tipo | tamanho |

   O bit mais significativo (cont) indica se este é o último campo da estrutura (1) ou não (0). Os bits 4-0 (tamanho) têm o número de bytes usados para representar o valor e os bit 6-5 (tipo) têm:

   - 00 - se o campo é do tipo **u** (unsigned int)
   - 01 - se o campo é do tipo **i** (signed int)

   Escreva, em C, uma função chamada *num_bytes_u* que receba um ponteiro para uma **região de memória** para onde foi lido **todo o conteúdo de uma única estrutura** armazenada na forma descrita acima e retorne a soma dos tamanhos de todos os campos do tipo *unsigned int* dessa estrutura. **Assuma que na estrutura lida só há campos do tipo inteiro (com e sem sinal).**

   O protótipo da função a ser escrita deve ser:

```
int num_chars_u (unsigned char *pbyte);
```

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;
  printf("tamanho = %d\n", n);
  while (n--) {
    printf("%p - %02x\n", p1, *p1);
    p1++;
  }
}

char nome[] = "Ana";
struct X {
  int   i;
  char *p;
  char  c;
};
struct X x[2] = {{-514, NULL, 'b' << 2}, {131, nome, 257}};
int main (void) {
  dump (x, sizeof(x));
  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 `nome` na memória | 0x55f42a153010 |
| endereço de `x` na memória | 0x55f42a153020 |
| valor do caractere 'a' na tabela ASCII | 97 (decimal) |

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

3. 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)

```
void foo(long x[], long y[], int n) {
  int i;
  for (i = 0, n-=1; n >=0; i++, n--) {
    y[n] = x[i];
  }
}
```

   (b) (3,0 pontos)

```
#define NULL 0

struct X {
  int v0, v1, v2;
  struct X* next;
};

int boo(struct X *p) {
  if (p == NULL) return 0;
  if (p->v0) {
    if (p->v1 > p->v2)
      return p->v1 + boo(p->next);
    else
      return p->v2 + boo(p->next);
  }
  else
    return boo(p->next);
}
```

Boa Prova!