PUC-Rio – Software Básico – INF1612
Prova 1 – 30/04/09 – Turma 3WA

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 ponto {
  char* vizinhos;
  short x;
  int y;
  char rep;
};

int main (void) {
  char coisas [2][3] = {{1,2,-1},{3,-4,2}};
  struct ponto Ponto = {coisas[0], 25, -2050,  '6'};
  dump ((void *)&Ponto, sizeof(Ponto));
  return 0;
}
```

   Considerando que `coisas` seja alocado na posição de memória 0xbffff7b8 e `Ponto` na posição de memória 0xbffff7a8, 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 '0' na tabela ASCII é 48, 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 f (int x);

int foo (int *l, int n) {
  int i; int a=0;
  for (i=0;i<n;i++)
    if (l[i] != 0)
      a += f(l[i]);
  return a;
}
```

   (b) (2,5 pontos)

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

```
void boo (struct X *x, int acc){
  while (x!=NULL) {
    if (x->v != 0)
      acc += x->v;
    else
      x->v = acc;
    x = x->filho;
  }
}
```

3. (2,5 pontos) Implemente uma função C, chamada `compacta`, que receba como parâmetro um vetor de *structs* e escreva os dados desse vetor em um outro vetor de forma compactada, isto é, sem bytes de alinhamento (*padding*). A sua função deve ter o seguinte protótipo:

```
int compacta (void* entrada, int tamanho, char campos[], void* saida);
```

Onde:

- `entrada` é o endereço do vetor de *structs* original;
- `tamanho` é o tamanho (número de elementos) do vetor original;
- `campos` é uma descrição dos *structs* que compõem o vetor (descrição dos campos). Esse parâmetro armazena valores inteiros representando, na ordem, os tipos de cada campo dos *structs* a serem armazenados no vetor de saída, de acordo com o código a seguir:
  - char → 1
  - int → 2
  - fim → 0
- `saida` é o endereço do vetor de saída, isto é, o vetor onde devem ser escritos os *structs* sem bytes de alinhamento. Considere que esse vetor é grande o suficiente para receber os dados que serão armazenados nele.

A função deve retornar a quantidade de bytes escritos no vetor `saida`.

Obs.: Considere que este programa será executado em uma máquina Pentium com sistema operacional Linux (inteiros de 32 bits em *little-endian*).