PUC-Rio – Software Básico – INF1612
Prova 2 – 20/06/06

1. Traduza as funções `dfs` e `zee` abaixo para assembly IA-32 (o assembly visto em sala), utilizando as regras usuais de alinhamento, passagem de parâmetros e retorno em C/Linux. Comente seu código.

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

   (a) (2,5 pontos)

```
typedef struct SNo no;
struct SNo
{
  int val;
  no* esq;
  no* dir;
};
int dfs(no* v)
{
  if (v == 0)
    return 0;
  else
    {
      v->val = 1;
      return (1+dfs(v->esq)+dfs(v->dir));
    }
}
```

   (b) (2,5 pontos)

```
double power (double x, double y)
double zee (int a[], int n, double exp) {
  int i;
  double acc=0.0;
  for (i=0;i<n;i++)
    acc += power (a[i], exp);
  return acc;
}
```

2. (2,5 pontos) Dada a função `cria_func`, do trabalho 2, corretamente implementada,

```
void* cria_func (void* f, int n, Parametro params[]);
```

   Forneça o código assembly equivalente ao código de máquina gerado pela `cria_func` para os seguintes casos:

   (a)
```
params[0].tipo = DOUBLE_PAR;
params[0].e_constante = 0;
params[1].tipo = DOUBLE_PAR;
params[1].e_constante = 1;
params[1].valor.v_double = 0.0;
f = cria_func(f1, 2, params);
```

   Considere que o endereço da função `f1` é `0x00237A20`.

   (b)
```
params[0].tipo = PTR_PAR;
params[0].e_constante = 0;
params[1].tipo = INT_PAR;
params[1].e_constante = 1;
```

```
params[1].valor.v_int = 24;
params[2].tipo = CHAR_PAR;
params[2].e_constante = 0;
f = cria_func(f2, 3, params);
```

Considere que o endereço da função `f2` é `0x00237F00`.

3. (2,5 pontos) Considere o programa C a seguir (as reticências na inicialização de S 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 s {
  float f;
  int i;
  char c;
  double d;
} S = {...};
int main (void) {
  dump (&S, 20);
  return 0;
}
```

Ao ser executado em uma máquina Linux IA-32, esse programa imprimiu a saída a seguir:

```
0x80495dc - 00
0x80495dd - 28
0x80495de - 80
0x80495df - c4
0x80495e0 - e9
0x80495e1 - ff
0x80495e2 - ff
0x80495e3 - ff
0x80495e4 - 5f
0x80495e5 - 00
0x80495e6 - 00
0x80495e7 - 00
0x80495e8 - 00
0x80495e9 - 00
0x80495ea - 00
0x80495eb - 00
0x80495ec - 00
0x80495ed - 00
0x80495ee - e4
0x80495ef - 3f
```

Analisando a saída e a declaração de `struct s`, diga quais foram os valores atribuídos a `S` na sua inicialização. (ATENÇÃO: o programa pede que dump imprima 20 bytes! Esse número é suficiente para mostrar `S`, mas podem ser impressos bytes a mais.) Expresse os valores sempre na base 10. **Explique** como você chegou aos valores (mostre suas contas e a posição relativa dos dados!!). Suponha que a máquina de execução é *little-endian*.