**PUC-Rio – Software Básico – INF1018**
**Prova 2 – Turma 3wb – 01/12/2016**

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++;
  }
}
double d;
union U {
  unsigned int i;
  float f;
} u;

int main() {
  u.f = -4.25;
  d = (double) u.f;
  int valor = u.i;
  dump(&d, sizeof(d));
  dump(&u, sizeof(u));
  printf("valor: %d\n", valor >> 24);
  return 0;
}
```

   Suponha que o endereço da variável `d` na memória é 0x601040, que o endereço da union `u` é 0x601050 e que a máquina de execução é *little-endian*, Linux IA-64 (usada em sala). Mostre o que esse programa irá imprimir quando executado. (Valores sem contas **NÃO** valem ponto!).

2. Considere o módulo abaixo (`arq1.c`):

```
#include <stdio.h>
extern int j;
float foo(int x, float y);

int i = 1234;
static float f = 1.5;

int main () {
  int a = 10;
  float b = foo(a, f);
  printf("%d %f\n", i + j, b);
  return 0;
}
```

   (a) (1,0 pontos) Liste todos os símbolos exportados e importados por esse módulo, ou seja, o que apareceria como D (dados exportados), T (código exportado) e U (undefined, referência externa) na saída do `nm`.

(b) (1,0 pontos) Considere agora o módulo `arq1.c` original e esse outro módulo abaixo (`arq2.c`):

```
int i = 1;
double foo (int a, double b) {
  if (a > 0)
    return b + i;
  return 0;
}
```

Se tentássemos gerar um executável composto por esses dois módulos com o comando

```
gcc -Wall -o programa arq1.c arq2.c
```

o executável seria gerado? Se não, em qual passo o `gcc` falharia (compilação de um dos módulos ou linkedição), e qual seria a causa do erro?

3. (1,0 pontos) Escreva o código *assembly* (**não é o código de máquina!**) da função que seria criada por sua `geracod` (do segundo trabalho) para o código SBF a seguir. Use *labels* para indicar o destino dos *jumps* (se for necessário).

```
function
ret? p0 $0
v0 = p0 + $1
ret? $0 v0
end
```

4. (x,x pontos) Traduza as funções `bar` e `foo` abaixo para assembly IA-64 (o assembly visto em sala), utilizando as regras usuais de alinhamento, passagem de parâmetros, salvamento de registradores e resultados em C/linux. Traduza o mais diretamente possível o código de C para assembly. (Não se preocupe em entender o que cada função faz, apenas traduza-as literalmente.)

(a) (2,0 pontos)

```
float bar(float *vf, int n) {
  double prod = 1.0;
  while (n--) {
    prod *= (double) *vf;
    vf++;
  }
  return (float) prod;
}
```

(b) (2,5 pontos)

```
struct X {
  int type;
  double val;
  struct X *prox;
};
int boo(double d, double *pd);

float foo(struct X *px) {
  float local[5];
  double res;
  int i;
  while (px) {
    i = boo(px->val, &res);
    local[i] = (float) res;
    px = px->prox;
  }
  return bar(local, 5);
}
```