PUC-Rio – Software Básico – INF1612
Prova 2 – 28/11/02

1. Liste todos os símbolos exportados e importados pelo módulo abaixo, ou seja, os que aparecerão nas listas de importação e exportação do módulo objeto gerado pelo compilador.

```
extern int a;
static int b = 2;
float f = 1.0;

void fazalgo (int *p, int q);

static void boba2 (int p) {
    a = 1;
    fazalgo(&b, p);
}

void boba1 (int *p1, int p2) {
    fazalgo(p1,p2);
    boba2(p2);
}
```

2. Faça uma função em assembly que receba como parâmetro um string *null-terminated* UTF-8 (uma cadeia de caracteres UTF-8 terminada com o byte 00h) e retorne um inteiro com o número de caracteres contidos nesta string. A função deve poder ser chamada de C e seu protótipo deve ser:

```
int strlenUTF8(char *utf8);
```

obs: o formato UTF-8 é um formato de tamanho variável. Cada caracter tem tamanho mínimo de 8 bits, porém, se o código necessitar de mais espaço para ser representado, o formato UTF-8 pode utilizar mais de um byte. Abaixo é apresentada a faixa dos números e o número de bytes necessários para representar cada faixa do código UNICODE (onde x é o valor real do bit):

Código UNICODE	Representação UTF-8
U+0000 - U+007F	0xxxxxxx
U+0080 - U+07FF	110xxxxx 10xxxxxx
U+0800 - U+FFFF	1110xxxx 10xxxxxx 10xxxxxx
U+10000 - U+10FFFF	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx

3. Considere o programa C a seguir:

```
#include <stdio.h>

void dump (void *p, int n) {
    unsigned char *p1 = p;
    while (n--) {
        printf("%p - %02x\n", p1, *p1);
        p1++;
    }
}

struct sbobo {
    char c;
    double d;
    short s;
};

int main(void) {
    struct sbobo bobo[2] = {{'z', -31.625, -1023},
                           {'Z', 0.375, 2048}};
    dump (&bobo, 2*sizeof(struct sbobo));
    return 0;
}
```

Diga o que este programa irá imprimir quando executado, *explicando* como você chegou aos valores exibidos (mostre suas contas! e outras infos usadas para chegar ao resultado!). Suponha que a máquina de execução é little-endian e que o endereço de bobo seja 0x2A000000. Coloque 'LL' nos bytes que contiverem lixo.

obs: O código ASCII de 'A' é 65 e de 'a' é 97.

4. Considere o módulo C a seguir:

```
void avisa ();

typedef struct {
    int soma;
    char atrapalha;
} tstruct;

int bembobaebemfeia (double d1, int nums[], int i) {
    tstruct slocal = {0, 'a'};
    nums[i] = i;
    slocal.soma += nums[i];
    avisa();
    return slocal.soma;
}
```

“Compile” a função `bembobaebemfeia` para assembly, ou seja, escreva o código equivalente ao que o compilador C geraria em assembly. As variáveis locais *devem necessariamente* ser alocadas na pilha.