PUC-Rio – Software Básico – INF1018
Prova 2 – 25/06/2015 – Turma 3WA

1. (1,5 pontos) Considere o módulo `usacoro.c` abaixo:

```
#include <stdio.h>
#include "corotinas.h"
#include "meusvalores.h"

int chamadasTotais = 100;
static Coroutine c;

static int body(int qtos) {
  int x = 0;
  coro_yield(0);
  while ((x++)<qtos) {
    coro_yield(indices[x]);
  }
  return -1;
}

int criacoro(int qtos) {
  if (chamadasTotais-- == 0) return -1;
  c = coro_create("coro1", body);
  coro_resume(c, qtos);
  return 1;
}

int usacoro() {
  return coro_resume(c, 0);
}
```

   Liste quais símbolos do módulo objeto `usacoro.o` apareceriam como **D** (símbolo na área de dados, exportado por usacoro.o), **T** (símbolo na área de código, exportado por usacoro.o) e **U** (símbolo indefinido, referência externa) na saída do programa `nm`.

2. Traduza as funções `boo` e `foo` abaixo para assembly IA-32 do gcc/Linux (visto em sala), utilizando as regras básicas de passagem de parâmetros, retorno de resultado e uso de registradores em C. (Não se preocupe se você não entender o que as funções fazem, apenas traduza-as literalmente.)

   (a) (2,5 pontos)

```
int f (int num);

int boo (double a[], int tamanho) {
  int indice = f(tamanho);
  return (int) a[indice];
}
```

   (b) (2,5 pontos)

```
#define MAX 5
double bar (double* vals, int tam);
```

```
double foo (float *vals, int tam) {
  double meusvals[MAX];
  double *pmv = meusvals;
  int i = 0;
  for (i=0;i<tam;i++) {
    *pmv = (double)*vals;
    pmv++; vals++;
  }
  return bar(meusvals, tam);
}
}
```

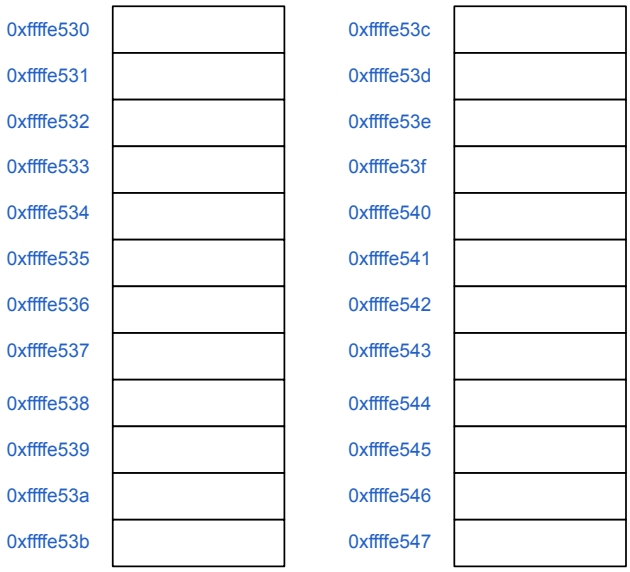
3. (2,5 pontos) Considere o trecho de código C

```
int fun1(double d1, double *d2, float f);
int fun2(...) {
  int i; double num = 1.0;
  i = fun1(-0.625,&num,1536.0);
  ...
}
```

Suponha que, na geração do executável, a instrução de chamada a `fun1` ficou no endereço `0x80483b1`:

```
80483b1:       e8 be ff ff ff          call   8048374 <fun1>
80483b6:       89 45 fc                mov    %eax,0xfffffffc(%ebp)
```

Suponha ainda que, em determinada execução do trecho mostrado, no momento exatamente antes da execução da instrução `call fun1`, `num` esteja no endereço `0xfffff55c` e `i` no endereço `0xfffff564`. Suponha também que, no momento em que o controle chegar ao label `fun1` (ou seja, imediatamente após a CPU executar `call fun1` e antes de começar a executar o código da função `fun1`), o valor de `%esp` seja `0xffffe530`. Preencha a figura abaixo com o conteúdo, em hexadecimal, de cada byte da memória nesse momento de chegada ao label `fun1`, para os bytes cujo valor você tem como conhecer com as informações dadas.



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