```c
#include <stdio.h>
#include <stdlib.h>

/**
 * Este é um código de demonstração para mostrar aos alunos os segmentos de
 * memória onde as variáveis de nosso programa podem ser armazenadas. Não estão
 * incluídas variáveis com modificador "register".
 *
 * Prof. Abrantes Araújo Silva Filho
 */

// Variáveis na DATA RW, escopo global:
int a = 10;
double b = 1.23;
static float c = (float) 4.56;

// Variáveis na DATA RO, escopo global:
double const pi = 3.14;
int const zero = 0;

// Variáveis na BSS, escopo global:
int d;
float e = (float) 0;
static double f = 0;

int main (int argc, [[maybe_unused]] char *argv[argc])
{
    // Variáveis na DATA RW, escopo local:
    static int g = 1;
    static float h = (float) 7.89;

    // Variáveis na DATA RO, escopo local:
    static int const i = 5;
    static float const zerof = (float) 0.0;

    // Variáveis na BSS, escopo local:
    static int j;
    static double k = 0.0;

    // Variáveis no STACK:
    int l;
    int m = 0;
    int numeros[10];
    int mais_numeros[10] = {0};

    // Variáveis no STACK/HEAP:
    int *p1 = malloc(sizeof(*p1) * 10); // p1 está no STACK
                                        // a área apontada por p1 está na HEAP

    float *f1 = malloc(sizeof(*f1) * 10); // f1 está no STACK
                                          // a área apontada por f1 está na HEAP

    if (!p1 || !f1)             // Verifica se a alocação deu errado
    {                           // free(NULL) é seguro: libera se estiver alocado
        free(p1);               // Libera memória apontada por p1 na HEAP
        free(f1);               // Libera memória apontada por f1 na HEAP
```

```
        return EXIT_FAILURE;
    }

    // "Usa" todas as variáveis: isso força o compilador a dar a cada uma um
    // lugar próprio na memória.
    printf(".data   : %p %p %p %p %p\n", (void*)&a, (void*)&b, (void*)&c,
           (void*)&g, (void*)&h);
    printf(".rodata : %p %p %p %p\n", (void*)&pi, (void*)&zero, (void*)&i,
           (void*)&zerof);
    printf(".bss    : %p %p %p %p %p\n", (void*)&d, (void*)&e, (void*)&f,
           (void*)&j, (void*)&k);
    printf("stack   : %p %p %p %p %p %p\n", (void*)&l, (void*)&m,
           (void*)numeros, (void*)mais_numeros, (void*)&p1, (void*)&f1);
    printf("heap    : %p %p\n", (void*)p1, (void*)f1);

    free(p1);      // Libera memória apontada por p1 na HEAP
    free(f1);      // Libera memória apontada por f1 na HEAP
    p1 = NULL;     // Anula o ponteiro (evita dangling pointer)
    f1 = NULL;     // Anula o ponteiro (evita dangling pointer)

    return EXIT_SUCCESS;
}
```