Alocação Dinâmica de Memória

Algoritmos e Estruturas de Dados - 2025/02

Última atualização: February 19, 2026

> **Aviso**
>
> Em várias partes dessa atividade iremos rodar código. Acesse o exercício "Alocação Dinâmica de Memória" no PrairieLearn e use o workspace do exercício nessas atividades.

Leia com atenção o seguinte programa.

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

#define N 12

int *aloca_vetor(int n) {
    int *vetor = malloc(n * sizeof(int));
    return vetor;
}

void atribui(int *vetor, int n) {
    for (int i = 0; i <= n; i++) {
        vetor[i] = i;
    }
}

int main(int argc, char *argv[]) {
    int *vetor = aloca_vetor(N);
    int i;

    atribui(vetor, N);

    for (i = 0; i <= N; i++) {
        printf("Elemento %d: %d\n", i+1, vetor[i]);
    }

    return 0;
}
```

Vamos começar analisando o código do programa **antes de rodá-lo**.

Exercício: Analisando seu código-fonte, o que este programa faz?

Exercício: Na execução deste programa, o que acontece se `malloc` falhar?

Exercício: O seu programa libera toda memória que aloca? Se não, aponte onde ele deveria fazer isto. (Não lembra? Olhe de novo os slides :)

Agora que entendemos um pouco melhor o programa e percebemos que ele tem vários erros, vamos executá-lo para ver o que acontece :) Salve o conteúdo do programa como `ex1.c` e compile-o com o comando abaixo.

```
$ gcc -Wall -std=c99 -pedantic -Og ex1.c -o ex1
```

> **Dica**
>
> O terminal do VSCode é muito bom e podemos usá-lo para rodar todos os comandos dessa atividade. Como a saída de vários comandos é meio grande, pode ser uma boa maximizá-lo quando formos ler mensagens de erro.

Exercício: Execute o programa. Ocorreu algum problema durante a execução?

Exercício: Agora tente ir mudando os valores de `N`. O comportamento de seu programa muda conforme N muda? Começe com `N=10` e vá incrementando de um em um. Coloque abaixo suas observações.

> **Aviso**
>
> Em C é comum programas com problemas funcionarem de maneira imprevisível. Essencialmente, coisas erradas acontecem, mas elas podem não ser graves o suficiente para que o programa seja finalizado pelo sistema. Pequenas mudanças no código podem fazer um programa aparentemente OK quebrar.

Exercício: Existem três problemas no código. O primeiro (`vetor` não é desalocado) já identificamos no exercicío anterior. Você consegue identificar os outros dois?

Exercício: Corrija os erros apontados na questão anterior e salve o programa em um arquivo *ex1-certo.c*. Refaça o teste com vários valores de `N` e verifique se agora o programa roda sem problemas.

Ferramentas de verificação de memória

Identificar erros lendo código fonte é demorado e bastante difícil conforme nossos programas ficam maiores e mais complexos. Para poder identificar mais facilmente problemas relativos a memória, iremos utilizar uma ferramenta chamada **Address Sanitizer**. Ele é um detector de mau uso de memória que roda seu programa em cima de um ambiente modificado e aponta os seguintes erros:

1. memória alocada e não liberada
2. acessos (leituras e escritas) a posições de memória não alocada ou inválidas

Para que os problemas encontrados sejam mais facilmente identificados, iremos passar a compilar utilizando a flag `-g` e a incluir o **Address Sanitizer** no nosso programa usando `-fsanitize=address`.

```
$ gcc -Og -g -Wall -std=c99 ex1.c -o ex1 -fsanitize=address
$ gcc -Og -g -Wall -std=c99 ex1-certo.c -o ex1-certo -fsanitize=address
```

Exercício: Rode o seu programa com `./ex1`. Leia rapidamente a saída do programa e anote abaixo coisas que observou.

Nas próximas atividades iremos mostrar como encontrar os três erros identificados pela leitura do código na saída do *Address Sanitizer*. Nossa estratégia será **ler a saída de cima para baixo e consertar sempre o primeiro erro encontrado**.

Erro 1

O primeiro erro encontrado é

```
==1290==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x606000000058 at pc 0x5645bfb73203 bp 0x7
WRITE of size 4 at 0x606000000058 thread T0
    #0 0x5645bfb73202 in atribui /home/coder/project/ex1.c:13
    #1 0x5645bfb73223 in main /home/coder/project/ex1.c:21
    #2 0x7f1f461bd1c9 in __libc_start_call_main ../sysdeps/nptl/libc_start_call_main.h:58
    #3 0x7f1f461bd284 in __libc_start_main_impl ../csu/libc-start.c:360
    #4 0x5645bfb730e0 in _start (/home/coder/project/ex1+0x10e0)
```

Exercício: Em qual linha o erro ocorre? Para facilitar, sublinhe ou grife a mensagem de erro. O que a mensagem acima significa?

Exercício: Corrija o erro e rode novamente o programa. Verifique que o erro acima não ocorre mais e que o primeiro erro mostrado mudou antes de prosseguir.

Erro 2

Os erros do *Address Sanitizer* são mostrados quando acontecem, intercalados com os `printf` do nosso programa. O segundo erro está mostrado abaixo e ocorre logo antes de lermos da linha "Elemento 13" na saída.

```
==1746==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x606000000058 at pc 0x559e63a32274 bp 0x7
READ of size 4 at 0x606000000058 thread T0
    #0 0x559e63a32273 in main /home/coder/project/ex1.c:24
    #1 0x7f4a8df131c9 in __libc_start_call_main ../sysdeps/nptl/libc_start_call_main.h:58
    #2 0x7f4a8df13284 in __libc_start_main_impl ../csu/libc-start.c:360
    #3 0x559e63a320e0 in _start (/home/coder/project/ex1+0x10e0)
```

Exercício: Em qual linha o erro ocorre? Para facilitar, sublinhe ou grife a mensagem de erro. O que a mensagem acima significa?

Exercício: Corrija o erro e rode novamente o programa. Verifique que o erro acima não ocorre mais e que o primeiro erro mostrado mudou antes de prosseguir.

Erro 3

Agora temos somente o erro abaixo na saída:

```
==1999==ERROR: LeakSanitizer: detected memory leaks

Direct leak of 56 byte(s) in 1 object(s) allocated from:
    #0 0x7fdb357999cf in __interceptor_malloc ../../../../src/libsanitizer/asan/asan_malloc_linux.cpp:69
    #1 0x55a3fef721b8 in aloca_vetor /home/coder/project/ex1.c:7

SUMMARY: AddressSanitizer: 56 byte(s) leaked in 1 allocation(s).
```

Exercício: Ela mostra algum problema? Se sim, qual linha de código é apontada? Para facilitar, sublinhe ou grife a mensagem de erro .Qual é o problema diagnosticado por este aviso?

Exercício: Corrija o erro e rode novamente o programa. Verifique que nenhum erro ocorre mais.

Para entregar

A prática mais completa dessa aula está disponível no PrairieLearn e envolverá implementar funções que lidam com strings. Os objetivos são

- revisar uso de strings e sua codificação na memória
- criar funções que aloquem novos objetos na memória e os retorne
- determinar o tamanho correto a ser alocado para cada operação
- usar o AddressSanitizer para checar se são feitos acessos indevidos à memória
- usar o AddressSanitizer para checar se todo malloc tem um free correspondente. Note que nem sempre iremos chamar `free` na mesma função que `malloc`. É muito comum alocar memória em uma função e devolver esses dados para que outras funções usem e eventualmente liberem a memória quando acabarem.