Alocação Dinâmica de Memória

Linguagem C

Wesley Dias Maciel

wesleydiasmaciel@gmail.com

Alocação Dinâmica de Memória

Objetivos:

- Entender o conceito de alocação dinâmica de memória.
- Aprender a usar as funções **malloc**, **calloc**, **realloc** e **free** em C.
- Compreender a importância da alocação dinâmica de memória em programas C.

Introdução:

- A alocação dinâmica de memória é uma técnica essencial na programação em C.
- Ela permite que os programas reservem e liberem memória durante a execução, o que é útil quando não sabemos com antecedência o tamanho exato que uma estrutura de dados precisará.
- As funções malloc, calloc, realloc e free da biblioteca stdlib.h ajudam gerenciar a alocação dinâmica de memória em linguagem C.

Função malloc (Memory Allocation):

- A função malloc em C é uma das principais funções usadas para alocar memória dinamicamente durante a execução de um programa.
- Ela pertence à biblioteca padrão stdlib.h e é usada para reservar um bloco de memória contígua de tamanho especificado e retorna um ponteiro para o início desse bloco de memória.
- Esse bloco alocado pode ser usado para armazenar dados durante a execução do programa.
- Sintaxe:

void *malloc(num_elementos * sizeof(tipo));

num_elementos: é o número de elementos que você deseja alocar na memória.

sizeof(tipo): é o tamanho, em bytes, de cada elemento.

- A função malloc aloca memória e retorna um ponteiro void * para o início do bloco de memória alocado.
    - É importante observar que o ponteiro retornado é do tipo void *, o que significa que ele precisa ser convertido para o tipo de ponteiro apropriado antes de ser usado para armazenar dados.

OBS:

- É importante verificar se a alocação de memória com malloc foi bem-sucedida.
- Se a alocação não puder ser concluída (por falta de memória disponível, por exemplo), a função malloc retornará um ponteiro nulo (NULL).

- Portanto, é uma prática fortemente recomendada verificar se o ponteiro retornado por malloc não é nulo antes de começar a usá-lo.

Exemplo:

```c
#include <stdio.h>
#include <stdlib.h> //malloc(), free(), srand () e rand ().
#include <time.h> //time().

int main() {
    int *array, n, i;

    srand(time (NULL)); //Inicia a geração de inteiros aleatórios com uma nova semente.

    printf("Informe o tamanho do array: ");
    scanf("%d", &n);

    // Alocação dinâmica de memória usando malloc:
    array = (int *) malloc(n * sizeof(int));

    if (array == NULL) {
        printf("Falha na alocação de memória.\n");
        return 1;
    }

    // Agora você pode usar o 'array' para armazenar dados dinamicamente.
    for(i = 0; i < n; i++)
        array[i] = rand () % 100; //Número aleatório de 0 a 99.

    for(i = 0; i < n; i++)
        printf("\n%d", array[i]);

    // Liberar a memória alocada quando não for mais necessária
    free(array);

    return 0;
}
```

```
Informe o tamanho do array: 5

80
24
97
50
26
```

- Em resumo, a função malloc é usada para alocar dinamicamente memória em C, permitindo que você reserve um bloco de memória de tamanho especificado durante a execução do programa.
- É uma ferramenta fundamental para trabalhar com estruturas de dados dinâmicas e evitar a necessidade de alocar memória estática.
- No entanto, é essencial gerenciar adequadamente a memória alocada e liberá-la usando a função free quando não for mais necessária para evitar vazamentos de memória.

Função calloc (Contiguous Allocation):

- A função calloc em linguagem C é usada para alocar memória dinamicamente, assim como a função malloc, mas ela tem uma característica adicional: ela inicializa todos os bytes alocados com zeros.
- A função calloc faz parte da biblioteca padrão stdlib.h e é útil quando você deseja garantir que a memória alocada seja inicializada com valores zero.
- Sintaxe:

  void *calloc(num_elementos, sizeof(tipo));

  num_elementos: é o número de elementos que você deseja alocar.

  sizeof(tipo): é o tamanho, em bytes, de cada elemento.

- A função calloc aloca memória para um bloco de tamanho (num_elementos * (sizeof(tipo))) e retorna um ponteiro void * para o início desse bloco.
- Todos os bytes desse bloco são inicializados com zero.

OBS:

- A principal diferença entre calloc e malloc é que calloc inicializa automaticamente a memória alocada com zeros, enquanto malloc não faz isso.
- Portanto, calloc é especialmente útil quando você deseja começar com uma memória limpa e não deseja se preocupar em inicializar manualmente os dados alocados.

Exemplo:

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

int main() {
    int *array, n, i;

    printf("Digite o tamanho do array: ");
    scanf("%d", &n);

    // Alocação dinâmica de memória usando calloc
    array = (int *)calloc(n, sizeof(int));
```

```c
    if (array == NULL) {
        printf("Falha na alocação de memória.\n");
        return 1;
    }

    // Agora você pode usar 'array' para armazenar dados dinamicamente
    // Todos os elementos são inicializados com zero
    for(i = 0; i < n; i++)
        printf("\n%d", array[i]);

    // Liberar a memória alocada com calloc quando não for mais
necessária
    free(array);

    return 0;
}
```

```
Digite o tamanho do array: 5

0
0
0
0
0
```

- Este exemplo aloca dinamicamente um array de inteiros e inicializa todos os elementos com zero usando a função calloc.
- Lembre-se de verificar se a alocação foi bem-sucedida e de liberar a memória alocada com free quando não for mais necessária, da mesma forma que com malloc.

- Resumindo, a função calloc é usada para alocar memória dinamicamente e inicializar todos os bytes alocados com zeros, tornando-a uma escolha conveniente quando você deseja começar com uma memória limpa.
- A escolha entre malloc e calloc em C depende das necessidades específicas do seu programa e da inicialização dos dados que você deseja alocar dinamicamente.
- Algumas diretrizes para quando usar calloc em vez de malloc:
    - Inicialização com Zeros: use calloc quando você deseja que os blocos de memória alocados sejam inicializados com zeros.
    - Por exemplo: ao alocar matrizes ou estruturas que requerem inicialização com zeros, calloc é uma escolha conveniente.

Exemplo:

```c
struct Pessoa {
    char nome[50];
```

```
    int idade;
};

struct Pessoa *p = (struct Pessoa *) calloc(1, sizeof(struct Pessoa)); //
Inicializa com zeros
```

OBS:

- Prevenção de Bugs: usar calloc pode ajudar a evitar bugs sutis que resultam de acessar valores não inicializados.
    - Se você não inicializar seus dados manualmente após a alocação, calloc garantirá que eles sejam definidos para zero.
- No entanto, é importante observar que calloc pode ser um pouco mais lento do que malloc simplesmente porque envolve a etapa adicional de inicialização.
    - Portanto, se você não precisa de inicialização com zeros, malloc pode ser uma escolha mais eficiente em termos de desempenho.

- Em resumo, use calloc quando quiser alocar memória dinâmica e garantir que ela seja inicializada com zeros. Caso contrário, se a inicialização não for necessária, malloc pode ser mais apropriado para melhor desempenho.

Função realloc.

- A função realloc em linguagem C é usada para redimensionar dinamicamente um bloco de memória previamente alocado com malloc, calloc ou uma função similar de alocação dinâmica de memória.
- Ela pertence à biblioteca padrão stdlib.h e é útil quando você precisa aumentar ou diminuir o tamanho da memória alocada durante a execução de um programa.

Sintaxe:

void *realloc(void *pt, num_elementos * sizeof(tipo));

pt: é um ponteiro para o bloco de memória previamente alocado que você deseja redimensionar.

num_elementos: é o novo número de elementos que você deseja alocar.

sizeof(tipo): é o tamanho, em bytes, de cada elemento.

- A função realloc retorna um ponteiro void * para o início do bloco de memória redimensionado.
    - É importante observar que o ponteiro retornado pode ser diferente do ponteiro original.
    - Portanto, você deve atualizar seu ponteiro original com o valor retornado por realloc.

Algumas considerações importantes ao usar realloc:

- Verificação de Alocação Bem-Sucedida: assim como com malloc e calloc, você deve verificar se a realocação de memória com realloc foi bem-sucedida.
- Se a realocação não puder ser concluída (por falta de memória disponível, por exemplo), a função realloc retornará um ponteiro nulo (NULL).

- Cuidado com a Memória Original: após a realocação, a memória original não é mais válida. Portanto, você não deve acessar ou modificar a memória original após chamar realloc.

- Redimensionamento para um Tamanho Maior: se você usar realloc para aumentar o tamanho de um bloco de memória, os dados previamente armazenados no bloco original serão mantidos na nova memória, contanto que a nova memória inclua a área original.

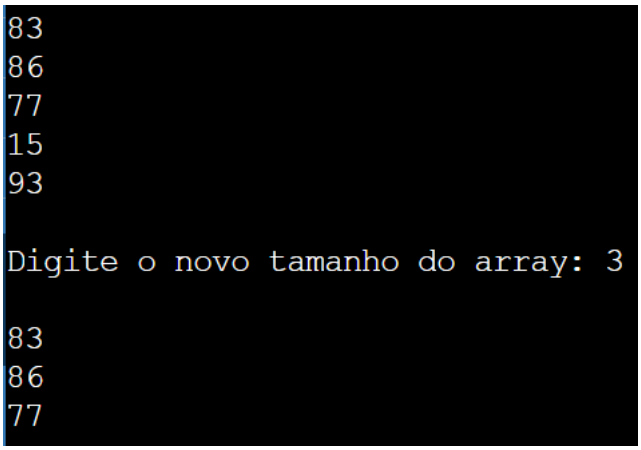
- Redimensionamento para um Tamanho Menor: se você usar realloc para reduzir o tamanho de um bloco de memória, a função retornará um ponteiro para o novo bloco de memória com o tamanho especificado. No entanto, os dados além do novo tamanho serão perdidos.

Exemplo:

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

int main() {
    int *array, n = 5, novo_n, i;

    // Alocação dinâmica de memória usando malloc
    array = (int *) malloc(n * sizeof(int));

    if (array == NULL) {
        printf("Falha na alocação de memória.\n");
        return 1;
    }

    // Agora você pode usar o 'array' para armazenar dados dinamicamente.
    for(i = 0; i < n; i++)
        array[i] = rand () % 100; //Número aleatório de 0 a 99.

    for(i = 0; i < n; i++)
        printf("\n%d", array[i]);
```

```c
    printf("\n\nDigite o novo tamanho do array: ");
    scanf("%d", &novo_n);

    // Realocar o array para o novo tamanho
    array = (int *) realloc(array, novo_n * sizeof(int));

    if (array == NULL) {
        printf("Falha na realocação de memória.\n");
        return 1;
    }

    // Agora você pode usar 'array' com o novo tamanho
    for(i = 0; i < novo_n; i++)
        printf("\n%d", array[i]);

    // Liberar a memória alocada com realloc quando não for mais
necessária
    free(array);

    return 0;
}
```

```
83
86
77
15
93

Digite o novo tamanho do array: 7

83
86
77
15
93
0
0
```



- Neste exemplo, começamos alocando um array de inteiros com malloc, e então usamos realloc para redimensioná-lo para um novo tamanho especificado pelo usuário.
- Como sempre, é importante verificar se a alocação ou realocação foi bem-sucedida e liberar a memória alocada quando não for mais necessária para evitar vazamentos de memória.

Liberação de Memória com a Função free:

- A função free em linguagem C é usada para liberar a memória previamente alocada dinamicamente por meio das funções malloc, calloc, realloc, ou outras funções de alocação dinâmica.
- A função free faz parte da biblioteca padrão stdlib.h.

Sintaxe:

void free(void *pt);

pt: é um ponteiro para o bloco de memória que você deseja liberar.

- A função free auxilia no gerenciamento adequado da memória alocada dinamicamente em um programa em linguagem C.
- Quando você aloca memória dinamicamente usando malloc, calloc ou realloc, é responsabilidade do programador garantir que essa memória seja liberada quando não for mais necessária.
- Não liberar memória alocada dinamicamente pode levar a vazamentos de memória, onde o programa consome continuamente recursos de memória sem liberá-los, o que pode causar problemas sérios, incluindo o esgotamento de memória do sistema.

- Alguns pontos importante:
    - Evitar Vazamentos de Memória: o uso adequado de free é fundamental para evitar vazamentos de memória.

        - Sempre que você alocar memória dinamicamente, deve planejar liberá-la quando não for mais necessária.
    - Liberar a Memória no Momento Adequado: libere a memória alocada dinamicamente assim que você não precisar mais dela.
        - Isso pode ser no final de uma função ou em algum ponto específico do programa, dependendo da lógica do seu código.
    - Não Acesse Memória Desalocada: após chamar free, a memória alocada dinamicamente não é mais válida, e qualquer acesso a essa memória pode resultar em comportamento indefinido.
        - Portanto, certifique-se de não acessar ou modificar a memória após a chamada de free.
    - Evitar Liberar a Memória Duas Vezes: chamar free duas vezes para o mesmo bloco de memória pode causar falhas de segmentação e erros no programa.
        - Portanto, mantenha o controle de quais blocos de memória já foram liberados.

Exemplo:

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

int main() {
    int *array, n = 5, i;

    // Alocação dinâmica de memória usando malloc:
    array = (int *) malloc(n * sizeof(int));

    if (array == NULL) {
        printf("Falha na alocação de memória.\n");
        return 1;
    }

    // Usar 'array' para armazenar dados dinamicamente
    for(i = 0; i < n; i++)
        array[i] = rand () % 100; //Número aleatório de 0 a 99.

    for(i = 0; i < n; i++)
        printf("\n%d", array[i]);

    // Liberar a memória alocada com malloc quando não for mais
necessária
    free(array);

    return 0;
}
```

- Neste exemplo, depois de usar a memória alocada dinamicamente, chamamos free(array) para liberar essa memória quando ela não é mais necessária.

- Em resumo, a função free é fundamental para o gerenciamento adequado de memória em programas C que fazem uso de alocação dinâmica.
    - O uso correto de free evita vazamentos de memória e ajuda a manter o programa eficiente e confiável.

Exercícios

1) O exemplo abaixo usa malloc para alocar dinamicamente um array de inteiros de acordo com o tamanho especificado pelo usuário. Ele verifica se a alocação foi bem-sucedida, assegurando que o ponteiro retornado por malloc não seja nulo. Além disso, a memória alocada é liberada com free quando não for mais necessária para evitar vazamentos de memória.

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

int main() {
    int *array, n;

    printf("Digite o tamanho do array: ");
    scanf("%d", &n);

    // Alocação dinâmica de memória usando malloc
    array = (int *) malloc(n * sizeof(int));

    if (array == NULL) {
        printf("Falha na alocação de memória.\n");
        return 1;
    }

    // Preencher o array
    for (int i = 0; i < n; i++) {
        array[i] = i * 10;
    }

    // Imprimir o array
    printf("Array: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", array[i]);
    }

    // Liberar a memória alocada com malloc
    free(array);
```

```
    return 0;
}
```

Altere o programa para que os trechos de código para alocação de memória, preenchimento do vetor e impressão do vetor seja realizado por funções.

2) O exemplo abaixo usa calloc para alocar dinamicamente um array de inteiros e inicializar todos os elementos com zero. A abordagem para verificar a alocação e liberar a memória é a mesma do exercício anterior.

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

int main() {
    int *array, n;

    printf("Digite o tamanho do array: ");
    scanf("%d", &n);

    // Alocação dinâmica de memória usando calloc
    array = (int *) calloc(n, sizeof(int));

    if (array == NULL) {
        printf("Falha na alocação de memória.\n");
        return 1;
    }

    // Imprimir o array
    printf("Array: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", array[i]);
    }

    // Liberar a memória alocada com calloc
    free(array);

    return 0;
}
```

Altere o exemplo para que seja um vetor de estrutura Pessoa. A estrutura Pessoa deve possuir os campos: CPF, nome e salário. Os trechos de código para alocação de memória, preenchimento do vetor e impressão do vetor devem ser realizados por funções.

3) O exemplo abaixo usa realloc para redimensionar dinamicamente um array alocado anteriormente. Observe que a realocação pode falhar, então é importante verificar se o

ponteiro retornado por realloc não é nulo após a operação. Igualmente importante é a liberação de memória ao final do algoritmo.

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

int main() {
    int *array, n;

    printf("Digite o tamanho inicial do array: ");
    scanf("%d", &n);

    // Alocação dinâmica de memória usando malloc
    array = (int *) malloc(n * sizeof(int));

    if (array == NULL) {
        printf("Falha na alocação de memória.\n");
        return 1;
    }

    // Preencher o array
    for (int i = 0; i < n; i++) {
        array[i] = i * 10;
    }

    // Imprimir o array
    printf("Array (antes da realocação): ");
    for (int i = 0; i < n; i++) {
        printf("%d ", array[i]);
    }

    // Realocar o array para um tamanho maior
    int novo_tamanho = n + 5;
    array = (int *) realloc(array, novo_tamanho * sizeof(int));

    if (array == NULL) {
        printf("Falha na realocação de memória.\n");
        return 1;
    }

    // Preencher os elementos adicionais
    for (int i = n; i < novo_tamanho; i++) {
        array[i] = i * 10;
    }

    // Imprimir o array após a realocação
    printf("\nArray (após a realocação): ");
```

```
    for (int i = 0; i < novo_tamanho; i++) {
        printf("%d ", array[i]);
    }

    // Liberar a memória alocada
    free(array);

    return 0;
}
```

Altere o exemplo para que seja um vetor de estrutura Veículo. A estrutura Veículo deve possuir os campos: número do chassi, marca, modelo e preço. Os trechos de código para alocação de memória, preenchimento do vetor, impressão do vetor e realocação de memória devem ser realizados por funções.