

Prática: Gerenciamento de Memória Dinâmica em C - Alocação de Vetores

Disciplina: Programação em C

1. Introdução

1.1 Objetivos

- Compreender os conceitos fundamentais de alocação dinâmica de memória em C
- Implementar e gerenciar vetores dinâmicos
- Analisar o impacto do uso de memória dinâmica no sistema
- Identificar e evitar vazamentos de memória (memory leaks)
- Praticar boas práticas de gerenciamento de memória

1.2 Pré-requisitos

- Conhecimento básico da sintaxe da linguagem C
- Familiaridade com ponteiros em C
- Acesso a um ambiente de desenvolvimento C
- Conhecimento básico do Gerenciador de Tarefas

2. Conceitos Fundamentais

2.1 Memória em C

Em C, existem três tipos principais de alocação de memória:

1. Alocação Estática

- Ocorre em tempo de compilação
- Tamanho fixo e imutável
- Exemplo: `int array[100];`

2. Alocação Automática (Stack)

- Ocorre em tempo de execução
- Gerenciada automaticamente
- Exemplo: variáveis locais de função

3. Alocação Dinâmica (Heap)

- Ocorre em tempo de execução
- Gerenciada manualmente pelo programador
- Exemplo: `malloc()`, `calloc()`, `realloc()`

2.2 Funções de Alocação Dinâmica

Alocação Lazy (Preguiçosa)

- Quando `malloc()` é chamado, o sistema operacional não aloca imediatamente a memória física
- Apenas reserva um espaço no espaço de endereçamento virtual do processo
- A memória física real só é alocada quando o programa acessa essa memória pela primeira vez
- Isso é uma otimização do sistema operacional para:
 - Evitar alocar memória física que pode nunca ser usada
 - Permitir melhor gerenciamento da memória física
 - Reduzir o overhead de alocação inicial

malloc()

```
void* malloc(size_t size);
```

- Aloca um bloco de memória do tamanho especificado
- Retorna um ponteiro para o início do bloco
- Retorna NULL se a alocação falhar
- Não inicializa a memória alocada

free()

```
void free(void* ptr);
```

- Libera a memória alocada dinamicamente
- Deve ser chamada para cada bloco alocado
- Não faz nada se ptr for NULL
- Não deve ser chamada duas vezes para o mesmo ponteiro

3. Alocação de Vetores Dinâmicos

3.1 Alocação Básica

```
int *vetor = (int *)malloc(tamanho * sizeof(int));
```

3.2 Desalocação

```
free(vetor);  
vetor = NULL;
```

4. Boas Práticas

1. Sempre verifique o retorno das funções de alocação

```
if (ptr == NULL) {  
    // Tratar erro  
}
```

2. Use sizeof() para calcular tamanhos

```
int *ptr = malloc(n * sizeof(int));
```

3. Defina ponteiros como NULL após liberar

```
free(ptr);  
ptr = NULL;
```

4. Evite vazamentos de memória

- Mantenha controle de todas as alocações
- Libere a memória quando não for mais necessária
- Use ferramentas como Valgrind para detectar vazamentos

5. Exercícios Práticos

Exercício 1: Implementação Básica

Complete as funções `alocarVetor()` e `desalocarVetor()` no código fornecido.

Exercício 2: Operações com Vetores

Implemente funções para:

- Inserir elementos
- Remover elementos
- Buscar elementos
- Ordenar elementos

Exercício 3: Redimensionamento

Implemente uma função que redimensione um vetor dinâmico.

Exercício 4: Análise de Memória

1. Execute o programa com diferentes tamanhos de vetor
2. Observe o uso de memória no Gerenciador de Tarefas
3. Documente suas observações

Exercício 5 (Opcional): Demonstração da Alocação Lazy

1. Modifique a função `alocarVetor()` para incluir uma opção de inicialização:

```
void alocaVetor(int n, bool inicializar) {
    vetor_atual = (int *)malloc(n * sizeof(int));

    if (vetor_atual == NULL) {
        printf("Erro: Falha ao alocar vetor\n");
        return;
    }

    if (inicializar) {
        for(int i = 0; i < n; i++) {
            vetor_atual[i] = 0; // Isso forçará a alocação física
        }
    }

    tamanho_vetor_atual = n;
}
```

2. Execute o programa com um vetor grande (ex: 1.000.000 elementos)
3. Compare o uso de memória no Gerenciador de Tarefas:
 - Sem inicialização: apenas alocação virtual
 - Com inicialização: alocação física real
4. Documente as diferenças observadas no uso de memória

6. Código Base

```

#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

int *vetor_atual = NULL;
int tamanho_vetor_atual = 0;

void alocarVetor(int n, bool inicializar) {
    // Aloca o vetor
    vetor_atual = (int *)malloc(n * sizeof(int));

    if (vetor_atual == NULL) {
        printf("Erro: Falha ao alocar vetor\n");
        return;
    }

    if (inicializar) {
        for(int i = 0; i < n; i++) {
            vetor_atual[i] = 0; // Isso forçará a alocação física
        }
    }

    tamanho_vetor_atual = n;
}

void desalocarVetor() {
    if (vetor_atual == NULL) {
        return;
    }

    // Libera o vetor
    free(vetor_atual);

    // Reseta as variáveis
    vetor_atual = NULL;
    tamanho_vetor_atual = 0;
}

void redimensionarVetor(int novo_tamanho) {
    if (vetor_atual == NULL) {
        printf("Erro: Nenhum vetor alocado para redimensionar\n");
        return;
    }

    // Aloca um novo vetor com o novo tamanho
    int *novo_vetor = (int *)malloc(novo_tamanho * sizeof(int));

    if (novo_vetor == NULL) {
        printf("Erro: Falha ao alocar novo vetor\n");
        return;
    }

    // Copia os elementos do vetor antigo para o novo
    int elementos_para_copiar = (novo_tamanho < tamanho_vetor_atual) ?
                                novo_tamanho : tamanho_vetor_atual;

    for (int i = 0; i < elementos_para_copiar; i++) {
        novo_vetor[i] = vetor_atual[i];
    }
}

```

```

    // Libera o vetor antigo
    free(vetor_atual);

    // Atualiza as variáveis
    vetor_atual = novo_vetor;
    tamanho_vetor_atual = novo_tamanho;
}

void imprimirVetor() {
    if (vetor_atual == NULL) {
        printf("Nenhum vetor alocado.\n");
        return;
    }

    printf("\nVetor atual (%d elementos):\n", tamanho_vetor_atual);
    for (int i = 0; i < tamanho_vetor_atual; i++) {
        printf("%d ", vetor_atual[i]);
    }
    printf("\n");
}

int main() {
    int opcao;
    int n;
    bool executando = true;
    bool inicializar;

    while (executando) {
        printf("\n--- Gerenciamento de Vetor ---\n");
        printf("1. Alocar vetor\n");
        printf("2. Desalocar vetor\n");
        printf("3. Redimensionar vetor\n");
        printf("4. Imprimir vetor\n");
        printf("5. Sair\n");
        printf("Escolha uma opção: ");
        scanf("%d", &opcao);

        switch (opcao) {
            case 1:
                if (vetor_atual != NULL) {
                    printf("Já existe um vetor alocado. Desaloque-o primeiro (Opção 2).\n");
                } else {
                    printf("Digite o tamanho do vetor: ");
                    scanf("%d", &n);
                    if (n > 0) {
                        printf("Deseja inicializar o vetor? (1-Sim/0-Não): ");
                        scanf("%d", &inicializar);
                        alocarVetor(n, inicializar);
                        if (vetor_atual != NULL) {
                            printf("Vetor de tamanho %d alocado com sucesso.\n", n);
                        }
                        if (inicializar) {
                            printf("Vetor inicializado com zeros.\n");
                        }
                    } else {
                        printf("Falha ao alocar o vetor.\n");
                    }
                }
            }
        }
    }
}

```

```

        } else {
            printf("Tamanho inválido.\n");
        }
    }
    break;
case 2:
    if (vetor_atual == NULL) {
        printf("Nenhum vetor alocado para desalocar.\n");
    } else {
        desalocarVetor();
        printf("Vetor desalocado com sucesso.\n");
    }
    break;
case 3:
    if (vetor_atual == NULL) {
        printf("Nenhum vetor alocado para redimensionar.\n");
    } else {
        printf("Digite o novo tamanho do vetor: ");
        scanf("%d", &n);
        if (n > 0) {
            redimensionarVetor(n);
            printf("Vetor redimensionado com sucesso.\n");
        } else {
            printf("Tamanho inválido.\n");
        }
    }
    break;
case 4:
    imprimirVetor();
    break;
case 5:
    if (vetor_atual != NULL) {
        printf("Desalocando vetor antes de sair...\n");
        desalocarVetor();
    }
    executando = false;
    printf("Saindo do programa.\n");
    break;
default:
    printf("Opção inválida.\n");
}

return 0;
}

```

7. Solução Sugerida

Função alocarVetor()


```

void alocarVetor(int n, bool inicializar) {
    // Aloca o vetor
    vetor_atual = (int *)malloc(n * sizeof(int));

    if (vetor_atual == NULL) {
        printf("Erro: Falha ao alocar vetor\n");
        return;
    }

    if (inicializar) {
        for(int i = 0; i < n; i++) {
            vetor_atual[i] = 0; // Isso forçará a alocação física
        }
    }

    tamanho_vetor_atual = n;
}

```

Função desalocarVetor()

```

void desalocarVetor() {
    if (vetor_atual == NULL) {
        return;
    }

    // Libera o vetor
    free(vetor_atual);

    // Reseta as variáveis
    vetor_atual = NULL;
    tamanho_vetor_atual = 0;
}

```

Função redimensionarVetor()

```
void redimensionarVetor(int novo_tamanho) {
    if (vetor_atual == NULL) {
        printf("Erro: Nenhum vetor alocado para redimensionar\n");
        return;
    }

    // Aloca um novo vetor com o novo tamanho
    int *novo_vetor = (int *)malloc(novo_tamanho * sizeof(int));

    if (novo_vetor == NULL) {
        printf("Erro: Falha ao alocar novo vetor\n");
        return;
    }

    // Copia os elementos do vetor antigo para o novo
    int elementos_para_copiar = (novo_tamanho < tamanho_vetor_atual) ?
                                novo_tamanho : tamanho_vetor_atual;

    for (int i = 0; i < elementos_para_copiar; i++) {
        novo_vetor[i] = vetor_atual[i];
    }

    // Libera o vetor antigo
    free(vetor_atual);

    // Atualiza as variáveis
    vetor_atual = novo_vetor;
    tamanho_vetor_atual = novo_tamanho;
}
```

8. Análise de Desempenho

8.1 Uso de Memória

- Execute o programa com diferentes tamanhos de vetor
- Observe o uso de memória no Gerenciador de Tarefas
- Compare o uso de memória entre alocação e desalocação

8.2 Testes de Estresse

1. Tente alocar um vetor muito grande
2. Observe o comportamento do sistema
3. Documente quando ocorrem falhas de alocação

9. Referências

1. [C Programming Language](#)
2. [Dynamic Memory Allocation in C](#)
3. [Memory Management in C](#)