Programação de Computadores

Raoni F. S. Teixeira

Aula 7 e 8 - Vetores e Strings

1 Introdução

Imagine que precisamos armazenar as notas de 3 alunos. Uma primeira abordagem seria:

```
float nota1, nota2, nota3;
printf("Nota␣do␣aluno␣1:␣");
scanf("%f", &nota1);
printf("Nota␣do␣aluno␣2:␣");
scanf("%f", &nota2);
printf("Nota␣do␣aluno␣3:␣");
scanf("%f", &nota3);
```

Isso funciona para 3 alunos, mas e se precisarmos armazenar 100 notas? Teríamos que declarar 100 variáveis (`nota1`, `nota2`, ..., `nota100`) e escrever 100 instruções `scanf`! Além de extremamente trabalhoso, este código seria impossível de manter.

E se o número de alunos for variável, determinado apenas em tempo de execução? Não poderíamos sequer declarar as variáveis necessárias! Precisamos de uma estrutura de dados que permita armazenar coleções de valores do mesmo tipo de forma eficiente.

2 Vetores

2.1 Definição

Um **vetor** é uma coleção de variáveis de um mesmo tipo referenciada por um nome comum.

Características dos vetores:

- Cada variável (elemento) é acessada por meio de um **índice**.
- O tamanho máximo n da coleção é pré-definido e deve ser uma constante.
- Em C, os índices variam de 0 a $n - 1$ (o primeiro elemento tem índice 0).
- Todos os elementos são armazenados em posições consecutivas de memória.

2.2 Declaração de Vetores

A declaração de um vetor segue a sintaxe:

```
tipo identificador[N];
```

Onde `N` é o tamanho do vetor (número de elementos). Em C, o primeiro elemento está na posição 0 e o último na posição `N-1`.

Exemplos:

```
float notas[100];      // Vetor de 100 numeros reais
int medias[100];       // Vetor de 100 inteiros
char nome[200];        // Vetor de 200 caracteres
```

A Figura 1 ilustra a estrutura de um vetor na memória.

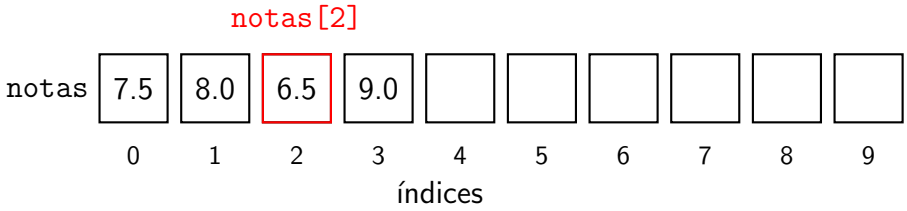

Figura 1: Representação de um vetor de 10 elementos na memória.

2.3 Acessando Elementos de um Vetor

Para acessar ou modificar um elemento do vetor, usamos a notação de colchetes com o índice desejado:

```
vetor[i] = valor;      // Atribui valor ao elemento i
x = vetor[i];          // Le o valor do elemento i
```

O índice pode ser uma constante, variável ou expressão que resulte em um valor inteiro.

2.4 Exemplo Completo

```
float notas[10], ultima_nota;
int i;

// Atribui nota 8.5 para todos os alunos
for (i = 0; i < 10; i++) {
    notas[i] = 8.5;
}

// O ultimo aluno ganhou um ponto de bonus
```

```
notas[9] = notas[9] + 1.0;
ultima_nota = notas[9];

printf("Ultima␣nota:␣%.1f\n", ultima_nota);  // 9.5
```

2.5 Exemplo: Lendo Até 100 Notas

Agora podemos resolver o problema inicial de forma elegante:

```
int numero_notas, i;
float notas[100];

printf("Numero␣de␣notas:␣");
scanf("%d", &numero_notas);

// Validacao importante!
if (numero_notas > 100) {
    printf("ATENCAO:␣Numero␣de␣alunos␣muito␣grande.\n");
    printf("Lendo␣apenas␣os␣100␣primeiros...\n");
    numero_notas = 100;
}

for (i = 0; i < numero_notas; i++) {
    printf("Nota␣do␣aluno␣%d:␣", i + 1);
    scanf("%f", &notas[i]);
}
```

Note a importância da validação: se o usuário informar mais de 100 notas, limitamos o número para evitar acessar posições inválidas do vetor.

3 Cuidados com Vetores

3.1 Tamanho Fixo

O tamanho do vetor é constante e não pode ser mudado durante a execução do programa. Por isso, devemos escolher um tamanho apropriado:

- Se for muito grande, haverá desperdício de memória.
- Se for muito pequeno, faltará espaço para os dados.

3.2 Índices Fora dos Limites

IMPORTANTE: Em C, o compilador **não** verifica se os índices estão dentro dos limites válidos do vetor. Acessar posições inválidas causa comportamento indefinido e pode:

- Alterar valores de outras variáveis
- Causar *segmentation fault* (erro de segmentação)
- Corromper dados do programa

3.3 Exemplo de Acesso Inválido

Considere o seguinte código problemático:

```
int d, vetor[5], f;
d = 0;
vetor[3] = 9;
vetor[5] = 5;       // ERRO: indice invalido!
vetor[-1] = 1;      // ERRO: indice negativo!
printf("%d\n", d);
```

A Figura 2 mostra o que pode acontecer na memória.

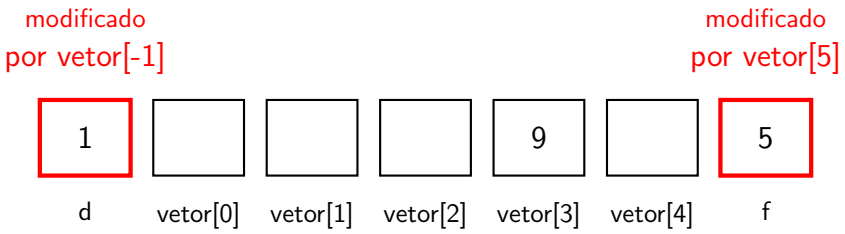

Figura 2: Acesso fora dos limites do vetor modifica outras variáveis.

O programa imprimirá 1 (não 0!) porque `vetor[-1]` acidentalmente modificou a variável `d`. Dependendo de como o compilador organiza a memória, o programa pode até causar *segmentation fault*.

4 Strings (Cadeias de Caracteres)

4.1 Problema Motivador

Considere o problema: *Escreva um programa que leia o nome de uma cidade e imprima as letras na ordem inversa. Por exemplo, se o nome for ROMA, deverá imprimir AMOR.*

Como podemos representar um nome em C? Um nome é uma coleção de caracteres, então podemos usar um vetor de caracteres!

4.2 Definição de String

Em C, textos são representados por vetores de caracteres com uma característica especial: para indicar o final do texto, usamos o caractere especial `'\0'` (caractere nulo, com valor inteiro 0).

String (Cadeia de Caracteres): Um vetor de caracteres terminado com o caractere `'\0'`.
ATENÇÃO: Sempre reserve espaço extra para o caractere de fim! Para armazenar "ROMA" (4 letras), precisamos de um vetor de tamanho 5.

A Figura 3 ilustra a representação de uma string na memória.

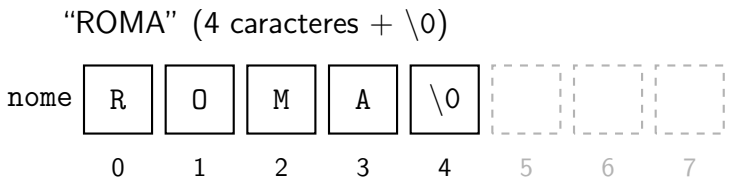

Figura 3: Representação da string "ROMA" em um vetor de 8 posições.

4.3 Lendo e Imprimindo Strings

As funções `scanf` e `printf` facilitam a leitura e impressão de strings usando o especificador `%s`:

```
char nome[30];
printf("Escreva o nome: ");
scanf("%s", nome);
printf("O nome digitado foi %s\n", nome);
```

Observe: Para strings, não usamos o operador `&` no `scanf`. Isso ocorre porque o nome do vetor sem colchetes já representa o endereço do primeiro elemento.

O `scanf` automaticamente adiciona o caractere `'\0'` ao final da string lida. Entretanto, `scanf` para de ler quando encontra um espaço, então não é adequado para ler textos com espaços.

4.4 Exemplo: Invertendo um Nome

```
char nome[30], nome_invertido[30];
int i, tamanho;

// Le o nome
printf("Escreva o nome: ");
scanf("%s", nome);

// Descobre o numero de caracteres do nome
for (tamanho = 0; nome[tamanho] != '\0'; tamanho++);

// Copia o nome em outro vetor na ordem inversa
for (i = 0; i < tamanho; i++)
    nome_invertido[i] = nome[tamanho - i - 1];

// Coloca o caractere de final de string
nome_invertido[tamanho] = '\0';

printf("O nome invertido e %s\n", nome_invertido);
```

O primeiro laço é um padrão comum para descobrir o comprimento de uma string: percorremos o vetor até encontrar '\0'.

5 Definindo Constantes

Ao trabalhar com vetores, frequentemente usamos valores constantes para definir tamanhos. Observe o seguinte código:

```
#include <stdio.h>

int main() {
    float provas[60], exercicios[60], media_turma;
    int i;

    printf("Notas das provas e exercicios: ");
    for (i = 0; i < 60; i++)
        scanf("%f %f", &provas[i], &exercicios[i]);

    for (media_turma = 0, i = 0; i < 60; i++)
        media_turma += (provas[i] + exercicios[i]) / 2;

    media_turma = media_turma / 60;
    printf("Media da turma: %f\n", media_turma);
    return 0;
}
```

Quantas vezes aparece o número 60? Se o número de alunos mudar, precisaremos alterar o código em vários lugares, aumentando a chance de erros.

5.1 A Diretiva #define

Podemos dar um nome a uma constante usando a diretiva `#define`:

```
#include <stdio.h>
#define NUM_ALU 60

int main() {
    float provas[NUM_ALU], exercicios[NUM_ALU], media_turma;
    int i;

    printf("Notas das provas e exercicios: ");
    for (i = 0; i < NUM_ALU; i++)
        scanf("%f %f", &provas[i], &exercicios[i]);

    for (media_turma = 0, i = 0; i < NUM_ALU; i++)
        media_turma += (provas[i] + exercicios[i]) / 2;

    media_turma = media_turma / NUM_ALU;
    printf("Media da turma: %f\n", media_turma);
    return 0;
}
```

Agora, para mudar o número de alunos, basta alterar uma única linha. O pré-processador substitui todas as ocorrências de `NUM_ALU` por 60 antes da compilação.

Boa prática: Use `#define` para constantes que aparecem múltiplas vezes no código. Isso torna o programa mais legível e fácil de manter.

6 Vetores em Funções

6.1 Passagem de Vetores como Parâmetros

Para acessar uma posição específica do vetor, usamos colchetes: `vetor[3]`. Mas e se quisermos referenciar o vetor inteiro?

Quando escrevemos o nome do vetor *sem* os colchetes, obtemos uma referência para a coleção inteira. Por exemplo, simplesmente `vetor`.

6.2 Consequências Importantes

1. **Vetores são sempre passados por referência!** Diferente de tipos simples como `int` ou `float`.
2. **Modificações no vetor dentro da função afetam o vetor original.** Não estamos trabalhando com uma cópia.

6.3 Declaração de Função com Vetor

Para declarar uma função que recebe um vetor, usamos colchetes vazios:

```
void funcao(int vetor[], int n) {
    // codigo
}
```

O parâmetro `n` indica o tamanho útil do vetor (quantos elementos estão sendo usados).

6.4 Exemplo Completo

```
#include <stdio.h>

void preencher_vetor(int vetor[], int n) {
    int i;
    for (i = 0; i < n; i++)
        vetor[i] = 5;
}

int main() {
    int vetor[10], i;

    // Inicializa com 8
    for (i = 0; i < 10; i++)
        vetor[i] = 8;

    // Chama funcao que modifica o vetor
    preencher_vetor(vetor, 10);

    // Imprime o vetor
    for (i = 0; i < 10; i++)
        printf("%d␣", vetor[i]);

    return 0;
}
```

Este programa imprimirá dez vezes o número 5, pois a função `preencher_vetor` modificou o vetor original.

6.5 Funções Auxiliares para Vetores

É comum criar funções reutilizáveis para operações com vetores:

```
#include <stdio.h>
#define NUM_ELEM1 10
#define NUM_ELEM2 30

void ler_vetor(int vetor[], int n) {
    int i;
    for (i = 0; i < n; i++) {
        printf("vetor[%d]: ", i);
        scanf("%d", &vetor[i]);
    }
}

void escrever_vetor(int vetor[], int n) {
    int i;
    for (i = 0; i < n; i++)
        printf("vetor[%d]: %d\n", i, vetor[i]);
}

int main() {
    int vetor1[NUM_ELEM1];
    int vetor2[NUM_ELEM2];

    printf("Digite os dados do primeiro vetor:\n");
    ler_vetor(vetor1, NUM_ELEM1);

    printf("Digite os dados do segundo vetor:\n");
    ler_vetor(vetor2, NUM_ELEM2);

    printf("Primeiro vetor:\n");
    escrever_vetor(vetor1, NUM_ELEM1);

    printf("Segundo vetor:\n");
    escrever_vetor(vetor2, NUM_ELEM2);

    return 0;
}
```

Observe como as mesmas funções `ler_vetor` e `escrever_vetor` podem ser usadas com vetores de tamanhos diferentes.

7 Exercícios

1. Escreva um programa que lê duas strings do teclado com até 80 caracteres e informa se elas são iguais. *Dica: Compare caractere por caractere.*

2. Escreva uma função que recebe um vetor de inteiros e seu tamanho, e retorna a soma de todos os elementos.

3. Escreva uma função que recebe um vetor de inteiros e seu tamanho, e retorna o maior elemento do vetor.

4. Escreva um programa que lê 10 números inteiros em um vetor e depois imprime apenas os números pares.

5. Escreva uma função que recebe uma string e conta quantas vogais ela contém (a, e, i, o, u, maiúsculas ou minúsculas).

6. Escreva uma função que lê uma palavra do teclado e informa se ela é um palíndromo. Um palíndromo é uma palavra que pode ser lida da mesma forma de trás para frente. Exemplos: ARARA, RADAR, REVIVER, ANA.

7. Escreva uma função que recebe dois vetores de inteiros do mesmo tamanho e retorna (através de um terceiro vetor) a soma elemento a elemento dos dois vetores.

8. Escreva um programa que lê duas strings e verifica se a segunda é uma substring da primeira (está contida na primeira).

9. Escreva uma função que recebe um vetor de inteiros e seu tamanho, e inverte a ordem dos elementos no próprio vetor (sem usar vetor auxiliar).

10. Escreva uma função que remove todas as ocorrências de um caractere específico de uma string. A remoção deve ser feita na própria string.

11. Implemente a função `strlen` da biblioteca padrão, que retorna o comprimento de uma string (número de caracteres antes do `'\0'`).

12. Implemente a função `strcpy` da biblioteca padrão, que copia uma string para outra. A assinatura deve ser: `void strcpy(char destino[], char origem[])`.

13. Escreva um programa que lê um texto (string) e conta quantas palavras ele contém. Considere que as palavras são separadas por espaços.

14. Implemente o algoritmo de ordenação *bubble sort* para ordenar um vetor de inteiros em ordem crescente.

8 Resumo

Pontos-chave desta aula:

- Vetores permitem armazenar coleções de valores do mesmo tipo
- Os índices em C começam em 0 e vão até $n - 1$
- Acessos fora dos limites não geram erro de compilação, mas causam comportamento indefinido
- Strings são vetores de caracteres terminados com `'\0'`
- Use `#define` para constantes que aparecem múltiplas vezes
- Vetores são sempre passados por referência para funções
- Modificações em vetores dentro de funções afetam o vetor original
- Sempre passe o tamanho útil do vetor como parâmetro adicional