# Estruturas de Dados Simples: Arrays e Strings

Nesta semana, vamos explorar as estruturas de dados **arrays** (vetores e matrizes) e **strings** em C. Esses conceitos são fundamentais em programação e têm uma ampla gama de aplicações práticas, como o armazenamento e manipulação de listas de valores e textos.

## 1. Arrays Unidimensionais (Vetores)

Um **array** ou **vetor** em C é uma coleção de variáveis do mesmo tipo, armazenadas em locais consecutivos de memória. Arrays são úteis para armazenar múltiplos valores de forma organizada e acessá-los através de um índice.

### Definição e Inicialização

Para declarar um array, especificamos o tipo de dado, o nome, e o número de elementos entre colchetes:

```
int notas[5]; // Declara um array de 5 inteiros
```

O comando acima cria uma variável chamada notas, que é um array (ou lista) com espaço para armazenar 5 números inteiros. Esses 5 espaços na memória são reservados, mas ainda não têm nenhum valor atribuído (estão 'vazios' ou 'não inicializados'). Em espaços vazios normalmente ficam valores "lixo" - ou seja, um valor aleatório que já estava na posição de memória alocada por ela. Esse valor pode variar toda vez que o programa é executado.

É possível atribuir valores iniciais para o vetor em C:

```
int notas[5] = {85, 90, 78, 92, 88};
```

ou também:

```
int notas[5] = {0}; // Todos os valores são inicializados como 0
```

### Acessando Elementos do Array

Para acessar um elemento desse vetor utilizamos a seguinte sintaxe:

```
notas[indice]
```

Em C, os vetores começam com o índice 0. Logo, para o vetor `notas` temos os seguintes acessos:

```c
#include <stdio.h>

int main(){
    int notas[5] = {85, 90, 78, 92, 88};

    printf("%d \n", notas[0]); // 85
    printf("%d \n", notas[1]); // 90
    printf("%d \n", notas[2]); // 78
    printf("%d \n", notas[3]); // 92
    printf("%d \n", notas[4]); // 88

    return 0;
}
```

Ou de uma forma mais visual teriamos:

```
Indice:  0  1  2  3  4
Valores: 85 90 78 92 88
```

**Valores Não Inicializados**

Em caso do programador tentar acessar um vetor não inicializado como no caso abaixo, o programa retornará o "lixo" que estava naquela posição.

```c
#include <stdio.h>

int main(){
    int notas[5];

    printf("%d \n", notas[0]); // Pode exibir um valor lixo, por exemplo 9377160
    return 0;
}
```

**Modificando valores no Array**

Após criar um array, é possível alterar os valores dele durante a execução do programa:

```c
#include <stdio.h>

int main(){
    int notas[5] = {85, 90, 78, 92, 88};

    notas[0] = 100; // O vetor notas agora se apresenta da seguinte forma {100,
90, 78, 92, 88}
```

```
    printf("%d \n", notas[0]); // 100
    return 0;
}
```

Aplicações Práticas

- Cálculo da Média: Podemos usar arrays para armazenar notas de estudantes e calcular a média.

```
#include <stdio.h>

int main() {
    int notas[5] = {85, 90, 78, 92, 88};
    int soma = 0;
    for(int i = 0; i < 5; i++) {
        soma += notas[i];
    }
    float media = soma / 5.0;
    printf("Média: %.2f\n", media);
    return 0;
}
```

- Manipulação de Listas de Números: É comum usar arrays para armazenar e processar listas de valores numéricos.

**Observação**

Ao trabalhar com vetores, preste bastante atenção ao índice que você está acessando. Lembre-se de que, se um vetor tem 5 posições, seus índices vão de 0 a 4. Tentar acessar um índice negativo ou um índice maior que 4 (nesse caso) resultará em valores "lixo", um erro de falha de segmentação (`segmentation fault`), ou ainda em comportamento inesperado do programa, como travamentos ou alterações em dados que não deveriam ser modificados.

Para evitar problemas, sempre verifique se o índice que você está usando está dentro dos limites do vetor.

## 2. Arrays Multidimensionais (Matrizes)

Um **array multidimensional** ou **matriz** em C é uma estrutura que pode armazenar dados em várias dimensões. Uma matriz 2D, por exemplo, é útil para representar tabelas, grades de jogos ou planilhas.

**Definição e Inicialização**

Para declarar uma matriz 2D (duas dimensões), especificamos o número de linhas e colunas:

```
int matriz[3][4]; // Declara uma matriz de 3 linhas e 4 colunas
```

Cada elemento da matriz será acessado por um índice de linha e um índice de coluna. Se quisermos declarar uma matriz para um jogo da velha, com 3 linhas e 3 colunas, fazemos assim:

```c
int tabuleiro[3][3]; // Uma matriz 3x3
```

Para inicializar a matriz com valores específicos, podemos usar uma lista de valores aninhada:

```c
int matriz[3][4] = {
    {1, 2, 3, 4},
    {5, 6, 7, 8},
    {9, 10, 11, 12}
};
```

Isso criará uma matriz como esta:

```
1   2   3   4
5   6   7   8
9  10  11  12
```

**Acesso aos Elementos**

Para acessar um elemento, usamos dois índices: um para a **linha** e outro para a **coluna**:

```c
printf("Elemento na linha 1, coluna 2: %d\n", matriz[0][1]); // Saída: 2
matriz[1][3] = 15; // Altera o elemento na segunda linha e quarta coluna para 15
```

Lembre-se de que, em C, a contagem começa do índice 0. Logo, a linha 1 é o índice 0, a linha 2 é o índice 1, e assim por diante.

**Exemplos de acesso com índice**

Linha 1 e Coluna 2: `matriz[0][1]`

Linha 3 e Coluna 3: `matriz[2][2]`

## 3. Diferença entre Vetor e Matriz

A diferença entre vetor e matriz é que o vetor é um array **unidimensional**, enquanto a matriz é um array **multidimensional** (geralmente 2D). Um vetor armazena uma lista linear de dados, enquanto uma matriz permite representar dados organizados em linhas e colunas.

## 4. Manipulação de Strings

Strings em C são arrays (vetores) de caracteres, **terminados por um caractere nulo (\0)**. Ou seja, uma string é um conjunto de caracteres armazenados de forma sequencial, e o caractere `\0` indica o fim da string. Com elas, podemos armazenar e manipular textos.

### Definição e Inicialização

Para declarar uma string em C, usamos um array de caracteres:

```c
#include <stdio.h>

int main(){
    char nome[20] = "Joao"; // String com 5 caracteres (incluindo o '\0' no final)

    return 0;
}
```

Observe que a string "Joao" tem 5 caracteres, mas o array nome tem espaço para 20 caracteres, para que possamos adicionar mais letras, se necessário.

### Operações Básicas com Strings

Em C, a manipulação de strings é feita através de funções da biblioteca string.h. Algumas das funções mais comuns são:

- `strlen()`: Retorna o comprimento de uma string (não conta o caractere nulo).
- `strcat()`: Concatena uma string à outra (anexa uma string ao final da outra).

Exemplo de concatenação e cálculo de comprimento:

```c
#include <stdio.h>
#include <string.h> // Biblioteca para mexer com strings

int main() {
    char nome[20] = "Joao";
    char sobrenome[20] = " Silva";

    strcat(nome, sobrenome); // Concatena sobrenome a nome
    printf("Nome completo: %s\n", nome);

    printf("Comprimento: %lu\n", strlen(nome)); // Saída: 10
    return 0;
}
```

### Acesso aos Elementos

Se você não lembra dos tipos primitivos, recomendo uma releitura da segunda aula.

Lembre-se, strings em C são arrays de caracteres. Logo, o acesso por índice funciona da mesma forma que em um vetor de inteiros ou outros tipos. Cada índice do array representa um caractere da string.

Exemplo de acesso aos caracteres de uma string:

```c
#include <stdio.h>
#include <string.h> // Biblioteca para mexer com strings

int main() {
    char nome[20] = "Joao";

    printf("Primeira letra do nome: %c\n", nome[0]);
    printf("Segunda letra do nome: %c\n", nome[1]);
    printf("Terceira letra do nome: %c\n", nome[2]);
    printf("Quarta letra do nome: %c\n", nome[3]);

    return 0;
}
```

Também é possível fazer isso utilizando a função strlen():

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

int main(){
    char nome[20] = "Joao";
    char sobrenome[20] = " Silva";

    strcat(nome, sobrenome); // Concatena sobrenome a nome, nome = 'Joao Silva'

    for(int i = 0; i < strlen(nome); i++){
        printf("%c\n", nome[i]);
    }

    return 0;
}
```

Esse exemplo imprime cada caractere individualmente da string concatenada, usando o strlen para determinar o número de caracteres.

**Dicas Finais**

- **Sempre se lembre do caractere nulo (`\0`)**: Quando trabalhamos com strings em C, o caractere `\0` é fundamental para marcar o fim da string. Isso é importante para funções como `strlen` e `strcat`, que dependem dessa marcação para saber até onde a string vai.
- **Cuidado ao manipular strings com espaço insuficiente**: Sempre tenha espaço suficiente para armazenar a string e o caractere nulo no array.
- **`%c` para caracteres e `%s` para strings: Ao printar uma variável, você irá usar o formatador `%c` se for printar um espaço específico de uma variável e `%s` se for printar a variável inteira.

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

int main(){
    char nome[20] = "Joao";

    printf("%s\n", nome); // Saída esperada é João
    printf("%c\n", nome[0]); // Saída esperada é o caractere 'c'

    return 0;
}
```

## 5. Exercícios práticos

**Para vetores:**

1. **Cálculo da Média**: Crie um programa que leia 5 notas de um aluno, armazene-as em um array e calcule a média.
2. **Contagem de Números Pares**: Crie um programa que leia 10 números e conte quantos deles são pares.
3. **Ordenação de Valores**: Peça ao usuário que insira 10 números e armazene-os em um array. Ordene o array em ordem crescente.
4. **Elemento Máximo e Mínimo**: Escreva um programa que leia 20 números e determine o maior e o menor valor.
5. **Frequência de um Valor**: Solicite ao usuário que insira 15 números e um valor a ser procurado. Informe quantas vezes o valor aparece no array.

**Para Matrizes:**

1. **Soma de Elementos da Matriz**: Crie um programa que leia uma matriz 3x3 e calcule a soma de todos os seus elementos.
2. **Transposição de Matriz**: Peça ao usuário uma matriz 2x2 e exiba sua transposta.
3. **Multiplicação de Matrizes**: Crie um programa que leia duas matrizes 2x2 e exiba o resultado de sua multiplicação.
4. **Matriz Identidade**: Escreva um programa que verifique se uma matriz 3x3 inserida pelo usuário é uma matriz identidade.
5. **Soma das Linhas e Colunas**: Peça ao usuário uma matriz 3x3 e exiba a soma dos elementos de cada linha e de cada coluna.

**Para Strings:**

1. **Palíndromo**: Peça ao usuário que insira uma palavra e verifique se ela é um palíndromo.

2. **Contagem de Vogais e Consoantes**: Crie um programa que leia uma frase e conte o número de vogais e consoantes.
3. **Inversão de String**: Solicite ao usuário que insira uma palavra e exiba a palavra invertida.
4. **Remoção de Espaços**: Escreva um programa que leia uma frase e remova todos os espaços em branco.
5. **Contagem de Caracteres Específicos**: Peça ao usuário uma frase e conte quantas vezes um caractere específico aparece.

## 6. Conclusão

Com a prática e o entendimento de arrays e strings, você poderá manipular e estruturar dados de forma mais eficaz em seus programas em C. Experimente os exercícios propostos para aprimorar suas habilidades!