

Resumo sobre o que foi passado nas aulas de estrutura de dados

1 – Vetor e Matriz

1 - Estrutura de dados Homogênea ou Matriz é uma coleção de variáveis de mesmo tipo, acessíveis com um único nome e armazenados contiguamente (um após o outro) na memória.

2 - A individualização de cada variável é feita através do uso de índices.

3 - Os Vetores são matrizes de uma só dimensão, isto é, necessita apenas 1 índice para acesso as variáveis.

4 - As Matrizes possuem mais de uma dimensão, isto é, necessitam de um índice para cada dimensão para acesso as variáveis.

2 - Ponteiros

Input:

```
#include <stdio.h>
#include <locale.h>

int main() {
    setlocale(LC_ALL, "Portuguese");

    char palavra[60];

    printf("Digite qualquer coisa que você quiser: ");
    scanf("%[^\n]s", palavra); // Não precisa do &, porque vetor ja é um ponteiro

    printf("O texto digitado foi:\n%s", palavra);
    printf("\nO endereço de palavra é: %p\n", &palavra);
    printf("\nO caracter da palavra no indice 2 é: %c\n", palavra[2]);
    printf("\nO endereço de palavra é: %p\n", palavra);

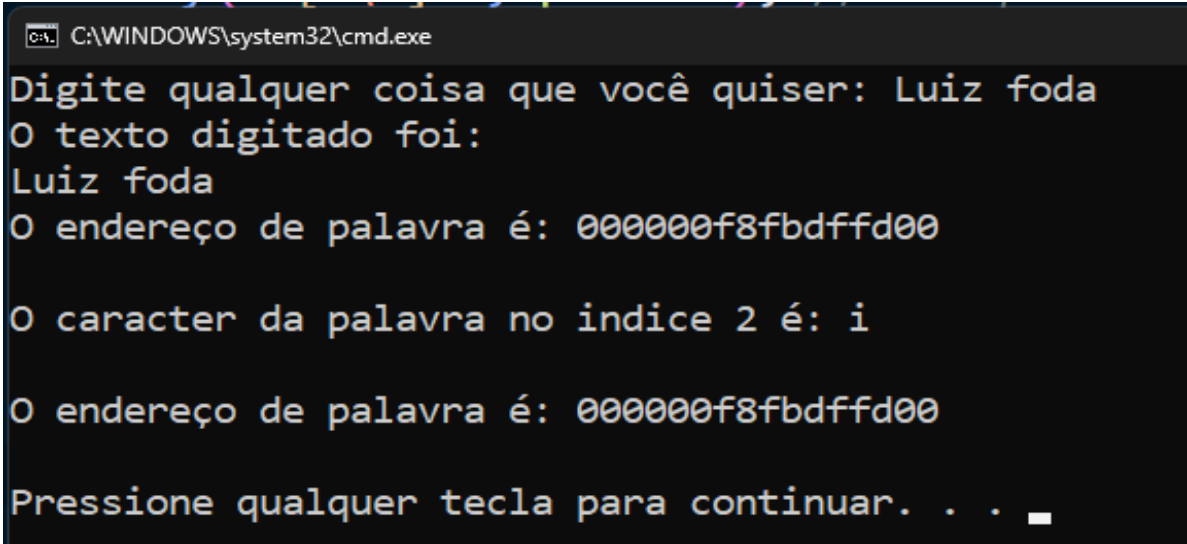
    return 0;
}
```

Um array de caracteres chamado palavra é declarado com espaço para 60 caracteres. Em seguida, o programa pede ao usuário que digite algo e usa a função scanf() para ler a entrada do usuário e armazená-la no array palavra. O formato de conversão "%[^\n]s" é usado para ler todos os caracteres

digitados pelo usuário, exceto o caractere de nova linha "\n", que é inserido pelo usuário ao pressionar Enter.

Observe que não é necessário usar o operador "&" antes do nome do array palavra em scanf(), pois palavra é um ponteiro para o primeiro elemento do array, então ele já contém o endereço da primeira posição do array.

Output:

A screenshot of a Windows command prompt window. The title bar shows 'C:\WINDOWS\system32\cmd.exe'. The prompt is 'C>'. The program has executed and displayed the following text: 'Digite qualquer coisa que você quiser: Luiz foda', 'O texto digitado foi:', 'Luiz foda', 'O endereço de palavra é: 000000f8fbdffd00', 'O caracter da palavra no indice 2 é: i', 'O endereço de palavra é: 000000f8fbdffd00', and 'Pressione qualquer tecla para continuar. . . _' with a cursor at the end.

```
C:\WINDOWS\system32\cmd.exe
C>
Digite qualquer coisa que você quiser: Luiz foda
O texto digitado foi:
Luiz foda
O endereço de palavra é: 000000f8fbdffd00

O caracter da palavra no indice 2 é: i
O endereço de palavra é: 000000f8fbdffd00

Pressione qualquer tecla para continuar. . . _
```

Observe que os endereços da &palavra e palavra são diferentes. O primeiro se refere ao endereço da variável palavra na memória, enquanto o segundo se refere ao endereço do primeiro elemento do array palavra.

Input:

```

#include <stdio.h>
#include <locale.h>

int main() {
    setlocale(LC_ALL, "Portuguese");

    int vet[5], *ptr;

    for(int i = 0; i < 5; i++) {
        printf("Digite o valor na posição %d: ", i);
        scanf("%d", &vet[i]);
    }

    ptr = &vet;

    for(int i = 0; i < 5; i++) {
        printf("O valor dos vetores é %d | e o endereço é %p\n", *(ptr+i), ptr+i);
    }

    return 0;
}

```

O operador `*(ponteiro+i)` é utilizado para acessar o valor da posição `i` do vetor a partir do ponteiro que aponta para o vetor.

Em outras palavras, o ponteiro é inicializado com o endereço da primeira posição do vetor. Ao usar o operador de referenciamento `*` em conjunto com o ponteiro e um índice `i`, podemos acessar o valor armazenado na posição `i` do vetor.

A expressão `*(ponteiro+i)` é equivalente a `ponteiro[i]`, pois ambos acessam o valor da posição `i` do vetor. Entretanto, a primeira forma é mais utilizada quando se trabalha com ponteiros em vez de arrays.

O uso de `*(ponteiro+i)` em vez de `ponteiro[i]` pode ajudar a lembrar que estamos trabalhando com ponteiros e que o operador de referenciamento `*` é necessário para acessar o valor da posição do vetor apontada pelo ponteiro.

Output:

```
C:\WINDOWS\system32\cmd.exe
Digite o valor na posição 0: 10
Digite o valor na posição 1: 15
Digite o valor na posição 2: 20
Digite o valor na posição 3: 30
Digite o valor na posição 4: 45
O valor dos vetores é 10 | e o endereço é 000000b02d3ff910
O valor dos vetores é 15 | e o endereço é 000000b02d3ff914
O valor dos vetores é 20 | e o endereço é 000000b02d3ff918
O valor dos vetores é 30 | e o endereço é 000000b02d3ff91c
O valor dos vetores é 45 | e o endereço é 000000b02d3ff920

Pressione qualquer tecla para continuar. . . ■
```

3 – Dados Heterogêneos

Uma estrutura que apresenta mais de um tipo básico de dado é denominada heterogênea.

Vetor com tipo de dado FLOAT

5,8	9,0	4,3	7,0	2,7	8,6	7,4
0	1	2	3	4	5	6

■ Posições
□ Informação alocada no vetor

Estrutura com diferentes tipos de dados

João 34 5124,00	Maria 25 2154,00	Célia 30 1654,00	José 22 3214,00	...	N N N
0	1	2	3		N

Para definir uma struct, iniciamos com typedef struct e passamos os parâmetros dentro do seu bloco:

```
typedef struct {  
    int ano;  
    char cor[30];  
    char modelo[50];  
    int potencia;  
    float torque;  
} Carro;
```

Depois temos que chamar o struct dentro da nossa classe main, mais ou menos parecido com POO:

```
Carro Nissan;  
Carro Toyota;  
  
printf("Digite o ano do carro: ");  
scanf("%d", &Nissan.ano);  
getchar(); // Precisa limpar o buffer  
printf("Digite a cor do carro: ");  
scanf("%[^\n]s", Nissan.cor);  
getchar();  
printf("Digite o modelo do carro: ");  
scanf("%[^\n]s", Nissan.modelo);
```

Na frente da referência Carro, declaramos o nome, e toda vez que formos utilizar os parâmetros da classe Carro, temos que referencia-la pelo nome

Podemos também, adicionar uma struct dentro de outra para acessar os seus dados:

```
typedef struct {
    int dia, mes, ano;
} Data;

typedef struct {
    int idade;
    char situacao;
    char nome[30];
    Data d;
} Pessoa;
```

Neste exemplo, vemos o uso de uma função utilizando struct:

```
Pessoa LerPessoa() {
    Pessoa p;
    getchar(); // Tem que limpar o buffer, pois antes ele le o

    printf("Digite o seu nome: ");
    scanf("%[^\n]s", &p.nome);
    printf("Digite a data de nascimento: ");
    scanf("%d %d %d", &p.d.dia, &p.d.mes, &p.d.ano);
    printf("Digite sua idade: ");
    scanf("%d", &p.idade);
    getchar();
    printf("Digite o estado [R] para regular, [I] para irregu]
    scanf("%c", &p.situacao);

    return p;
}
```

A função é do tipo Pessoa, e tem como objetivo receber os dados das pessoas, e retorna os dados no final, para que possamos referenciá-la

Podemos também adicionar mais de um valor na entrada de dados, com podemos ver neste exemplo:

```
printf("Digite a data de nascimento: ");  
scanf("%d %d %d", &p.d.dia, &p.d.mes, &p.d.ano);
```

Outra função, só que dessa vez, ela recebe a struct nos parâmetros da função:

```
void imprimirPessoa(Pessoa pes) {  
    printf("Nome: %s\n", pes.nome);  
    printf("Data nascimento: %d/%d/%d\n", pes.d.dia, pes.d.mes, pes.d.ano);  
    printf("Idade: %d\n", pes.idade);  
    printf("Situação: %c\n", pes.situacao);  
  
    printf("\n");  
}
```

Ela retorna apenas os dados no final do programa, por isso ela é void

Na classe main, utilizaremos do vetor, para armazenar as pessoas, para isso declaramos o vetor do tipo Pessoa, já que Pessoa é um struct e o vetor vai receber diferentes tipos de dados.

```
int main() {
    setlocale(LC_ALL, "Portuguese");

    int qtd;

    printf("Digite a quantidade de pessoas que deseja cadastrar\n");
    scanf("%d", &qtd);

    Pessoa vet[qtd];

    for(int i = 0; i < qtd; i++) {
        vet[i] = LerPessoa();
    }

    for(int i = 0; i < qtd; i++) {
        imprimirPessoa(vet[i]);
    }
}
```

Utilizamos da estrutura de repetição para alimentarmos o vetor homogêneo, e após atingir o tamanho, ele pula para a próxima estrutura de repetição, assim imprimindo os resultados cadastrados.