Struct e Ordenação – Codafofo

Struct:

Introdução:

Em C, uma **struct** (ou **registro**) é uma estrutura de dados composta que permite armazenar diferentes tipos de dados dentro de um único objeto. Isso é útil para representar entidades do mundo real, como alunos, funcionários ou produtos.

1. O que é uma Struct?

Uma **Struct** ou (**registro)** é uma coleção de dados que podem ser de diferentes tipos, agrupados sob um mesmo nome. Isso permite armazenar informações relacionadas de forma organizada.

Exemplo: Uma Struct para armazenar dados de um aluno pode conter:

- Nome (*string*)
- Idade (*int*)
- Sexo (*char*)
- Nota final (*float*)

2. Como definir uma Struct em C

Para definir um registro, usamos a palavra-chave **`struct`**.

```
#include <stdio.h>

#include <string.h>


// Definição do registro

typedef struct {

    char nome[30];
```

```
    int idade;
    char sexo;
    float nota;
} Aluno;

int main() {
    // Declaração de uma variável do tipo Aluno
    Aluno estudante;

    // Atribuição de valores
    strcpy(estudante.nome, "João Silva");
    estudante.idade = 20;
    estudante.sexo = 'M';
    estudante.nota = 8.5;

    // Exibição dos dados
    printf("Nome: %s\n", estudante.nome);
    printf("Idade: %d\n", estudante.idade);
    printf("Sexo: %c\n", estudante.sexo);
    printf("Nota: %.2f\n", estudante.nota);

    return 0;
}
```

Explicação do código

- Criamos um **registro Aluno** com quatro campos: nome, idade, sexo e nota.
- Utilizamos typedef struct para facilitar a declaração de variáveis do tipo Aluno.
- Atribuímos valores ao registro e exibimos os dados na tela.

3. Vetor de Struct

Se quisermos armazenar vários alunos, podemos criar um **vetor de Structs**.

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

typedef struct {
    char nome[30];
    int idade;
    char sexo;
    float nota;
} Aluno;

int main() {
    Aluno alunos[3];

    // Entrada de dados
    for (int i = 0; i < 3; i++) {
        printf("Digite o nome do aluno %d: ", i + 1);
        scanf(" %[^\n]", alunos[i].nome);
        printf("Digite a idade: ");
        scanf("%d", &alunos[i].idade);
        printf("Digite o sexo (M/F): ");
        scanf(" %c", &alunos[i].sexo);
        printf("Digite a nota final: ");
        scanf("%f", &alunos[i].nota);
    }
```

```
    // Exibição dos dados
    printf("\nLista de Alunos:\n");
    for (int i = 0; i < 3; i++) {
        printf("Nome: %s, Idade: %d, Sexo: %c, Nota: %.2f\n",
               alunos[i].nome, alunos[i].idade, alunos[i].sexo, alunos[i].nota);
    }

    return 0;
}
```

Explicação

- Criamos um **vetor `alunos[3]`** para armazenar três alunos.
- Usamos um **loop `for`** para inserir e exibir os dados de cada aluno.

4. Ponteiros para Structs

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

typedef struct {
    char nome[30];
    int idade;
} Pessoa;

void preencherDados(Pessoa *p) {
    printf("Digite o nome: ");
    scanf(" %[^\n]", p->nome);
    printf("Digite a idade: ");
    scanf("%d", &p->idade);
}

void exibirDados(const Pessoa *p) {
    printf("Nome: %s, Idade: %d\n", p->nome, p->idade);
}

int main() {
    Pessoa pessoa1;

    preencherDados(&pessoa1);
    exibirDados(&pessoa1);

    return 0;
```

```
}
```

Explicação

- Criamos a função `preencherDados()`, que recebe um **ponteiro para struct** (`Pessoa *p`).
- Em `p->campo`, usamos `->` (em vez de `.`) para acessar os membros da struct.
- Passamos o endereço da variável (`&pessoa1`) para a função.

Ordenação:

Introdução:

A ordenação é um processo fundamental em ciência da computação, onde organizamos um conjunto de elementos de acordo com uma determinada ordem (crescente ou decrescente). Isso facilita a recuperação de informações e otimiza o desempenho de muitos algoritmos.

Os algoritmos de ordenação são:

1. **Bubble Sort**
2. **Selection Sort**
3. **Insertion Sort**
4. **Quick Sort**
5. **Shell Sort**
6. **Radix Sort**
7. **Bucket Sort**
8. **Heap Sort**
9. **Merge Sort**

1. Bubble Sort

Ideia do algoritmo:

Em cada iteração (varredura), “borbulhar” o elemento com maior valor para o fim do vetor, por meio de um processo de comparação em pares.

Passos

1. Percorra o vetor comparando elementos adjacentes.
2. Se um elemento for maior que o próximo, troque-os.
3. Repita esse processo até que o vetor esteja ordenado.

Complexidade

- **Melhor caso:** O(n) (quando já está ordenado)
- **Pior caso:** O(n^2) (quando está na ordem inversa)

Vantagens e Desvantagens

✅ Fácil de implementar
❌ Ineficiente para grandes conjuntos de dados

Exemplo de implementação em C

```
// Função Bubble Sort

void bubbleSort(int arr[], int n) {

    for (int i = 0; i < n - 1; i++) {

        for (int j = 0; j < n - i - 1; j++) {

            if (arr[j] > arr[j + 1]) {

                // Troca os elementos se estiverem fora de ordem

                int temp = arr[j];

                arr[j] = arr[j + 1];
```

```
                arr[j + 1] = temp;
            }
        }
    }
} // Função para imprimir o array
void printArray(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}

int main() {
    int arr[] = {64, 34, 25, 12, 22, 11, 90};
    int n = sizeof(arr) / sizeof(arr[0]);

    bubbleSort(arr, n);
    printf("Bubble Sort: ");
    printArray(arr, n);

    return 0;
}
```

2. Selection Sort

Ideia do Algoritmo

Em cada iteração (varredura),

encontrar o elemento com menor valor e trocar de

posição com o primeiro elemento do vetor. Em seguida,

encontrar o segundo menor valor e trocar com o

segundo elemento do vetor....

Passos

1. Encontre o menor elemento do vetor e troque com o primeiro elemento.
2. Encontre o segundo menor e troque com o segundo elemento.
3. Repita até que o vetor esteja ordenado.

Complexidade

- **Melhor caso:** $O(n^2)$
- **Pior caso:** $O(n^2)$

Vantagens e Desvantagens

✅ Poucas trocas de elementos
❌ Ineficiente para grandes listas

Exemplos de implementação em C

```
#include <stdio.h>


// Função Selection Sort
void selectionSort(int arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        int min_idx = i;
        for (int j = i + 1; j < n; j++) {
            if (arr[j] < arr[min_idx]) {
```

```
                min_idx = j;
            }
        }
        // Troca os elementos
        int temp = arr[min_idx];
        arr[min_idx] = arr[i];
        arr[i] = temp;
    }
}

// Função para imprimir o array
void printArray(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}

int main() {
    int arr[] = {64, 34, 25, 12, 22, 11, 90};
    int n = sizeof(arr) / sizeof(arr[0]);

    selectionSort(arr, n);
    printf("Selection Sort: ");
    printArray(arr, n);

    return 0;
```

```
}
```

3. Insertion Sort

Ideia do Algoritmo

percorrer o conjunto de elementos da esquerda para a direita e à medida que avança vai deixando os elementos mais à esquerda ordenados.

Um exemplo prático do Insertion Sort é quando estamos jogando cartas de queremos ordená-las. Para cada carta, devemos procurar o seu devido lugar, deslocar as demais e inserir a carta no baralho.

Passos

1. Pegue o segundo elemento e compare com o primeiro, movendo-o para a posição correta.
2. Pegue o terceiro e insira na posição correta entre os primeiros dois, e assim por diante.

Complexidade

- **Melhor caso:** O(n) (se já estiver ordenado)
- **Pior caso:** $O(n^2)$

Vantagens e Desvantagens

✅ Eficiente para listas pequenas ou quase ordenadas
❌ Não é eficiente para grandes listas

Exemplos de implementação em C:

```
#include <stdio.h>


// Função Insertion Sort
```

```
void insertionSort(int arr[], int n) {
    for (int i = 1; i < n; i++) {
        int key = arr[i];
        int j = i - 1;

        // Move os elementos maiores que a chave para a direita
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}

// Função para imprimir o array
void printArray(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}

int main() {
    int arr[] = {64, 34, 25, 12, 22, 11, 90};
    int n = sizeof(arr) / sizeof(arr[0]);

    insertionSort(arr, n);
```

```
    printf("Insertion Sort: ");
    printArray(arr, n);

    return 0;
}
```

4. Quick Sort

Ideia do Algoritmo

1. Escolher um elemento do vetor, chamado pivô;
2. Rearranjar o vetor de modo que os elementos à esquerda do pivô sejam menores que ele e os elementos à direita do pivô sejam maiores que ele. Neste momento, o pivô estará em sua posição final e haverá dois subvetores. Este processo é chamado partição;
3. Recursivamente, aplique o Quick Sort no subvetor da esquerda e no subvetor da direita.

.

Passos

Escolha um elemento do vetor para ser o pivô. Em princípio, qualquer elemento pode ser o pivô. Entretanto, a maioria dos algoritmos escolhe como pivô:

O elemento mais à esquerda do vetor;

O elemento mais à direita do vetor;

O elemento do meio do vetor.

1. Percorra o vetor a partir da esquerda ATÉ QUE vetor[i]>= x
2. Percorra o vetor a partir da direita ATÉ QUE vetor[j]=<x
3. Troque vetor[i] com vetor[j]
4. Repita os passos de 2 a 4 até os índices i e j se cruzarem.

Exemplo de implementação em C:

```
#include <stdio.h>

// Função para trocar dois elementos void swap(int *a, int *b) { int temp = *a; *a
= *b; *b = temp; }

// Função para particionar o array int partition(int arr[], int low, int high) {
int pivot = arr[high]; // Escolhe o último elemento como pivô int i = (low - 1);

for (int j = low; j < high; j++) {
    if (arr[j] < pivot) {
        i++;
        swap(&arr[i], &arr[j]);
    }
}
swap(&arr[i + 1], &arr[high]);
return (i + 1);


}

// Função Quick Sort recursiva void quickSort(int arr[], int low, int high) { if
(low < high) { int pi = partition(arr, low, high);

   quickSort(arr, low, pi - 1);
    quickSort(arr, pi + 1, high);
}


}

// Função para imprimir o array void printArray(int arr[], int n) { for (int i =
0; i < n; i++) { printf("%d ", arr[i]); } printf("\n"); }

int main() { int arr[] = {64, 34, 25, 12, 22, 11, 90}; int n = sizeof(arr) /
sizeof(arr[0]);

quickSort(arr, 0, n - 1);
printf("Quick Sort: ");
printArray(arr, n);

return 0;


}
```

Link Sorting Algorithms:

Sorting Algorithms Animations | Toptal®