in4mation HUB

Fundamentos de Programação

Arrays de Structs

in4mation HUB

SINTAXE E EXEMPLOS

A sintaxe para declarar um array de structs em C é a seguinte:

```
tipo_da_struct nome_do_array[tamanho];
```

```
struct Aluno {
  char nome[50];
  int matricula;
};

struct Aluno alunos[10];
```

in4mation HUB

INICIALIZAÇÃO DE ARRAYS DE STRUCTS

Os elementos de um array de structs podem ser inicializados da mesma forma que variáveis normais:

```c
struct Aluno aluno1 = {"Ana Silva", 12345};
struct Aluno aluno2 = {"João Santos", 54321};

struct Aluno alunos[2] = {aluno1, aluno2};
```

in4mation HUB

ACESSANDO ELEMENTOS POR ÍNDICE E PONTEIROS

Os elementos de um array de structs podem ser acessados usando o operador `[]`:

```
1  struct Aluno aluno = alunos[0];
2  printf("Nome: %s\n", alunos[1].nome)
```

Ponteiros também podem ser usados para acessar elementos de arrays de structs:

```
1  struct Aluno *ptr_aluno = &alunos[0];
2  printf("Matricula: %d\n", ptr_aluno->matricula);
```

in4mation HUB

EXEMPLOS PRÁTICOS

```c
#include <stdio.h>

struct Pessoa {
  char nome[50];
  int idade;
};

int main() {
  struct Pessoa pessoas[3];

  // Inicializando elementos do array
  strcpy(pessoas[0].nome, "Joana");
  pessoas[0].idade = 25;

  strcpy(pessoas[1].nome, "Pedro");
  pessoas[1].idade = 30;

  strcpy(pessoas[2].nome, "Maria");
  pessoas[2].idade = 35;

  // Imprimindo informações de cada pessoa
  for (int i = 0; i < 3; i++) {
    printf("Nome: %s\n", pessoas[i].nome);
    printf("Idade: %d\n\n", pessoas[i].idade);
  }

  return 0;
}
```

Este exemplo cria um array `pessoas` de 3 structs `Pessoa`. As structs são inicializadas com dados específicos e um loop é utilizado para imprimir o nome e a idade de cada pessoa.

in4mation HUB

PASSANDO ARRAYS DE STRUCTS COMO PARÂMETROS

Arrays de structs podem ser passados como parâmetros para funções:

Nesse exemplo, a função `imprimirAlunos` recebe um array de structs `Aluno` como parâmetro e imprime os dados de cada aluno.

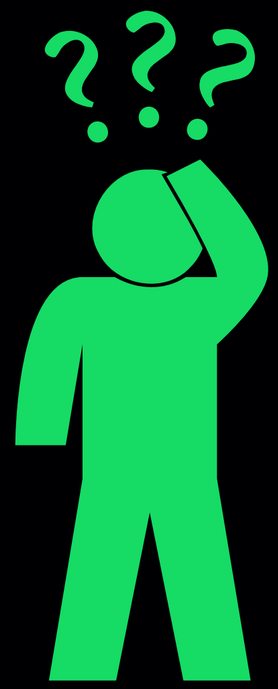
```c
void imprimirAlunos(struct Aluno alunos[], int tamanho) {
  for (int i = 0; i < tamanho; i++) {
    printf("Nome: %s\n", alunos[i].nome);
    printf("Matricula: %d\n\n", alunos[i].matricula);
  }
}

int main() {
  struct Aluno alunos[2];

  // ...

  imprimirAlunos(alunos, 2);

  return 0;
}
```

in4mation HUB

RETORNANDO ARRAYS DE STRUCTS COMO RESULTADO

Em C, não é possível retornar diretamente um array de structs como resultado de uma função. Isso porque o tamanho do array alocado na função chamadora pode ser diferente do tamanho esperado pela função que retorna o array.

Então, como podemos contornar essa limitação?



in4mation HUB

CONTORNANDO ESSA SITUAÇÃO

01 **Passar o array como parâmetro e modificá-lo dentro da função**

A função recebe o array como parâmetro e modifica seus elementos diretamente.

02 **Retornar um ponteiro para o array alocado dentro da função**

A função aloca memória para o array e retorna um ponteiro para o primeiro elemento. A responsabilidade de liberar a memória alocada é da função chamadora.

in4mation HUB

```c
void ordenarAlunosPorNome(struct Aluno alunos[], int tamanho) {
  // Ordenação por bolha (exemplo simples)
  for (int i = 0; i < tamanho - 1; i++) {
    for (int j = 0; j < tamanho - i - 1; j++) {
      if (strcmp(alunos[j].nome, alunos[j + 1].nome) > 0) {
        struct Aluno temp = alunos[j];
        alunos[j] = alunos[j + 1];
        alunos[j + 1] = temp;
      }
    }
  }
}

int main() {
  struct Aluno alunos[2];

  // ...

  ordenarAlunosPorNome(alunos, 2); // Passa o array para ordenação

  // ... (utilizar o array ordenado)

  return 0;
}
```

EXEMPLO

Modificando o array dentro da função

in4mation HUB

EXEMPLOS PRÁTICOS

Criando um array de structs para armazenar alunos utilizando a abordagem vista anteriormente

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

struct Aluno {
  char nome[50];
  int matricula;
};

void imprimirAlunos(struct Aluno alunos[], int tamanho) {
  for (int i = 0; i < tamanho; i++) {
    printf("Nome: %s\n", alunos[i].nome);
    printf("Matricula: %d\n\n", alunos[i].matricula);
  }
}

int main() {
  struct Aluno alunos[3];

  // Inicializando elementos do array
  strcpy(alunos[0].nome, "Carlos");
  alunos[0].matricula = 98765;

  strcpy(alunos[1].nome, "Beatriz");
  alunos[1].matricula = 12345;

  strcpy(alunos[2].nome, "Daniel");
  alunos[2].matricula = 54321;

  imprimirAlunos(alunos, 3);

  return 0;
}
```

in4mation HUB

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

struct Aluno {
  char nome[50];
  int matricula;
};

void ordenarAlunosPorNome(struct Aluno alunos[], int tamanho) {
  // ... (implementação da ordenação por bolha, como visto anteriormente)
}

int main() {
  struct Aluno alunos[2];

  // ... (inicializar o array de alunos)

  ordenarAlunosPorNome(alunos, 2);  // Ordena o array por nome

  // ... (utilizar o array ordenado)

  return 0;
}
```

Ordenando um array de structs por nome utilizando a abordagem de modificar o array dentro da função

in4mation HUB

BUSCANDO UM ELEMENTO EM UM ARRAY DE STRUCTS

01 **Busca sequencial**

Percorre o array elemento por elemento comparando um valor específico com o membro desejado (por exemplo, a matrícula) até encontrar uma correspondência.

02 **Busca binária (para arrays ordenados)**

Se o array de structs já estiver ordenado por um determinado critério (como nome ou matrícula), podemos empregar a busca binária para localizar um elemento de forma mais eficiente.

in4mation HUB

```c
#include <stdio.h>

struct Aluno {
  char nome[50];
  int matricula;
};

int buscarAlunoPorMatricula(struct Aluno alunos[], int tamanho, int matriculaBusca) {
  for (int i = 0; i < tamanho; i++) {
    if (alunos[i].matricula == matriculaBusca) {
      return i; // Retorna o índice do elemento encontrado
    }
  }
  return -1; // Elemento não encontrado
}

int main() {
  struct Aluno alunos[3];

  // ... (inicializar o array de alunos)

  int matricula = 12345; // Matrícula a ser buscada
  int indice = buscarAlunoPorMatricula(alunos, 3, matricula);

  if (indice != -1) {
    printf("Aluno encontrado no índice %d\n", indice);
  } else {
    printf("Aluno não encontrado\n");
  }

  return 0;
}
```

EXEMPLO

Busca sequencial por matrícula

in4mation HUB

OBSERVAÇÃO SOBRE A BUSCA BINÁRIA

01 A busca binária requer que o array de structs esteja previamente ordenado por um determinado critério.

02 Implementar a busca binária envolve conceitos adicionais de ordenação e pesquisa binária.

03 Essas são apenas algumas aplicações básicas de arrays de structs em C.

04 Existem muitas outras possibilidades de manipulação e organização de dados complexos utilizando esse recurso.

in4mation HUB

DESAFIO

Crie uma struct chamada `Aluno` com os membros `nome` (string) e `idade` (int). Em seguida, declare um array de 3 `Aluno`s e preencha-o com dados lidos do usuário. Por fim, imprima na tela os dados dos alunos.