INSTITUTO DE INFORMÁTICA
Universidade Federal de Goiás

Estruturas

Bruno Brandão

INSTITUTO DE INFORMÁTICA
UFG

UFG
UNIVERSIDADE FEDERAL DE GOIÁS

O que é uma estrutura (*struct*)?

- Um tipo de dado definido pelo usuário que agrupa variáveis de diferentes tipos.
- Por exemplo, um livro:
- Livro
  - título (char[])
  - autor (char[])
  - preço (float)
- Útil quando variáveis estão relacionadas a uma mesma entidade

Criando um Struct

```c
struct Book {
    char title[50];
    char author[50];
    float price;
};
```

- Este código cria o formato da estrutura, mas ainda não cria nenhuma variável

Criando variáveis

- Após criar o struct podemos criar variáveis com esse tipo

```
struct Book {
    char title[50];
    char author[50];
    float price;
};

struct Book myBook;
struct Book favoriteBooks[10];  // Array of Book structs
```

Typedef

- Define um tipo de dado com um nome específico
- Facilita a escrita de códigos grandes onde precisamos lembrar os tipos de variáveis que definimos para cada elemento.
- Exemplo, fazermos um sistema de notas de alunos onde todas as notas são floats.
  - `typedef float nota;`
- Assim não seria necessário lembrar que uma nota é float e poderíamos criar variáveis com:
  - `nota prova1;`

Typedef e structs

```c
typedef struct {
    char title[50];
    char author[50];
    float price;
} Book;

Book myBook;  // Now you can use 'Book' directly!
```

Acessar elementos

- Para acessar elementos basta utilizar o operador ponto

```
myBook.price = 19.99;
strcpy(myBook.title, "The Alchemist");
strcpy(myBook.author, "Paulo Coelho");
```

- Para vetores, é preciso indicar o índice

```
favoriteBooks[0].price = 10.99;
strcpy(favoriteBooks[0].title, "1984");
strcpy(favoriteBooks[0].author, "George Orwell");
```

Ponteiros para Estruturas

- Ponteiros para estruturas são definidos normalmente
- O acesso aos elementos é feito pelo operador ->
- Equivale a usar (*ptr).membro

```
Book *bookPtr = &myBook;
bookPtr->price = 15.75;  // Use '->' when accessing via a pointer
```

Passando Estruturas para Funções

- Passar estruturas diretamente para funções cria uma cópia, use ponteiros caso queira alterar elementos da estrutura

```
void updatePrice(Book *b, float newPrice) {
    b->price = newPrice;
}

updatePrice(&myBook, 22.99);
```

Copiando Estruturas

- Estruturas podem ser copiadas normalmente como variáveis

```
book2 = book1;  // Direct assignment, shallow copy
```

- ATENÇÃO: Copiar estruturas desta forma é seguro quando não há ponteiros dentro da estrutura
- Ponteiros são copiados APENAS os endereços e não os dados
- Vetores são ok

Inicialização Agregada e Parcial

```
// Aggregate initialization
struct Book book1 = {"1984", "George Orwell", 9.99};

struct Book book2 = {"The Alchemist"};  // Only 'title' initialized
// Remaining members ('author' and 'price') are set to zero or null.

struct Book library[2] = {
    {"1984", "George Orwell", 9.99},
    {"The Alchemist", "Paulo Coelho", 14.99}
};
```

Inicialização padrão

- É uma boa prática não inicializar estruturas parcialmente, mas sim definir uma inicialização padrão caso os valores ainda não estejam definidos

```
const struct Book DEFAULT_BOOK = {"Unknown Title", "Unknown Author", 0.0};

struct Book book = DEFAULT_BOOK;
```

Exercícios

- Faça um programa que receba uma lista de produtos e preços. Em seguida, receba uma lista de compras com quantidades e nomes, e calcule o total de dinheiro a ser gasto.

Estruturas Aninhadas

- Úteis para dados estruturados de forma hierárquica

```c
struct Address {
    char street[50];
    char city[50];
    int zipCode;
};

struct Person {
    char name[50];
    int age;
    struct Address address; // Nested struct
};
```

Acessando elementos aninhados

- Utilizar o operador ponto para caminhar nos níveis

```
struct Person p1;

strcpy(p1.name, "John Doe");
p1.age = 30;

// Access nested struct members
strcpy(p1.address.street, "123 Elm St");
strcpy(p1.address.city, "Metropolis");
p1.address.zipCode = 12345;
```

Exercícios

- Faça um programa que guarde times de esporte (com nome, e país) e seus jogadores (com nome, idade, e posição).

Alocação Dinâmica de Estruturas

- Estruturas são alocadas como qualquer outra variável
- Porém, como são compostas de vários membros, é preciso ser mais atencioso quanto à memória gasta.

```c
int numPeople = 3;
struct Person *people = malloc(numPeople * sizeof(struct Person));

if (people == NULL) {
    printf("Memory allocation failed!\n");
    return 1;
}
```