Structs

Prof. Andrei Braga
Prof. Geomar Schreiner

Structs

1. Até agora, vimos uma estrutura de dados: vetores
2. Propriedades importantes de um vetor:
    a. Todos os elementos de um vetor são do mesmo tipo
    b. Para selecionar um elemento de um vetor, especificamos a posição (índice) do elemento
3. Usamos uma **struct** para armazenar uma coleção de dados de tipos possivelmente diferentes
4. Propriedades importantes de uma struct:
    a. Os elementos (**membros**) de uma struct podem ser de tipos diferentes
    b. Para selecionar um elemento de uma struct, especificamos o nome do elemento

Structs - declaração de variáveis

1. Para declarar variáveis que são structs, podemos escrever

```c
struct {
  int dia;
  int mes;
  int ano;
} data1, data2;
```

```c
struct {
  int id;
  char nome[TAM_NOME+1];
  double salario;
} funcionario1, funcionario2;
```

2. Representação de `data1` na memória do computador:

3. Os nomes dos membros de uma struct não conflitam com outros nomes de fora da struct

Structs - inicialização de variáveis

1. Assim como vetores, variáveis que são structs podem ser inicializadas quando declaradas

```c
struct {
  int dia;
  int mes;
  int ano;
} data1 = { 9, 11, 2003 },
  data2 = { 3, 1, 2008 };
```

```c
struct {
  int id;
  char nome[TAM_NOME+1];
  double salario;
} funcionario1 = { 51, "Jose Silva", 5000.00 },
  funcionario2 = { 89, "Maria Souza", 5000.00 };
```

Structs - operações

1. Para acessar um membro de uma variável que é uma struct, escrevemos o nome da variável seguido de um . seguido do nome do membro

```c
struct {
  int dia;
  int mes;
  int ano;
} data1, data2;
```

```c
struct {
  int id;
  char nome[TAM_NOME+1];
  double salario;
} funcionario1, funcionario2;
```

```c
printf("Dia: %d\n", data1.dia);
printf("Nome do funcionario: %s\n", funcionario1.nome);
```

Structs - operações

1. Podemos atribuir valores aos membros de uma variável que é uma struct e usá-los em operações aritméticas (quando cabível)

```c
struct {
  int dia;
  int mes;
  int ano;
} data1, data2;
```

```c
struct {
  int id;
  char nome[TAM_NOME+1];
  double salario;
} funcionario1, funcionario2;
```

```c
data1.dia = 3;
media = (funcionario1.salario + funcionario2.salario) / 2;

scanf("%d", &data2.mes);
scanf("%lf", &funcionario1.salario);
```

Structs - operações

1. Diferente do que vale para vetores, podemos usar o operador `=` para atribuir uma struct a outra struct - desde que as structs sejam de tipos compatíveis

```
struct {
  int dia;
  int mes;
  int ano;
} data1, data2;
```

```
struct {
  int id;
  char nome[TAM_NOME+1];
  double salario;
} funcionario1, funcionario2;
```

```
data1 = data2;
funcionario2 = funcionario1;
```

2. O efeito do comando `data1 = data2;` é copiar `data2.dia` para `data1.dia`, `data2.mes` para `data1.mes` e `data2.ano` para `data1.ano`.

Structs - nomeando tipos

1. Para passar uma variável que é uma struct como argumento para uma função, precisamos definir um nome que indique o tipo desta variável
2. Opção 1: Definir uma struct tag

```c
struct data {
  int dia;
  int mes;
  int ano;
};
```

```c
struct funcionario {
  int id;
  char nome[TAM_NOME+1];
  double salario;
} funcionario1, funcionario2;
```

```c
struct data data1, data2;                       /* não é possível omitir */
struct funcionario funcionario3, funcionario4; /* a palavra struct! */
```

3. Nas declarações acima, não é possível omitir a palavra `struct`!

Structs - nomeando tipos

1. Para passar uma variável que é uma struct como argumento para uma função, precisamos definir um nome que indique o tipo desta variável
2. Opção 2: Definir um novo tipo usando `typedef`

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

```c
typedef struct funcionario {
  int id;
  char nome[TAM_NOME+1];
  double salario;
} Funcionario;
```

```c
Data data1, data2;
Funcionario funcionario1, funcionario2;
```

Structs - como argumentos e retorno de funções

1. Funções podem receber structs como argumentos e retornar structs

```c
void imprimeData(Data data) {
  printf("Dia: %d\n", data.dia);
  printf("Mes: %d\n", data.mes);
  printf("Ano: %d\n", data.ano);
}
```

```c
Data constroiData(int dia, int mes, int ano) {
  Data data;
  data.dia = dia;
  data.mes = mes;
  data.ano = ano;
  return data;
}
```

```c
imprimeData(data1);

data2 = constroiData(9, 11, 2003);
```

Structs - como argumentos e retorno de funções

1. Desvantagens de funções receberem uma struct como argumento e retornarem uma struct
    a. É feita uma cópia de todos os membros da struct - uma operação ineficiente, especialmente se a struct tiver muitos membros e se os seus membros contiverem muitos elementos
    b. Na lógica de algumas aplicações, pode ser necessário que não sejam feitas cópias da struct (ou seja, que exista no programa uma única cópia da struct) - por exemplo, quando manipulamos um arquivo aberto para leitura ou escrita através da struct FILE (<stdio.h>)
2. Em geral, é uma melhor abordagem funções receberem como argumentos ponteiros para structs e retornarem um ponteiro para uma struct

Structs - ponteiros para structs

1. Para declarar variáveis que são ponteiros para structs, podemos escrever

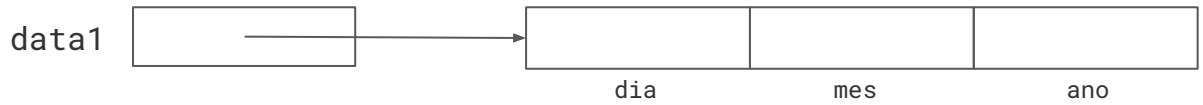
```c
typedef struct {
  int dia;
  int mes;
  int ano;
} Data;
```

```c
typedef struct {
  int id;
  char nome[TAM_NOME+1];
  double salario;
} Funcionario;
```

```c
Data *data1, *data2;
Funcionario *funcionario1, *funcionario2;

data1 = malloc(sizeof(Data));
```

2. Representação de `data1` na memória do computador:



Structs - ponteiros para structs

1. Para acessar um membro de uma struct que é apontada por um ponteiro, temos duas opções

```
Data *data1;

data1 = malloc(sizeof(Data));
```

   a. Opção 1: usar o operador *

```
(*data1).dia = 3;  /* os parênteses são necessários! */
```

   b. Opção 2: usar o operador ->

```
data1->dia = 3;
```

Structs - ponteiros para structs

1. Para acessar um membro de uma struct que é apontada por um ponteiro, temos duas opções

```
Data *data1;

data1 = malloc(sizeof(Data));
```

   a. Opção 1: usar o operador *

```
scanf("%d", &(*data1).mes);
```

   b. Opção 2: usar o operador ->

```
scanf("%d", &data1->mes);
```

Exercícios

1. Escreva as seguintes funções considerando o tipo `Data` definido nesta apresentação:
   a. `int extraiDia(Data data)`
      Retorna o dia que compõe a data `data`.
   b. `int comparaDatas(Data data1, Data data2)`
      Retorna `-1` se a data `data1` é anterior à data `data2`, `1` se a data `data1` é posterior à data `data2` e `0` se as datas `data1` e `data2` são iguais.
2. Declare um tipo `Complexo` que consista em uma struct contendo dois membros, `real` e `imaginario`, do tipo double, e faça o seguinte:
   a. Escreva uma função `criaComplexo` que recebe dois argumentos do tipo `double`, os armazena em uma variável do tipo `Complexo` e retorna esta variável.
   b. Escreva uma função `somaComplexos` que recebe dois argumentos do tipo `Complexo`, soma os valores dos membros correspondentes das structs recebidas, armazena os resultados em uma outra variável do tipo `Complexo` e retorna esta variável.

Exercícios

3. Faça o seguinte considerando o tipo `Data` definido nesta apresentação:
    a. Declare `data1` como uma variável que é um ponteiro para um valor do tipo `Data`. Faça o mesmo para `data2`.
    b. Faça a alocação da memória necessária para armazenar um valor do tipo `Data` e faça `data1` apontar para esta memória. Faça o mesmo para `data2`.
    c. Leia da entrada os dados necessários para preencher todos os membros de `data1`. Faça o mesmo para `data2`.
    d. Reescreva as funções do Exercício 1 considerando agora que os argumentos destas funções são ponteiros para valores do tipo `Data`. Use as novas funções passando `data1` e `data2` como argumentos.
    e. Libere a memória apontada por `data1`. Faça o mesmo para `data2`.

Exercícios

4. Considerando as definições a seguir, faça o que é pedido nos itens abaixo:

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

```c
typedef struct {
  int id;
  char nome[41];
  double salario;
  Data *nascimento;
} Funcionario;
```

   a. Declare `funcionario` como uma variável do tipo `Funcionario`.
   b. Faça a alocação da memória necessária para armazenar um valor do tipo `Data` e faça o membro `nascimento` da variável `funcionario` apontar para esta memória.
   c. Leia da entrada os dados necessários para preencher todos os membros da struct apontada pelo membro `nascimento` da variável `funcionario`.
   d. Use a função `imprimeData` definida nesta apresentação para imprimir a data armazenada na struct apontada pelo membro `nascimento` da variável `funcionario`.
   e. Libere a memória apontada pelo membro `nascimento` da variável `funcionario`.

Exercícios

5. Considerando a estrutura

```
typedef struct {
  float x;
  float y;
  float z;
} Vetor;
```

para representar um vetor no R3, implemente um programa que calcule a soma de doisvetores.

Exercícios

6. Crie uma estrutura representando os alunos de um determinado curso. A estrutura deve conter a matrícula do aluno, nome, nota da primeira prova, nota da segunda prova e nota da terceira prova.
    a. Permita ao usuário entrar com os dados de 5 alunos.
    b. Encontre o aluno com maior nota da primeira prova.
    c. Encontre o aluno com maior média geral.
    d. Encontre o aluno com menor média geral.
    e. Para cada aluno diga se ele foi aprovado ou reprovado, considerando o valor 6 para aprovação.

Exercícios

7. Faça um programa que leia os dados de 10 alunos (Nome, matrícula, Média Final), armazenando em um vetor. Uma vez lidos os dados, divida estes dados em 2 novos vetores,o vetor dos aprovados e o vetor dos reprovados, considerando a média mínima para a aprovação como sendo 5.0. Exibir na tela os dados do vetor de aprovados, seguido dos dados do vetor de reprovados.

Referências

1. Esta apresentação é baseada nos Capítulos 16 e 17 do livro King, K.N., C Programming: A Modern Approach, Norton, 1996.