Estruturas (*structs*)

BCC201 - Introdução a Programação

Professores: Túlio A. M. Toffolo e Puca V. P. Huachi

Slides: **Kelly Márcia de Oliveira** (estagiária em docência)

DECOM - UFOP

2019/1

Sumário

O que é uma estrutura

- Uma estrutura é um conjunto de uma ou mais variáveis (chamadas de campos ou membros) agrupadas sobre um único nome.
- As estruturas podem conter elementos com qualquer tipo de dados válidos em C (tipos básicos, vetores, ponteiros, ou mesmo outras estruturas).

Declaração de estruturas

- A declaração de estruturas é realizada através da seguinte sintaxe:

```
struct NomeDaEstrutura{
  tipo1 campo1, campo2;
  ...
  tipon campo;
};
```

- Exemplo: estrutura capaz de suportar datas.

```
struct Data{
  int dia, ano;
  char mes[10];
};
```

- A partir deste momento o compilador passa a conhecer um outro tipo, chamado **struct Data**.

Declaração de variáveis do tipo estrutura

- Para declarar uma variável do tipo **struct Data** basta indicar qual o tipo (**struct Data**) seguido do nome das variáveis.

```
1  struct Data d, datas[100], *ptrData;
```

- **d** é uma variável do tipo *struct Data*.
- **datas** é um vetor de 100 elementos, sendo cada um deles uma estrutura do tipo *struct Data*.
- **ptrData** é um ponteiro para o tipo *struct Data*.

Declaração de variáveis do tipo estrutura

- A declaração de variáveis pode também ser realizada quando se faz a definição da própria estrutura através da sintaxe:

```
struct NomeDaEstrutura{
  tipo1 campo1, campo2;
  ...
  tipon campo;
}v1, v2, ... , vn;
```

- Exemplo de definição de uma estrutura e declaração de variáveis:

```
struct Data{
  int dia, ano;
  char mes[10];
}d, datas[100], *ptrData;
```

Declaração de variáveis do tipo estrutura

- A declaração de uma estrutura pode ser realizada sem indicar qual o seu nome. No entanto, todas as variáveis desse tipo têm que ser declaradas no momento da definição. Essas estruturas não podem ser enviadas ou recebidas dentro de funções, pois não possuem um nome que as identifique na definição de parâmetros de uma função.

```
1  struct{
2    int dia, ano;
3    char mes[10];
4  }dtNasc;
```

Acesso aos membros de uma estrutura

- Para acessar o campo **c** de uma estrutura **e** usa-se o operador ponto (.) fazendo **e.c**.
- Exemplo:

```
1  struct Data{
2    int dia, ano;
3    char mes[10];
4  }dtNasc;
5
6  dtNasc.dia = 23;
7  strcpy(dtNasc.mes, "Janeiro");
8  dtNasc.ano = 1996;
9
10 printf("Data: %d/%s/%d", dtNasc.dia, dtNasc.mes, dtNasc.ano);
```

Carga inicial automática de estruturas

- Uma estrutura pode ser iniciada quando é declarada usando-se a sintaxe:

```
struct NomeDaEstrutura var = {valor_1, valor_2, ..., valor_n};
```

- O programa anterior poderia ter sido escrito de duas maneiras diferentes:

```
struct Data{
  int dia, ano;
  char mes[10];
}dtNasc = {23,1966,"Jan"};
```

ou

```
//Declaração da estrutura
struct Data{
  int dia, ano;
  char mes[10];
};

//Declaração da variável
struct Data dtNasc = {23,1966,"Jan"};
```

Carga inicial automática de estruturas

- Se a variável a ser inicializada for um vetor, a carga inicial é feita do mesmo modo, colocando cada um dos elementos dentro de chaves.

```
1  //Declaração da estrutura
2  struct Data{
3    int dia;
4    char mes[10];
5    int ano;
6  };
7
8  //Declaração da variável
9  struct Data dtNasc = {23,"Jan",1966};
10
11 //Declaração de um vetor de estruturas
12 struct Data v[] = {{1,"Jan",1990},{2,"Fev",1920}};
```

ou

```
1  struct Data v[2] = {{1,"Jan",1990},{2,"Fev",1920}};
```

Definição de tipos

- É possível representar uma estrutura unicamente por meio de uma palavra usando a palavra reservada **typedef**, cuja sintaxe é:

```
typedef tipo_existente sinônimo
```

- A palavra **typedef** não cria um novo tipo. Permite apenas que um determinado tipo possa ser denominado de forma diferente, de acordo com as necessidades do programador.

```
typedef struct pessoa{
  int idade;
  char sexo, estCivil;
  char nome[60];
}Pessoa;
```

Definição de tipos

- Agora a declaração de variáveis pode ser feita de duas formas diferentes:

```
1  struct pessoa Maria, Joao;
```

ou

```
1  Pessoa Maria, Joao;
```

- Se deseja-se utilizar sempre o sinônimo na definição de tipos e variáveis, é possível fazer a declaração da estrutura sem o nome.

```
1  typedef struct{ //A estrutura não tem nome
2    int idade;
3    char sexo, estCivil;
4    char nome[60];
5  }Pessoa;
```

Na definição de um typedef não podem ser declaradas variáveis.

Onde definir estruturas e typedef

- Se a definição de uma estrutura for realizada dentro de uma função ou procedimento, apenas essa função ou procedimento conhece essa definição.
- As estruturas devem ser definidas de forma a serem visíveis por todo o programa, fazendo a sua definição fora de qualquer função.

```
1  #include <...>
2  #include <...>
3
4  struct NomeDaEstrutura{...};
5  typedef ...;
6
7  // protótipos das funções
8  ...
9
10 //funções
11 f() {...}
12 g() {...}
13 main() {...}
14 h() {...}
```

Estruturas dentro de estruturas

- Como foi dito anteriormente, uma estrutura pode conter, na sua definição, variáveis simples, vetores, ponteiros ou mesmo outras estruturas.
- A única restrição é que todas as estruturas ou tipos utilizados na definição de uma nova estrutura têm que estar previamente definidos.

```
1  typedef struct{
2    int dia;
3    int mes;
4    int ano;
5  }Data;
6
7  typedef struct funcionario{
8    char nome[100];
9    int idade;
10   float salario;
11   Data nasc;
12 }Funcionario;
```

Passagem de estruturas para funções

- A passagem de estruturas para funções se faz indicando no parâmetro o tipo associado à estrutura (ou o typedef).

```
1  void Mostrar(struct funcionario x){
2    printf("Nome       : %s\n",x.nome);
3    printf("Idade      : %d\n",x.idade);
4    printf("Salario    : %.2f\n",x.salario);
5    printf("Dt. Nasc   : %d/%d/%d\n",x.nasc.dia,x.nasc.mes, x.nasc.
     ano);
6  }
7
8  int main(){
9    struct funcionario f = {"Joao",23,12345.67,{23,5,1954}};
10   Mostrar(f);
11   return 0;
12 }
```

Passagem de estruturas para funções

```
1  void Mostrar(Funcionario x){
2     printf("Nome       : %s\n",x.nome);
3     printf("Idade      : %d\n",x.idade);
4     printf("Salario    : %.2f\n",x.salario);
5     printf("Dt. Nasc   : %d/%d/%d\n",x.nasc.dia,x.nasc.mes, x.nasc.
      ano);
6  }
7
8  int main(){
9     Funcionario f = {"Joao",23,12345.67,{23,5,1954}};
10    Mostrar(f);
11    return 0;
12 }
```

- Nos dois casos, a passagem de parâmetros é realizada por valor, pois é colocada em x uma cópia da variável que lhe é enviada.
- Neste caso, não é possível alterar a variável enviada como argumento à função.

Passagem de estruturas para funções

- Como em C só existe passagem de parâmetros por valor, temos, obrigatoriamente, que passar o endereço daquilo que pretendemos alterar, nesse caso, o endereço da variável f.

```
1  void Ler(Funcionario *ptr){
2    printf("Qual o nome          : ");
3    gets((*ptr).nome);
4    printf("Qual a idade         : ");
5    scanf("%d",&(*ptr).idade);
6    printf("Qual o salario       : ");
7    scanf("%f",&(*ptr).salario);
8    printf("Qual a data nascim.  : ");
9    scanf("%d %d %d",&(*ptr).nasc.dia,&(*ptr).nasc.mes, &(*ptr).
     nasc.ano);
10 }
11
12 int main(){
13   struct funcionario f = {"Joao",23,12345.67,{23,5,1954}};
14   Mostrar(f);
15   Ler(&f);
16   Mostrar(f);
17   return 0;
18 }
```

Passagem de estruturas para funções

- A linguagem C coloca à disposição dos programadores o operador ->.

```
1  void Ler(Funcionario *ptr){
2    printf("Qual o nome          : ");
3    gets(ptr->nome);
4    printf("Qual a idade         : ");
5    scanf("%d",&ptr->idade);
6    printf("Qual o salario       : ");
7    scanf("%f",&ptr->salario);
8    printf("Qual a data nascim.  : ");
9    scanf("%d %d %d",&ptr->nasc.dia,&ptr->nasc.mes, &ptr->nasc.ano)
     ;
10 }
11
12 int main(){
13   struct funcionario f = {"Joao",23,12345.67,{23,5,1954}};
14   Mostrar(f);
15   Ler(&f);
16   Mostrar(f);
17   return 0;
18 }
```

Operações sobre estruturas

- Se **e** é uma estrutura e **c** é um campo dessa estrutura, então o operador ponto (.) permite obter o valor do campo **c** de **e** por meio de **e.c**.
- Se **p** é um ponteiro para uma estrutura e **c** é um campo dessa estrutura, então é possível obter o valor do campo **c** de **e** por meio de **(*p).c** ou **p->c**.
- Se **e1** e **e2** forem duas variáveis com a mesma estrutura, então, para copiar todos os campos de **e1** para **e2** basta fazer **e2 = e1**, isto é, pode-se fazer atribuição de estruturas.

Operações sobre estruturas

- Se **e** é uma estrutura, então **&e** devolve o endereço da estrutura em memória, isto é, o endereço do início da memória ocupada pela estrutura.
- Se **e** é uma estrutura e **c** é um campo dessa estrutura, então **&e.c** devolve o endereço de memória do campo **c** de **e**.
- Não se pode fazer comparações diretas entre estruturas através dos operadores $<$, $<=$, $>=$, $==$ ou $!=$. O programador deverá estabelecer qual a relação entre duas variáveis do tipo estrutura a partir de comparações entre os seus campos.

Operações sobre estruturas

```
Funcionario funcionarios[3] = {{"Maria",27,10400.50,{12,3,1992}},
{"Carlos",36,12345.67,{18,9,1982}},
{"Ana",33,12345.67,{23,4,1986}}};
```

- Para adicionar um dia à data de nascimento do primeiro funcionário do vetor:

```
funcionarios[0].nasc.dia++;
```

- Para colocar o ano de 1999 no segundo funcionário:

```
funcionarios[1].nasc.ano = 1999;
```

- Para alterar o nome do último funcionário:

```
strcpy(funcionarios[2].nome, "Anna");
```

- Para mostrar na tela o salário de cada funcionário:

```
for(i=0; i<3; i++)
   printf("%.2f\n", funcionarios[i].salario);
```

Exercícios

1. Crie uma estrutura para cadastrar o nome, a matrícula e duas notas de vários alunos. Em seguida imprima a matrícula, o nome e a média de cada um deles.

2. Escreva um programa que cadastre o nome, a altura, o peso, o cpf e sexo de algumas pessoas utilizando uma estrutura. Com os dados cadastrados, crie uma função que localiza uma pessoa através do seu CPF e imprime o seu IMC.

Dúvidas