Estruturas e Tipos definidos pelo programador

Estrutura

Estrutura é uma coleção de variáveis referenciadas com um mesmo nome. A linguagem C utiliza estruturas para fornecer um meio conveniente de manter informações relacionadas em um mesmo local.

A *definição de uma estrutura* forma um meio que pode ser utilizado para criar variáveis estruturadas. Cada estrutura é formada por uma ou mais variáveis ligadas logicamente. Essas variáveis são denominadas elementos da estrutura.

As estruturas, como grupos de variáveis logicamente ligadas, podem ser facilmente passadas para funções. Com a utilização de estruturas, seu código fonte poderá ser muito mais fácil de ser lido porque a conexão lógica entre os elementos da estrutura é óbvia.

Por exemplo, um agrupamento de nome e endereço em uma lista de endereçamento postal (CEP) é um conjunto comum de informação relacionada. O trecho do código que segue declara uma estrutura para os campos de nome e endereço; a palavra chave **struct** informa ao compilador que uma estrutura está sendo definida.

```
7  struct end_residencia{
8          char name[30];
9          char endereco[40];
10         char cidade[20];
11         char estado[2];
12         int cep;
13 };
```

Existem duas observações a fazer a respeito dessa definição:

- Terminação em ponto e vírgula, porque uma definição de estrutura é uma declaração;
- O rótulo de estrutura **end_residencia**, identifica essa estrutura de dados em particular, e é o seu nome.

Até o momento *nenhuma variável foi realmente declarada.* Apenas a forma dos dados foi definida. Para realmente declarar uma variável com essa estrutura, devemos escrever:

```
15 struct end_residencia func_end
```

Isso declara uma variável do tipo **end_residencia**, chamada **func_end**. Quando se define uma estrutura, na verdade estamos definindo uma variável complexa formada por elementos de estrutura.

É possível ainda declarar uma ou mais variáveis ao mesmo tempo em que se define a estrutura. Por Exemplo:

Este trecho de código define uma estrutura denominada **end_residencia** e declara as variáveis **func_end** e **cliente_end**, e que são do tipo **end_residencia**.

```
7  struct end_residencia{
8          char name[30];
9          char endereco[40];
10         char cidade[20];
11         char estado[2];
12         int cep;
13 }func_end, cliente_end;
```

A forma geral de uma definição de uma estrutura é:

```
4  struct nome_da_estrutura{
5     tipo nome_variável;
6     tipo nome_variável;
7     tipo nome_variável;
8     tipo nome_variável;
9     .
10    .
11    .
12 }variáveis_estrutura;
```

Referenciando os elementos de uma estrutura

O código abaixo atribui o CEP à variável de estrutura **func_end** (variável CEP interna à estrutura) anteriormente declarada:

```
14 func_end.cep = 07115000
```

Como é possível observarmos, o nome da variável **func_end** seguido por um ponto e o nome do elemento, referencia aquele elemento específico. O ponto significa que *segue um elemento de estrutura*. Todos os elementos de estrutura são acessados dessa forma. A sintaxe geral é:

```
16 nome_da_estrutura.nome_do_elemento
```

Portanto, para imprimir na tela o código do CEP, será necessário o seguinte comando:

```
18 printf("%d", func_end.cep);
```

Isso imprimirá o código CEP armazenado na variável **cep** da estrutura **func_end**.

Por exemplo, considere **func_end.name**. Esse elemento é uma string. Utilizando o comando gets() para inserir o nome, o comando deverá ser assim:

```
20 gets(func_end.name);
```

Isso passa o ponteiro de caractere para o início de **name**.

Vetores de Estruturas

Para declarar um vetor de estruturas, você deve primeiro definir a estrutura e em seguida declarar a variável de vetor daquele tipo. Por exemplo, para declarar um vetor de estrutura de 100 elementos, do tipo **end_residencia**, que já foi definido anteriormente, seria necessário o seguinte código:

```
22 struct end_residencia func-end[100];
```

Isso gera 100 conjuntos de variáveis organizados conforme definido na estrutura **end_resdencia.**

Para imprimir, por exemplo, o campo **cep** da estrutura 3 do vetor, utilizaríamos o seguinte código:

```
24 printf("%d", func_end[2].cep)
```

Passando estruturas para funções

Até aqui todas as estruturas e vetores dos exemplos eram globais. A seguir veremos como passar um elemento de uma estrutura para uma função. Quando uma estrutura é passada para uma função, ocorrem diversas mudanças na maneira pela qual um elemento da estrutura é referenciado.

Passando elementos para funções

Quando passamos um elemento de uma variável de estrutura para uma função, em verdade estamos passando o valor daquele elemento para a função (passagem por cópia). Portanto estamos passando uma variável simples, a menos é claro, que este elemento seja complexo como um vetor de caracteres. Por exemplo:

- func(mike.x) – passa o caractere contido em x;
- func2(mike.y) – passa o inteiro contido em y;
- func3(mike.z) – passa o valor de ponto flutuante contido em z;
- func4(mike.s) – passa o endereço do vetor s;
- func5(mike.s[4]) – passa o caractere contido na posição de índice 4 do vetor s.

```
34 struc fred{
35     char x;
36     int y;
37     float z;
38     char s[10];
39 }mike;
```

Entretanto se você quisesse passar o endereço de elementos de estrutura individuais, colocaria o operador & antes do nome da estrutura. Por exemplo, para passar o endereço dos elementos na estrutura **mike**, utilizamos os formatos:

- func(&mike.x) – passa endereço da variável caractere x;
- func2(&mike.y) – passa endereço da variável inteira y;
- func3(&mike.z) – passa o endereço da variável de ponto flutuante z;
- func4(&mike.s) – passa o endereço do vetor s ;
- func5(&mike.s[4]) – passa o endereço da posição de índice 4 do vetor s.

```
34 struc fred{
35     char x;
36     int y;
37     float z;
38     char s[10];
39 }mike;
```

Observe que o operador & precede nome da estrutura e não o nome do elemento individual.

Operador especial -> (operador-flecha)

A maioria dos programadores o chama de operador-flecha. É formado utilizando um sinal de menos seguido por um sinal de maior que o sinal -> é utilizado em lugar do operador de ponto (.), ao acessar um elemento de estrutura dentro de uma função. Exemplo:

(*t).horas é o mesmo que **t->horas**

O operador ponto é utilizado para acessar elementos de estrutura, quando a estrutura é global ou definida dentro a mesma função que o código a referência. Utilizamos -> para referenciar elementos de estrutura quando um ponteiro de estrutura foi passado para uma função.

Lembre-se também, de que **é necessário passar o endereço da estrutura para uma função utilizando o operador &**. Estruturas não são como vetores, que podem ser passados apenas como o nome do vetor.

Definindo novos nomes de tipos de dados

A Linguagem C permite que se defina novos nomes de tipos de dados, utilizando explicitamente a palavra chave **typedef**. Não se cria realmente uma nova classe de dado, mas é definindo um novo nome para um tipo de dado.

A forma geral da declaração **typedef** é:

typedef tipo nome;

onde:

- tipo: é qualquer tipo de dado permissível;
- nome: é o novo nome para este tipo.

O novo nome é definido em adição ao – e não em substituição do – tipo de nome existente. Por exemplo, podemos criar um novo nome para um tipo float utilizando:

```
51  typedef float balanco;
```

Esta declaração diz ao compilador para que reconheça **balanco** como outro nome de **float**. Em seguida é possível cria uma variável **float** utilizando **balanco**:

```
53  balanco salario;
```

Onde **salario** é uma variável de ponto flutuante do tipo **balanco**, que no entanto, é outra designação para **float**.

É possível utilizar **typedef** para criar nomes para tipos mais complexos também; por exemplo:

```
56  typedef struct strc_func{
57      int cod;
58      char nome[40];
59      char cargo[15];
60      float salario;
61  }FUNCIONARIO;
62
63  FUNCIONARIO empreg[100];//define um vetor de
64                  //estruturas empreg com 100 posições
65                    //do tipo FUNCIONARIO
```

Operador sizeof

O operador unário **sizeof** é utilizado para informar o tamanho de qualquer tipo de variável incluindo estruturas e variáveis definidas pelo programador. Por exemplo, na tabela abaixo temos os tamanhos dos tipos de dados para uma implementação C comum a muitos compiladores C:

Portanto, considerando o tamanho reservado aos tipos de variáveis da tabela 1:

```
43 char ch;
44 int i;
45 float f;
46
47 printf("%d", sizeof(ch));
48 printf("%d", sizeof(i));
49 printf("%d", sizeof(f));
```

Tipo	Tamanho em Bytes
char	2
int	4
long int	8
Float	16
Double	32

Tabela 1 – Bytes ocupados por tipos de variáveis

O fragmento de código acima imprimirá os números 2, 4 e 16 na tela, a partir do tamanho coletado por **sizeof**, das variáveis declaradas “ch”, “i” e “f”; **sizeof** é um *operador do momento de compilação*: toda a informação para computar o tamanho de qualquer variável é conhecida no momento da compilação.