Capítulo 4
Estruturas

We can´t solve the problem by using the same kind of thinking we used when we created them.

Albert Einstein

OBJETIVOS

- Entender tipos definidos pelos programadores.
- Aprender e utilizar estruturas.
- Saber como especificar estruturas e acesso a membros.
- Entender a relação entre estruturas e classes.

O Capítulo 3 apresentou um conjunto de estruturas de controle e decisão que você pode utilizar para definir o comportamento de programas. Adicionalmente, novos conceitos da linguagem C++ e mecanismos de controle de programa foram apresentados, permitindo tratar uma variedade de problemas que combinem o uso de operadores lógicos e relacionais, controlando a execução de programas. Neste capítulo, o foco está em explorar mecanismos para permitir criar e utilizar novos tipos de dados. Assim, questões que surgem englobam: o que faço, se precisar trabalhar com partes de uma figura geométrica como o raio ou o centro (isto é, as coordenadas x e y) de um círculo? Qual o mecanismo apropriado para acessar dados de uma figura geométrica como, por exemplo, o raio do círculo na solução de um problema que tenha em mãos?

Note que o interesse aqui recai em explorar mecanismos de novos tipos de dados. Responder às questões supracitadas compreende os propósitos deste capítulo e, para tanto, a apresentação é feita com o uso de exemplos ilustrando várias aplicações e mostrando como criar novos tipos de dados.

4.1. INTRODUÇÃO

Até aqui, temos visto variáveis de tipos de dados simples, tais como *int, char* e *float*. Variáveis desses tipos representam apenas um item de informação: uma altura, uma largura ou similar. Porém, assim como os empregados são organizados em departamentos e palavras são estruturadas em sentenças, é frequentemente conveniente organizar variáveis simples em entidades mais complexas. Em C++, a construção que permite fazer isto é denominada *estrutura*.

4.1.1. Estrutura

Estrutura é um tipo definido pelo usuário, em que sua declaração serve para definir as propriedades dos dados desse novo tipo. Uma estrutura é um conjunto de variáveis simples. As variáveis em uma estrutura podem ser de diferentes tipos: podem ser *int*, *float*, e assim por diante. Isso é diferente de um *array* (discutido no Capítulo 7), o qual tem todas as variáveis do mesmo tipo. Os itens de dados em uma estrutura são denominados *membros* da estrutura.

Na programação C++, estruturas são um dos elementos básicos para entender objetos e classes. Tipicamente, uma estrutura é uma coleção de dados, enquanto uma classe é uma coleção de dados e funções. Uma grande variedade de estruturas é apresentada neste capítulo, fazendo uso de exemplos. A seção seguinte apresenta as estruturas simples e ilustra o seu uso.

4.2. ESTRUTURAS SIMPLES

4.1.1. Estrutura

A linguagem C++ provê suporte para estruturas, as quais podem ser utilizadas quando se necessita criar um tipo de dado novo. Note que se trata de um tipo de dado definido pelo usuário ou programador, geralmente citado como *user-defined data type*. Esse tipo de dado novo que se pretende definir agrupa, de modo lógico, vários campos ou membros que fazem parte da estrutura que será criada.

Considere os exemplos apresentados a seguir:

```
struct ponto {
  double x;
  double y;
};
struct circulo {
  ponto centro;
  double raio;
```

```
  double área;
};
ponto p1, p2;
```

Nesse exemplo, dois novos tipos de dados foram definidos: ponto e círculo. Observe que, uma vez definido um tipo como ponto no exemplo anterior, pode-se utilizá-lo para declarar novas variáveis, como foi feito com o círculo. Para entender mais, vamos explorar um exemplo.

Praticando um Exemplo. Escreva um programa em C++ que solicite do usuário dia, mês e ano atual e, em seguida, o programa exibe a data informada. Para elaborar esse programa, você deve fazer uso de uma estrutura para criar um novo tipo, chamado de data. Os membros da estrutura (ou seja, dia, mês e ano) devem ser do tipo inteiro. Agora, feche o livro e tente implementar sua solução. Depois, consulte a solução do exemplo mostrada na Listagem 4.1.

```
1.  #include <iostream>
2.  using namespace std;
3.  // Programa para ilustrar o uso de estrutura
4.  struct data {
5.    int dia;
6.    int mes;
7.    int ano;
8.  };
9.  int main()
10. {
11.   data hoje;
12.   cout << "Digite o dia de hoje: ";
13.   cin >> hoje.dia;
14.   cout << "Digite o mes atual: ";
15.   cin >> hoje.mes;
16.   cout << "Digite o ano atual: ";
17.   cin >> hoje.ano;
18.   cout << "\nData de hoje informada: " << hoje.dia << " / "
      << hoje.mes << " / " << hoje.ano << endl << endl;
19.   system("PAUSE");
20.   return 0;
21. }
```

Listagem 4.1

O programa da Listagem 4.1 utiliza estrutura para definir um novo tipo de dado, chamado de data, conforme linhas 4-8. A estrutura contém três variáveis do

tipo inteiro. Essa estrutura representa um item de dado que chamamos de data, a qual engloba dia, mês e ano de uma data.

O programa da Listagem 4.1 especifica a estrutura data, define uma variável da estrutura chamada hoje, solicita que o usuário entre com os valores para seus membros (dia, mês e ano) e, finalmente, mostra a data de hoje informada. O programa da Listagem 4.1 tem três características: especificar uma estrutura, definir uma variável tipo estrutura (linha 11) e acessar os membros da estrutura (linha 18). Execute o programa e digite, por exemplo, os valores 25 (para dia), 12 (para mês) e 2009 (para ano) e, então, o programa exibirá a saída mostrada na Figura 4.1.

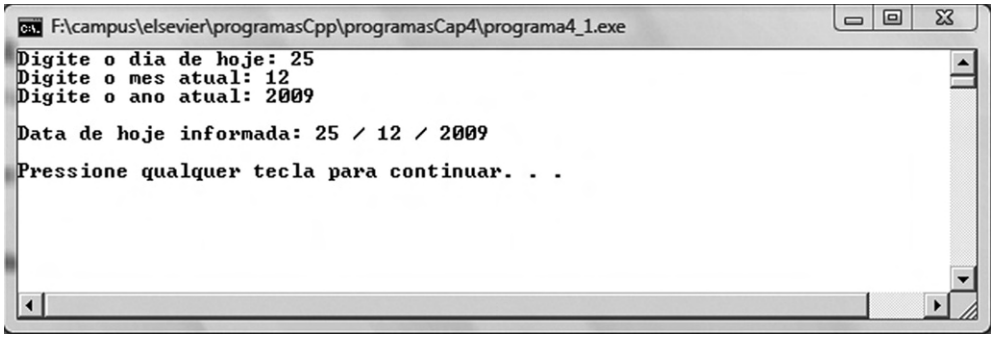

Figura 4.1 – Saída do programa da Listagem 4.1.

4.3. ESPECIFICAÇÃO DE ESTRUTURA

4.3.1. Definição de estrutura

A especificação de uma estrutura diz como a estrutura está organizada, ou seja, ela define os membros que a estrutura possui, conforme ilustrado a seguir.

```
struct data {  // especifica a estrutura
  int dia;     // dia da data
  int mes;     // mes da data
  int ano;     // ano da data
};
```

Perceba que a palavra *struct* introduz o especificador. Logo em seguida, vem o nome da estrutura ou *tag* (no exemplo: *data*). A declaração dos membros da estrutura (dia, mês e ano) fica entre chaves ({}). Observe que há um ponto-e-vírgula após o fecha-chaves (}), o que termina a estrutura. Note que isso é diferente de laços, decisões e funções, que não possuem ponto-e-vírgula após o fecha-chaves (}).

■ *É importante destacar que o especificador serve como um modelo ou template para a criação de variáveis do tipo data. O especificador não define qualquer variável, ou seja, ele não designa*

qualquer espaço de memória e nomeia uma variável. Ele serve apenas para especificar como devem ser as variáveis quando definidas.

4.3.2. Definição de uma Variável do Tipo Estrutura

A primeira instrução na Listagem 4.1 logo após a função *main()* é `data hoje;` que define uma variável, chamada de `hoje`, como sendo do tipo estrutura. Essa definição reserva espaço em memória para `hoje`. Quanto espaço? O suficiente para conter todos os membros da estrutura de `hoje`, ou seja, 4 bytes para cada *int* (*dia*, *mes* e *ano*).

É importante observar que podemos considerar a estrutura hoje como a especificação para um novo tipo de dados (o qual foi chamado de data). Além disso, perceba que um dos objetivos da linguagem C++ é tornar a sintaxe e a operação de tipos de dados definidos pelo usuário (ou programador) o mais similar possível aos tipos de dados predefinidos.

4.4. ACESSO A MEMBROS DE ESTRUTURA

4.4.1. Acesso a Membros de Estrutura

Após a variável de uma estrutura ter sido definida, seus membros podem ser acessados usando um operador do tipo ponto (ou *dot operator*). A seguir, é mostrado como os três membros da estrutura do programa da Listagem 4.1 são acessados, conforme a linha 18.

```
hoje.dia
hoje.mes
hoje.ano
```

Note que o membro da estrutura é escrito em três partes: o nome da variável da estrutura (`hoje`), o operador ponto (`.`) e o nome do membro, por exemplo, no primeiro membro (`dia`). Note que o primeiro elemento de uma expressão envolvendo o operador ponto é o nome da variável da estrutura (`hoje`, neste exemplo) e não o nome do especificador da estrutura (`data`).

4.4.2. Definição de Estruturas sem Rótulo (tag)

No programa da Listagem 4.1, você viu a especificação e a definição da estrutura como duas instruções separadas. Todavia, essas instruções podem ser combinadas em uma única instrução, conforme o exemplo da Listagem 4.2. Note que não existe uma instrução separada para definir a estrutura, como havia na Listagem 4.1. Executar o programa da Listagem 4.2 produz a mesma saída mostrada na Figura 4.1.

```
1.  #include <iostream>
2.  using namespace std;
3.  // Programa para ilustrar o uso de estrutura
4.  struct { // nao é feito uso de rótulo (ou tag)
5.    int dia;
6.    int mes;
7.    int ano;
8.  } hoje; // definicao da estrutura ocorre aqui
9.  int main()
10. {
11.   cout << "Digite o dia de hoje: ";
12.   cin >> hoje.dia;
13.   cout << "Digite o mes atual: ";
14.   cin >> hoje.mes;
15.   cout << "Digite o ano atual: ";
16.   cin >> hoje.ano;
17.   cout << "\nData de hoje informada: " << hoje.dia << " / "
      << hoje.mes << " / " << hoje.ano << endl << endl;
18.   system("PAUSE");
19.   return 0;
20. }
```

Listagem 4.2

■ *Observe que o rótulo (tag) da estrutura pode ser removido caso não haja mais variáveis para serem definidas. Combinar a especificação e a definição da estrutura, como feito na Listagem 4.2, é uma forma de economizar linhas de código. Todavia, é uma forma menos clara e menos flexível se comparada ao caso de ter especificação e definição separados (como na Listagem 4.1).*

4.4.3. Inicialização de Membros da Estrutura

Outro recurso que você tem na linguagem C++ é inicializar os membros de uma estrutura no momento em que a estrutura é definida. Além disso, também é possível ter mais de uma variável em uma estrutura. Para entender mais, nada melhor do que um exemplo.

Praticando um Exemplo. Escreva um programa em C++ que defina duas variáveis (`hoje` e `novaData`) do tipo data. Em seguida, o programa deve inicializar a variável `hoje`, exibir os valores dos membros da variável `hoje` e atribuir hoje `novaData`. Logo em seguida, o programa exibe os valores de `novaData`. Para elaborar esse programa, você deve fazer uso de uma estrutura para criar um novo tipo, chamado de data. Os membros dessa estrutura compreendem dia, mês e ano, os quais são do tipo inteiro. Agora, feche o livro e tente implementar sua solução. Depois, consulte a solução do exemplo mostrada na Listagem 4.3.

```
1.  #include <iostream>
2.  using namespace std;
3.  // Programa para ilustrar a inicializacao de estrutura
4.  struct data {
5.    int dia;
6.    int mes;
7.    int ano;
8.  };
9.  int main()
10. {
11.   data hoje = {25, 12, 2009};
12.   data novaData;
13.   cout << "\nData de hoje informada: " << hoje.dia << " / "
      << hoje.mes << " / " << hoje.ano << endl;
14.   novaData = hoje;
15.   cout << "\nNova data informada: " << hoje.dia << " / " <<
      hoje.mes << " / " << hoje.ano << endl << endl;
16.   system("PAUSE");
17.   Return 0;
18. }
```

Listagem 4.3

Se você executar o programa da Listagem 4.3, como a variável data foi inicializada no próprio código com os valores 15 (para dia), 10 (para mês) e 2009 (para ano), o programa exibirá a saída mostrada na Figura 4.2.

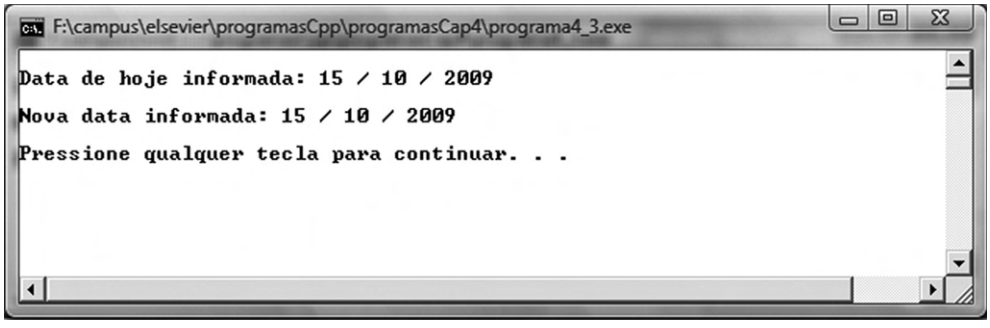

Figura 4.2 – Saída do programa da Listagem 4.3.

■ *Note que os valores atribuídos aos membros da estrutura data vêm entre {} e separados por vírgulas. Além disso, uma variável do tipo estrutura pode ser feita igual à outra variável definida pela mesma estrutura, conforme a linha 11 da Listagem 4.3.*

Praticando um Exemplo. Escreva um programa em C++ que defina uma estrutura que tenha dois membros variáveis (`metro` e `centímetro`) do tipo `medida`. Em seguida, você deve criar outra estrutura de nome terreno, a qual também tem dois membros, `largura` e `comprimento`. O objetivo do programa é determinar a área de um terreno e, para tanto, as medidas devem estar numa única unidade (no caso, o metro) para expressar a área do terreno. Para elaborar esse programa, você deve fazer uso de uma estrutura para criar um novo tipo, chamado de medida. Os membros dessa estrutura compreendem `metro` e `centímetro`, os quais são do tipo double. Você também deve criar a estrutura terreno, que terá membros do tipo medida. Agora, feche o livro e tente implementar sua solução. Depois, consulte a solução do exemplo mostrada na Listagem 4.4.

O que podemos mais fazer com estruturas? É possível ter uma estrutura dentro de outra? Para responder a essas e outras questões, nada melhor do que explorar outro exemplo.

```
1.  #include <iostream>
2.  using namespace std;
3.  // Programa para ilustrar a criacao de uma estrutura dentro
    de outra
4.  struct medida {
5.    float metro;
6.    float centimetro;
7.  };
8.  struct terreno {
9.    medida comprimento; // comprimento do terreno
10.   medida largura; // largura do terreno
11. };
12. int main()
13. {
14.   terreno meuTerreno;
15.   meuTerreno.comprimento.metro = 30; // definicao de valores
16.   meuTerreno.comprimento.centimetro = 50;
17.   meuTerreno.largura.metro = 60;
18.   meuTerreno.largura.centimetro = 90;
19.   // converte valores para metros
20.   float comprimentoTerreno = meuTerreno.comprimento.metro +
      meuTerreno.comprimento.centimetro/100;
21.   float larguraTerreno = meuTerreno.largura.metro + meuTerreno.
      largura.centimetro/100;
22.   cout << "\nArea do terreno: " << comprimentoTerreno << " *
      " << larguraTerreno << " = " << comprimentoTerreno * lar-
      guraTerreno << " metros quadrados\n\n";
```

```
23.   system("PAUSE");
24.   return 0;
25. }
```

Listagem 4.4

Se você executar o programa da Listagem 4.4, a variável meuTerreno, declarada como do tipo terreno na linha 14, tem seus membros inicializados nas linhas 15-18 com os valores 30 (para a porção metro do comprimento), 50 (para a porção centímetro da comprimento), 60 (para a porção metro da largura) e 90 (para a porção centímetro da largura) e, então, o programa exibirá a saída ilustrada na Figura 4.3.

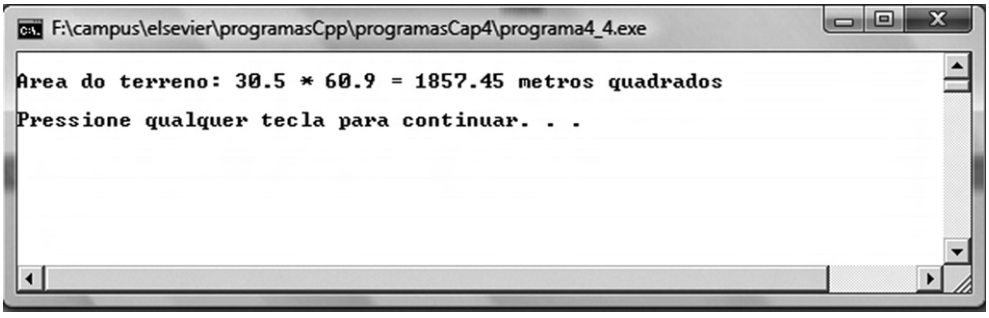

Figura 4.3 – Saída do programa da Listagem 4.4.

No programa da Listagem 4.4, a estrutura medida tem dois membros: `metro` e `centímetro`. As duas variáveis podem ter uma parte fracional e, portanto, *float* é usado. O programa inicializa os valores do membros nas linhas 15-18. Depois, é feita a multiplicação de `comprimentoTerreno por larguraTerreno` (obtidos nas linhas 20 e 21 a partir da conversão para metros) e, finalmente, a área do terreno é calculada e apresentada em metros quadrados na linha 22.

Observe que você não pode diretamente multiplicar ou adicionar duas medidas, como ilustrado nas instruções a seguir.

```
d3 = d1 * d2;
d5 = d3 + d4;
```

Você não pode fazer qualquer operação diretamente como ilustrado. Isso não é possível porque não existe qualquer procedimento predefinido na linguagem C++ que informe como multiplicar ou somar duas variáveis do tipo medida. Os operadores + e * funcionam apenas para os tipos de dados predefinidos, como *float* e *int*, e não com tipos de dados definidos pelo usuário ou programador.

■ *Os operadores * e + funcionam apenas para tipos predefinidos na linguagem, como float e int, e não com tipos de dados definidos pelo usuário ou programador. Todavia, quando você estudar classes (no Capítulo 6) verá como operações como essa e outras podem ser executadas com tipos definidos pelo usuário.*

4.4.4. Estruturas Dentro de Estruturas

Observando a Listagem 4.4, você verifica que é possível aninhar uma estrutura dentro de outra. Como isso ocorre? Considere a Listagem 4.4, onde criamos uma estrutura de dados que armazena as medidas de um terreno qualquer. Então, é feito uso de duas variáveis do tipo *medida* como as variáveis *comprimento* e *largura*.

O programa da Listagem 4.4 define uma única variável (`meuTerreno`), que é do tipo terreno. Em seguida, são atribuídos valores para vários membros dessa estrutura. Devido ao fato de que essa estrutura é aninhada dentro de outra, precisamos usar o operador ponto (.) duas vezes para ter acesso aos membros da estrutura. Assim, temos:

```
meuTerreno.comprimento.metro = 30;
```

Nessa instrução, `meuTerreno` é o nome da variável da estrutura, declarada na linha 14, `comprimento` é o nome de um membro da estrutura mais externa (terreno) e `metro` é o nome da estrutura mais interna (`metro`). Depois que os valores são atribuídos aos membros de `meuTerreno`, o programa calcula a área do terreno. Para achar a área, o programa converte o comprimento e a largura das variáveis do tipo `medida` para variáveis do tipo *float* `comprimentoTerreno` e `larguraTerreno`, representando as medidas em metros. Os valores `comprimentoTerreno` e `larguraTerreno` são encontrados somando-se o membro metro de medida com o membro `centímetro` multipicado por 100.

Agora, surge uma questão interessante: como se pode inicializar uma variável de estrutura que contém outra(s) estrutura(s)? A seguinte instrução inicializa a variável `meuTerreno` para os mesmos valores que lhe são atribuídos no programa da Listagem 4.4.

```
meuTerreno = { {30, 50}, {60, 90} };
```

4.4.5. Tipos de Dados Enumerados (Enumerated Data Types)

Como visto anteriormente, estrutura é uma maneira através da qual o usuário pode definir seus próprios tipos de dados. Outra abordagem é usar tipos de dados enumerados. Tipos enumerados são usados quando conhecemos antecipadamente uma lista (geralmente pequena) de valores finitos que um tipo de dado pode assumir. Para entender mais, vamos ver um exemplo.

Praticando um Exemplo. Escreva um programa que permita definir um conjunto de valores (dias da semana) e, dependendo dos valores inicializados, informa quantos dias há entre esses dias e a ordem na qual os dias acontecem. Para elaborar esse programa, você deve fazer uso de enum para criar um novo tipo, chamado diasSemana. Agora, feche o livro e tente implementar sua solução. Depois, consulte a solução do exemplo mostrada na Listagem 4.5.

```
1.  #include <iostream>
2.  using namespace std;
3.  // Programa para ilustrar o uso tipo de dado enumerado
4.  int main()
5.  {
6.    enum diasDaSemana { Seg, Ter, Qua, Qui, Sex, Sab, Dom };
7.    diasDaSemana dia1, dia2;
8.    dia1 = Seg;
9.    dia2 = Qua;
10.   int diferencaDias = dia2 - dia1;
11.   cout << "Existe " << diferencaDias << " dias entre Seg e
      Qua" << endl;
12.   if(dia1 < dia2)
13.    cout << "\nDia1 acontece antes de dia2\n\n";
14.   else
15.    cout << "\nDia2 acontece antes de dia1\n\n";
16.   system("PAUSE");
17.   return 0;
18. }
```

Listagem 4.5

Se você executar o programa da Listagem 4.5, como as variáveis `dia1` e `dia2` foram inicializadas com `Seg` e `Qua` nas linhas 8 e 9, o programa exibirá a saída ilustrada na Figura 4.4.

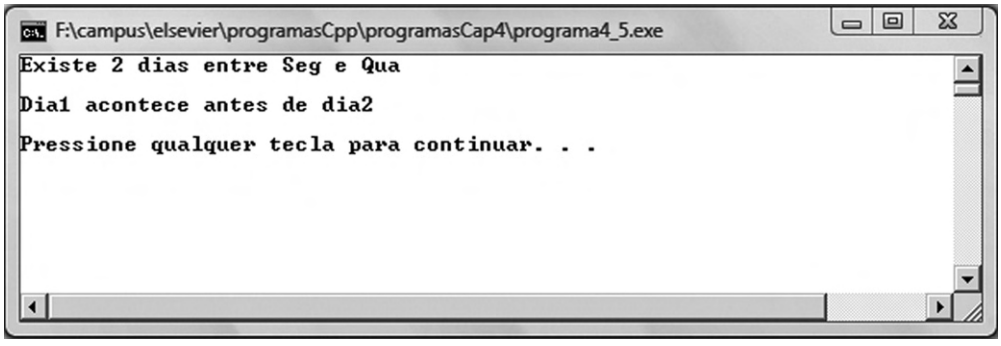

Figura 4.4 – Saída do programa da Listagem 4.5.

4.4.6. Especificador enum

O especificador *enum* define o conjunto de todos os nomes que serão os valores possíveis desse tipo. Tais valores são chamados de *membros*. O tipo enum `diasDaSemana` tem sete membros (Seg, Ter,..., Sab). *Enumerado* vem do fato de que todos os valores são listados. Usando *enum*, você está atribuindo um nome específico para cada valor possível.

É importante salientar que se pode usar operadores aritméticos com tipos *enum*. Da mesma forma, pode-se utilizar operadores de comparação, conforme mostrado na Listagem 4.5. Deve-se observar que tipos de dados enumerados são tratados como inteiros. Daí o fato de podermos usar operadores aritméticos e de comparação. Assim, o primeiro nome da lista recebe o valor 0, o seguinte 1, e assim por diante. Para entender mais, nada melhor do que um exemplo.

Praticando um Exemplo. Escreva um programa que permita definir e utilizar um tipo que se pode chamar de meuBoolean no qual se definirão os valores falso e verdadeiro utilizando enum. O programa deve solicitar que o usuário digite, por exemplo, uma frase e depois deve exibir a quantidade de palavras digitadas. Para tanto, você deve utilizar um contador de palavras. Pode utilizar a função getche(), que lê dados digitados até que você tecle Enter. Note que, para elaborar esse programa, você deve fazer uso de enum para criar um novo tipo, chamado de meuBoolean. Agora, feche o livro e tente implementar sua solução. Depois, consulte a solução do exemplo mostrada na Listagem 4.6.

```
1.  #include <iostream>
2.  #include <conio.h> // para funcao getche()
3.  using namespace std;
4.  // Programa para ilustrar o uso de tipo de dado enumerado
5.  int main()
6.  {
7.    enum meuBoolean {falso, verdadeiro};
8.    meuBoolean palavra = falso; // falso quando encontrar espaco
      em branco
9.    char ch = 'x'; // ch = caractere digitado pelo usuario
10.   int contadorPalavra = 0; // contador de palavras
11.   cout << "Digite uma frase: ";
12.   do
13.   {
14.    ch = getche(); // ler dados digitados pelo usuario
15.    if(ch==' ' || ch=='\r') { // testa se espaço em branco
16.      if( palavra ) { // se palavra
17.      contadorPalavra++; // entao, conta (incrementa) conta-
         dorPalavra
```

```
18.     palavra = falso; // reinicializa palavra como falso
19.    }
20.   }
21.   else
22.   if(!palavra )
23.    palavra = verdadeiro;
24.   } while( ch!= '\r' ); // finaliza programa
25.
26.   cout << "\nNumero de palavras contadas: " << contadorPalavra
      << endl << endl;
27.   system("PAUSE");
28.   return 0;
29. }
```

Listagem 4.6

■ *Note que, quando você utilizar o tipo enum, pode se deparar com uma situação na qual deseja que o primeiro elemento do tipo enum não inicie com o valor 0, mas com outro valor, como, por exemplo, 3. Nesse caso, você simplesmente faz:*

enum diasDaSemana { Seg = 3, Ter, Qua, Qui, Sex, Sab, Dom };

Fazendo isso, os elementos subsequentes recebem os valores 4, 5, e assim por diante.

Note que o programa da Listagem 4.6 conta a quantidade de vezes que uma palavra string ocorre, contando o número de espaços entre strings, testado na linha 15.

Observe que o programa fica num loop do, lendo os caracteres do teclado, e toda vez que ele encontra um espaço ele contabiliza uma palavra. Depois, o programa ignora novos espaços, se existirem, até encontrar um novo caractere, indo novamente, a partir desse ponto, esperar por um novo espaço em branco para contabilizar a segunda palavra. Esse processo se repete até você teclar Enter, saindo do laço. Se executar o programa da Listagem 4.6, o programa exibirá a saída ilustrada na Figura 4.5.

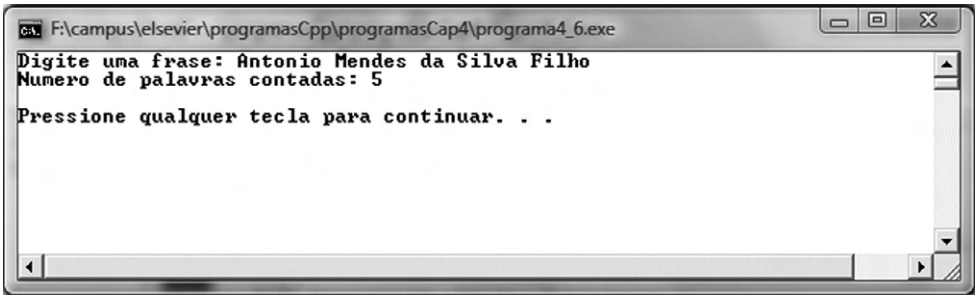

Figura 4.5 – Saída do programa da Listagem 4.6.

Para entender mais, nada melhor do que explorar um exemplo.

Praticando um Exemplo. Modifique o programa da Listagem 4.5 de modo que permita definir o conjunto de valores dos dias da semana, inicializando Seg = 3 e, dessa forma, os valores subsequentes devem receber 4, 5, 6, e assim por diante. Depois, o programa deve exibir os dias e respectivos valores. Para elaborar esse programa, você deve fazer uso de enum para criar um novo tipo, chamado diasDaSemana. Agora, feche o livro e tente implementar sua solução. Depois, consulte a solução do exemplo mostrada na Listagem 4.7.

```
1.  #include <iostream>
2.  using namespace std;
3.  // Programa para ilustrar o uso de tipo de dado enumerado
4.  int main()
5.  {
6.    enum diasDaSemana { Seg = 3, Ter, Qua, Qui, Sex, Sab, Dom };
7.
8.    cout << "Lista de dias e valores inicializados:\n\n";
9.    cout << "Seg = " << Seg << endl;
10.   cout << "Ter = " << Ter << endl;
11.   cout << "Qua = " << Qua << endl;
12.   cout << "Qui = " << Qui << endl;
13.   cout << "Sex = " << Sex << endl;
14.   cout << "Sab = " << Sab << endl;
15.   cout << "Dom = " << Dom << endl << endl;
16.   system("PAUSE");
17.   return 0;
18. }
```

Listagem 4.7

Se você executar o programa da Listagem 4.7, com o tipo enum contendo sete elementos e declarado como

```
enum diasDaSemana { Seg = 3, Ter, Qua, Qui, Sex, Sab, Dom };
```

o programa exibirá a saída ilustrada na Figura 4.6.

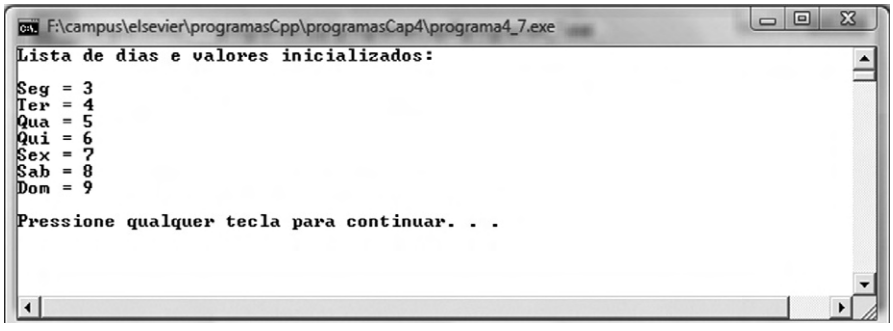

Figura 4.6 – Saída do programa da Listagem 4.7.

4.5. ESTRUTURAS E CLASSES

4.5.1. Estruturas

Estruturas em C++, bem como na linguagem C, foram inicialmente utilizadas com o objetivo de agrupar vários membros de dados (que podem ser de tipos diferentes) para realizar determinada funcionalidade, conforme visto anteriormente no capítulo. (O leitor pode consultar as seções 4.2 e 4.3.)

4.5.2. Classes

Por outro lado, uma classe é uma coleção de objetos que possuem as mesmas propriedades. Em outras palavras, classe é o conjunto de objetos que possuem o mesmo comportamento.

Uma classe serve como um modelo ou *template* para definir as características dos objetos. O termo classe é, na realidade, uma abreviação do termo classe de objetos. Em C++, uma classe tem como componentes principais os dados e as funções-membros.

4.5.3. Diferenças entre Estruturas e Classes

Diferenças entre estruturas e classes compreendem a definição de uma estrutura, na qual você utiliza a palavra reservada *struct*, enquanto na definição de uma classe usa a palavra *class*.

Além disso, todos os membros de uma estrutura são de natureza pública (por default). Já em uma classe, todos os membros são privados. Vale ressaltar que classe em C++ é uma extensão de estrutura em C. Dessa forma, em C++, você pode definir uma estrutura como:

```
struct exemplo {
  int a;       // dados publicos (por default)
  int b;
  float c;
};]
```

A estrutura desse exemplo pode ser descrita como uma classe, conforme mostrado a seguir.

```
class exemplo {
  private int a;      // dados privados (por default)
  private int b;
  private int c;
};
```

RESUMO

Neste capítulo, você aprendeu que pode definir novos tipos além dos tipos predefinidos. Isso permite definir tipos como, por exemplo, data que consiste em três partes (membros): dia, mês e ano. Em C++, a construção que faz isso é chamada de *estrutura*. Uma estrutura compreende um conjunto de variáveis simples. Note que isso é diferente de um *array* (discutido no Capítulo 7), o qual tem todas as variáveis do mesmo tipo. Você também teve a oportunidade de entender as diferenças entre estruturas e classes (que constitui a base do paradigma da programação orientada a objetos). Cabe destacar que novos conceitos foram apresentados, e diversos exemplos foram realizados ilustrando o uso de estruturas. No próximo capítulo, você irá estudar e explorar funções, e saber como elas podem ser empregadas na programação orientada a objetos.

QUESTÕES

1. O que são estruturas? Apresente situações em que o uso de estruturas é apropriado.

2. É possível uma estrutura conter outra estrutura? Em caso afirmativo, apresente exemplos onde você poderia fazer isso.

3. Qual a diferença entre estrutura e tipo de dado enumerado? Use um exemplo para ilustrar sua resposta.

4. Qual a diferença entre estrutura e classe? Use um exemplo para ilustrar sua resposta.

■ *Note que a principal diferença entre estruturas e classes é a visibilidade dos membros que (por default) são de natureza pública nas estruturas e privada nas classes. Além disso, as classes também possuem funções membros, construtores e destrutores, conforme apresentado no Capítulo 6.*

EXERCÍCIOS

1. Faça uma pesquisa visando responder à seguinte questão: é possível uma estrutura conter um ponteiro (apontando) para ela mesma? Apresente um exemplo para ilustrar sua resposta.

2. Escreva um programa em C++ que permita a soma de números complexos (que contêm parte real e imaginária).

3. Escreva um programa em C++ que permita você com as coordenadas de dois pontos (p1 e p2) e que permita obter a soma p1 + p1.

4. Escreva um programa em C++ criando uma estrutura com membros hora, minuto e segundo, e que permita entrar com valores de tempo t e t2 e efetuar a soma t1 + t2.