

ou DVDs de filmes em uma caixa de filmes, e assim por diante. De maneira similar, em uma linguagem de programação, temos necessidade de agrupar itens de dados do mesmo tipo quando tentamos resolver um problema. A entidade que permite agrupar dados do mesmo tipo é denominada array. Os arrays podem conter poucos ou milhares de itens de dados e até mesmo ser tipos de dados definidos pelo próprio usuário (ou programador) como estruturas e objetos.

7.1.2. Definição de Arrays

Costuma-se definir e entender um array como um conjunto ou grupo de variáveis do mesmo tipo. Esse grupo de variáveis compartilha o mesmo nome (que é o nome do array) e é do mesmo tipo.

7.1.3. Diferença entre Arrays e Estruturas

Os arrays são como estruturas, desde que ambos agrupam uma quantidade de itens de dados em uma entidade maior. Contudo, as estruturas agrupam itens de tipos de dados diferentes, enquanto um array agrupa itens do mesmo tipo. Além disso, os itens (membros) em uma estrutura são acessados por nome, ao passo que os itens em um array são acessados por índice.

Usar um índice para especificar um item permite fácil acesso a uma quantidade maior de itens. Dada a importância desse tópico, o assunto será estudado neste capítulo juntamente com a apresentação de array de objetos e strings.

Os arrays são uma das estruturas de dados mais utilizadas na solução de problemas. Você pode utilizar os arrays em programas com o objetivo de armazenar dados que serão processados ou utilizados posteriormente. A maioria das linguagens de programação, dentre elas C++ e C, oferece suporte ao uso de arrays.

7.2. PRINCÍPIOS DE USO DE ARRAYS

Na solução de problemas que possa ter em mãos, você pode coletar em um experimento um conjunto de medições de temperatura. Em outra situação, você pode querer armazenar as notas de provas de alunos de um concurso ou vestibular para, em seguida, determinar a média das notas. Nessas e outras situações, você pode fazer uso de arrays.

De modo geral, a sintaxe para declarar um array unidimensional é:

```
tipoDado nomeArray[numeroElementos]
```

7.2.1. Regras para Declaração de Array

Na declaração de um array, contudo, você deve estar atento às seguintes regras:

- Você deve especificar o tamanho do array (N), que compreende o número de elementos que o array possui. Lembre-se de que o intervalo válido de índices se estende de 0 a $N - 1$, onde N é o número de elementos.
- Em C++, o limite inferior de um array é definido como 0 e não é permitido alterar esse valor (de limite inferior).

Veja a seguir exemplos de declaração de arrays:

```
double notas[60]; // array de tamanho 60 de notas de alunos
int numeroFaltas[60]; // array de tamanho 60 de frequência de alunos
```

Para entender melhor, vamos examinar um exemplo de uso de arrays.

Praticando um Exemplo. Escreva um programa em C++ que cria um array de double de tamanho 4. Chame esse array de nota[4]. Adicione também ao seu código um laço for para que possa ler as quatro notas. Além disso, você precisa acumular os valores das notas digitadas para que possa calcular a média depois. Após ler as quatro notas, seu programa deve exibir as notas digitadas e a média dessas notas. Para elaborar esse programa, defina tanto o array nota[4] quanto a variável total como sendo do tipo double. Agora, feche o livro e tente implementar sua solução. Depois, consulte a solução do exemplo mostrada na Listagem 7.1.

```
1. #include <iostream>
2. using namespace std;
3. // Programa para ilustrar uso de array
4.
5. int main()
6. {
7.     double nota[4]; // array de tamanho 4 para notas
8.     double total = 0;
9.
10.    for(int j=0; j<4; j++) // laço para leitura de notas
11.    {
12.        cout << "\nDigite uma nota: ";
13.        cin >> nota[j]; // acesso ao elemento do array
14.        total = total + nota[j];
15.    }
16.    for(int j=0; j<4; j++) // exibe notas digitadas
17.        cout << "\nVoce digitou a nota[" << j << "] = " << nota[j];
18.    cout << "\n\nMedia da turma = " << total / 4 << endl <<
        endl; // exibe media das notas
19.    system("PAUSE");
20.    return 0;
21. }
```

Listagem 7.1

O programa da Listagem 7.1 define um array unidimensional de notas de tamanho 4 e solicita ao usuário digitar valores de notas. Após digitar quatro valores de notas e teclar Enter, o programa lista as notas digitadas e, logo em seguida, a média dos valores entrados. Execute o programa e, depois, o programa exibirá a saída mostrada na Figura 7.1.

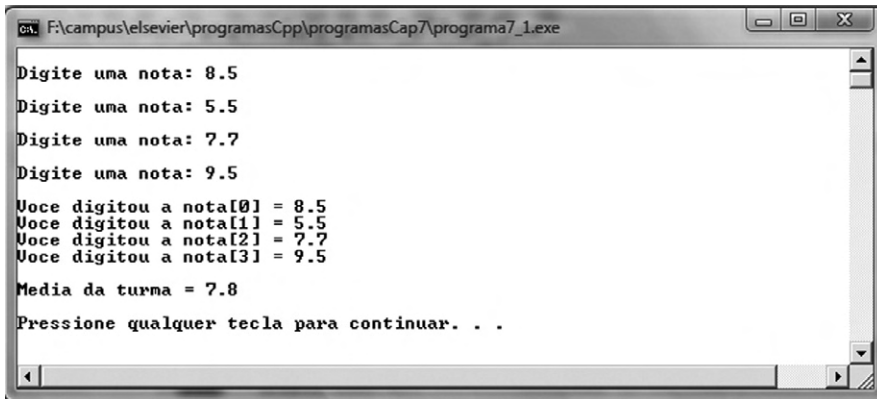


Figura 7.1 – Saída do programa da Listagem 7.1.

Como outras variáveis de C++, um array precisa ser definido (como na linha 7) antes de ser usado na linha 13, quando o array `nota[]` recebe um valor digitado pelo usuário. Além disso, é preciso informar o tamanho dele, bem como nome e tipo de dado, como ocorre na linha 7. Observe que todos os elementos ou itens do array são do mesmo tipo (`double`). Vamos examinar outro exemplo.

Praticando um Exemplo. Modifique o programa anterior de modo que o array de notas[] a ser criado tenha tamanho 60 do tipo `double`. Chame esse array de `nota[4]`. Agora, você deve também adicionar uma estrutura de controle, de modo a controlar quantas notas o usuário deseja entrar. Como o tamanho do array é 60, o usuário pode em princípio digitar até 60 notas. Entretanto, se o usuário digitar ‘-1’ (isto é, uma sentinela), o programa deixará de solicitar novas entradas de notas e exibirá a relação de notas digitadas e a média delas. Para elaborar esse programa, defina tanto o array `nota[60]` quanto a variável `total` como sendo do tipo `double`. Agora, feche o livro e tente implementar sua solução. Depois, consulte a solução do exemplo apresentada na Listagem 7.2.

1. `#include <iostream>`
2. `using namespace std;`
3. `// Programa para ilustrar uso de array`
- 4.
5. `const int MAX = 60; numero de elementos do array`
- 6.

```

7.  int main()
8.  {
9.      int n = 0; // contador de notas digitadas
10.     double nota[MAX]; // array de tamanho 60 para notas
11.     double total = 0;
12.
13.     for(int j = 0; j < MAX; j++) // laço para leitura de notas
14.     {
15.         cout << "\nDigite uma nota ou -1 para encerrar: ";
16.         cin >> nota[j]; // acesso ao elemento do array
17.         if (nota[j] != -1) // testa se usuario digitou -1
18.         {
19.             total = total + nota[j];
20.             n = n + 1;
21.         }
22.         else
23.             break;
24.     }
25.     for(int j = 0; j < n; j++) // exibe notas digitadas
26.         cout << "\nVoce digitou a nota[" << j << "] = " << nota[j];
27.     cout << "\n\nMedia das notas digitadas = " << total / n <<
        endl << endl; // exibe media das notas
28.     system("PAUSE");
29.     return 0;
30. }

```

Listagem 7.2

O programa da Listagem 7.2 define um array unidimensional de notas de tamanho 60 e permite que o usuário possa digitar até 60 notas. Entretanto, o usuário pode optar por digitar ‘-1’ caso queira encerrar antes. Execute o programa digitando três notas e -1; depois, o programa exibirá a saída mostrada na Figura 7.2.

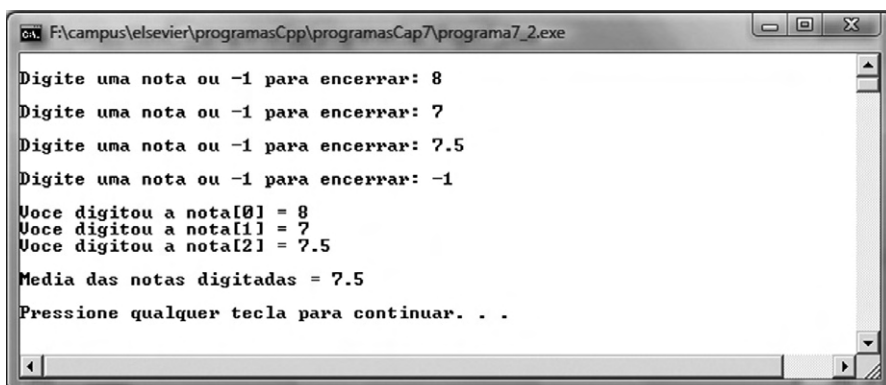


Figura 7.2 – Saída do programa da Listagem 7.2.

No programa da Listagem 7.2, há um detalhe adicional: o uso de uma variável *const* para o tamanho do array e para o valor limite do loop, como ocorre na linha 5.

■ *Note que usar uma variável (em vez de um número) torna mais fácil mudar o tamanho do array. Nesse caso, apenas uma linha de código precisa ser mudada para alterar o tamanho do array, o limite do loop e qualquer outro local onde o tamanho do array possa aparecer no programa.*

7.3. DECLARAÇÃO, INICIALIZAÇÃO E ACESSO DE ARRAYS

7.3.1. Declaração de Array

Para fazer uma declaração de um array unidimensional, você deve utilizar a sintaxe a seguir:

```
tipoDado nomeArray[numeroElementos]
```

Por outro lado, se você tiver necessidade de usar um array multidimensional, como, por exemplo, um array bidimensional ou matriz, deve seguir a sintaxe a seguir:

```
tipoDado nomeArray[numeroElementos_1][numeroElementos_2]
```

Note que você deve também seguir as regras de declaração apresentadas na seção anterior.

7.3.2. Inicialização de Array

A linguagem C++ permite inicializar arrays de maneira fácil. Para tanto, você deve colocar entre chaves ({}) um conjunto de valores que estariam inicializando os elementos de um array. Considere, por exemplo, que você queira inicializar um array de inteiros de tamanho 5 com os valores 11, 22, 33, 44 e 55. Em tal situação, deve usar a sintaxe a seguir:

```
int nomeArray[5] = {11, 22, 33, 44, 55};
```

Agora, para entender mais, vejamos o exemplo seguinte, que ilustra como você pode inicializar os elementos de um array.

Praticando um Exemplo. Elabore um programa C++ que cria e inicializa um array de inteiros de tamanho 20. Chame esse array de `numeroFaltas[20]`. Agora, você deve atribuir valores aos 20 elementos do array. Para tanto, deve colocar uma lista de valores entre chaves ({}) e separados por vírgula. Esses valores que você vai colocar na lista representam a quantidade de faltas de cada um dos 20 alunos. Depois, o programa solicita que o usuário selecione um número de aluno para exibir o número de faltas desse aluno e, em seguida, é feito o somatório do número de faltas, sendo obtidos o número total de faltas e a quantidade média. Agora, feche o livro e tente implementar sua solução. Depois, consulte a solução do exemplo apresentada na Listagem 7.3.

```

1. #include <iostream>
2. using namespace std;
3. // Programa para ilustrar inicializacao de array
4.
5. int main()
6. {
7.     int i; // contador de notas digitadas
8.     int totalFaltas = 0;
9.     int numeroFaltas[20] = { 15, 10, 12, 0, 12, 9, 6, 6, 3,
10.    9,12, 12, 15, 3, 6, 15, 15, 0, 12, 9 };
11.     cout << "\nEscolha um aluno: "; // escolha de um numero de
        aluno
12.     cin >> i;
13.     cout << "\nO " << i << "o. aluno tem " << numeroFaltas[i]
        << " faltas";
14.
15.     for(int j = 0; j < 20; j++) // totaliza numero de faltas
16.         totalFaltas = totalFaltas + numeroFaltas[j];
17.     cout << "\n\nNumero total de faltas da turma = " << total-
        Faltas;
18.     cout << "\nMedia do numero de faltas da turma = " << (dou-
        ble)totalFaltas / 20;
19.     cout << endl << endl;
20.     system("PAUSE");
21.     return 0;
22. }

```

Listagem 7.3

O programa da Listagem 7.3 inicializa um array unidimensional de número de faltas de tamanho 20 com valores predefinidos. Executando o programa e selecionando o aluno de número 5, você obterá a saída mostrada na Figura 7.3.

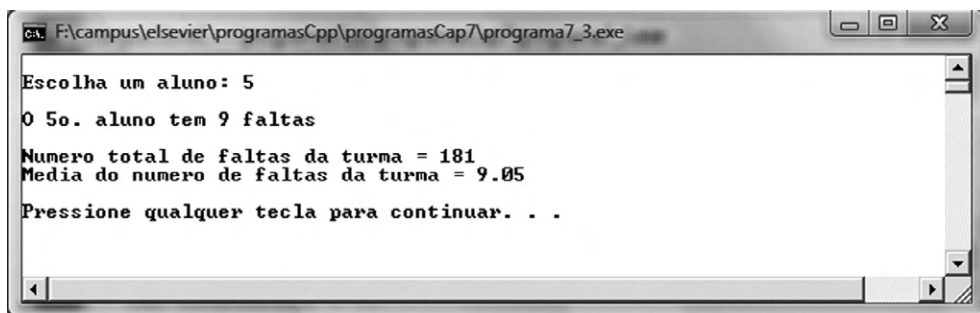


Figura 7.3 – Saída do programa da Listagem 7.3.

■ É importante observar que, se a lista de valores que está inicializando o array, tem menos itens do que a quantidade de elementos do vetor, os demais são inicializados com o valor 0 pelo compilador. Por outro lado, se houver mais valores do que o tamanho do array, o compilador sinaliza um erro.

O programa da Listagem 7.3 processa os valores dos array, calculando o número total e a média de faltas. Observe que os valores que inicializam o array *numeroFaltas* são colocados entre chaves e separados por vírgula. Nesse caso, quando inicializamos todos os valores do array, não é necessário colocar o tamanho do array, pois o compilador conta os valores de inicialização.

Agora, o que acontecerá se você definir o tamanho do array mas esse tamanho for diferente do número de elementos inicializados? Se existem menos elementos, aqueles inexistentes serão completados com 0. Todavia, se houver em excesso, um erro será sinalizado.

7.3.3. Arrays de Múltiplas Dimensões

Até aqui, temos visto arrays de uma dimensão (ou seja, uma única variável especifica cada elemento do array). Todavia, podemos ter arrays com mais de uma dimensão. A seguir, vamos examinar um exemplo de programa que solicita e armazena os valores de despesas de uma empresa por trimestre no período de dois anos.

Praticando um Exemplo. Elabore um programa C++ que cria um array bidimensional com valores do tipo double. Chame esse array de *despesas[ANO][TRIMESTRE]*. Adicionalmente, você deve definir ANO=2 e TRIMESTRE=4. Em seguida, deve criar um laço que solicite ao usuário digitar os valores de despesas para cada um dos semestres nos dois anos. Esses valores digitados devem ser armazenados no array que você criou. Ao final, o programa exibirá uma tabela que mostra os valores das despesas de cada ano, sendo listadas por trimestre, além do valor total de despesas nos dois anos. Agora, feche o livro e tente implementar sua solução. Depois, consulte a solução do exemplo apresentada na Listagem 7.4.

```
1. #include <iostream>
2. #include <iomanip>
3. const int ANO = 2; // tamanho do array
4. const int TRIMESTRE = 4;
5. using namespace std;
6. // Programa para ilustrar array bidimensional
7.
8. int main()
9. {
10.     int a, t;
11.     double despesas[ANO][TRIMESTRE]; // declaracao de array
12.     double total = 0.0;
13.
14.     for(a = 0; a < ANO; a++) // laço para obter valores
```

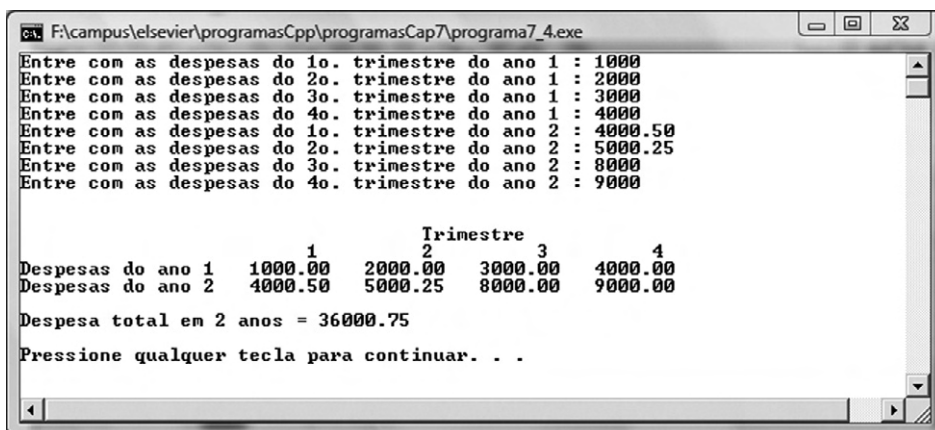
```

15.   for(t = 0; t < TRIMESTRE; t++)
16.   {
17.       cout << "Entre com as despesas do " << t + 1 << "o. tri-
           mestre do ano " << a + 1 << ": ";
18.       cin >> despesas[a][t]; // insercao de dados no array
19.       total = total + despesas[a][t];
20.   }
21.   cout << "\n\n Trimestre\n";
22.   cout << " 1 2 3 4";
23.   for(a = 0; a < ANO; a++)
24.   {
25.       cout << "\nDespesas do ano " << a+1;
26.       for(t = 0; t < TRIMESTRE; t++) // laço para exibir dados
27.           cout << setiosflags(ios::fixed)
28.               << setiosflags(ios::showpoint) // exibe ponto decimal
29.               << setprecision(2) // define precisao
30.               << setw(10)
31.               << despesas[a][t]; // exibe dado do array
32.   }
33.   cout << << "\n\nDespesa total em 2 anos = " << total;
34.   cout << endl << endl;
35.   system("PAUSE");
36.   return 0;
37. }

```

Listagem 7.4

O programa da Listagem 7.4 cria um array bidimensional que serve para armazenar valores de despesas de uma empresa e solicita que o usuário digite valores de despesas por trimestre (ou seja, valores como 1.000, 4.000,50 e assim por diante). Executando o programa, você obterá a saída mostrada na Figura 7.4.



```

F:\campus\elsevier\programasCpp\programasCap7\programa7_4.exe
Entre com as despesas do 1o. trimestre do ano 1 : 1000
Entre com as despesas do 2o. trimestre do ano 1 : 2000
Entre com as despesas do 3o. trimestre do ano 1 : 3000
Entre com as despesas do 4o. trimestre do ano 1 : 4000
Entre com as despesas do 1o. trimestre do ano 2 : 4000.50
Entre com as despesas do 2o. trimestre do ano 2 : 5000.25
Entre com as despesas do 3o. trimestre do ano 2 : 8000
Entre com as despesas do 4o. trimestre do ano 2 : 9000

Despesas do ano 1      1      2      3      4
                    1000.00 2000.00 3000.00 4000.00
Despesas do ano 2      4000.50 5000.25 8000.00 9000.00

Despesa total em 2 anos = 36000.75
Pressione qualquer tecla para continuar. . .

```

Figura 7.4 – Saída do programa da Listagem 7.4.

No programa anterior, a instrução `double despesas[ANO][TRIMESTRE]` diz que temos um array de array, isto é, cada elemento do array de ANO é um array dos elementos de TRIMESTRE.

Note que os valores mostrados pelo programa são formatados de modo adequado. Os valores mostrados possuem dois dígitos à direita do ponto decimal, e os valores estão todos alinhados à direita. Para obter tal resultado, precisamos fazer algumas coisas. Já usamos o manipulador `setw()` que estabelece a largura do campo de saída. Contudo, para formatação de números decimais, outros manipuladores são usados.

7.3.4. Formatação de Números Decimais

ios é uma classe que determina como a formatação será realizada (mas isso é um detalhe desnecessário aqui). *fixed* e *showpoint* são flags de *ios*. Para usar essas flags, utilizamos o manipulador *setiosflags* com o nome da flag como um argumento. O nome deve ser precedido pelo nome da classe (*ios*) e o operador de resolução de escopo (::). (Adicionalmente, podemos usar o manipulador *resetiosflags*). A flag *fixed* evita que números no formato exponencial (1.23e4) sejam mostrados. A flag *showpoint* especifica que sempre haverá um ponto decimal mesmo que o número não tenha a parte fracionária, ou seja: teríamos 123.00 em vez de 123. A fim de estabelecer a precisão de dois dígitos à direita do ponto decimal, usamos o manipulador *setprecision* com o número de dígitos como argumento.

Da mesma forma que inicializamos um array unidimensional, podemos fazê-lo para arrays multidimensionais. O único pré-requisito é a necessidade de colocar muitas chaves e vírgulas. Um exemplo disso é mostrado a seguir.

```
int nomeArray[3][4] = {{11, 22, 33, 30}, {44, 55, 66, 60},
                       {77, 88, 99, 90}};
```

É importante observar que os arrays podem ser usados de diversas maneiras. Em seguida, veremos como um array pode ser passado como argumento em uma chamada de função. Para ilustrar, vamos examinar o próximo exemplo.

Praticando um Exemplo. Modifique o programa da Listagem 7.4 para explorar como passar um array como parâmetro em uma chamada à função. Para tanto, similarmente ao programa anterior, você deve criar um array bidimensional com valores do tipo `double`. Chame esse array de `despesas[ANO][TRIMESTRE]`. Adicionalmente, você deve definir `ANO=2` e `TRIMESTRE=4`. Em seguida, deve criar um laço que solicite ao usuário digitar os valores de despesas para cada um dos semestres nos dois anos. Esses valores digitados devem ser armazenados no array que você criou. Agora, crie uma função cujo protótipo é:

```
void exibeDespesas(double[ANO][TRIMESTRE], double total);
```

Essa função deve exibir na tabela os valores das despesas de cada ano, sendo listadas por trimestre, além do valor total de despesas nos dois anos, similarmente ao programa anterior. Agora, feche o livro e tente implementar sua solução. Depois, consulte a solução do exemplo apresentada na Listagem 7.5.

```

1. #include <iostream>
2. #include <iomanip>
3. const int ANO = 2; // tamanho do array
4. const int TRIMESTRE = 4;
5.
6. using namespace std;
7. // Programa para ilustrar array bidimensional
8. void exibeDespesas(double despesas[ANO][TRIMESTRE], double
   total)
9. {
10.    cout << "\n\n Trimestre\n";
11.    cout << " 1 2 3 4";
12.    for(int a = 0; a < ANO; a++)
13.    {
14.        cout << "\nDespesas do ano " << a+1;
15.        for(int t = 0; t < TRIMESTRE; t++) // laço para exibir
           dados
16.            cout << setiosflags(ios::fixed)
17.                << setiosflags(ios::showpoint) // exibe ponto decimal
18.                << setprecision(2) // define precisão
19.                << setw(10)
20.                << despesas[a][t]; // exibe dado do array
21.    }
22.    cout << "\n\nDespesa total em 2 anos = " << total;
23.    cout << endl << endl;
24. }
25.
26. int main()
27. {
28.    void exibeDespesas(double[ANO][TRIMESTRE], double total);
       // prototipo
29.    double despesas[ANO][TRIMESTRE]; // declaração de array
30.
31.    double total = 0.0;
32.    for(int a = 0; a < ANO; a++) // laço para obter valores
33.        for(int t = 0; t < TRIMESTRE; t++)
34.        {
35.            cout << "Entre com as despesas do " << t + 1 << "o. tri-
               mestre do ano " << a + 1 << ": ";
36.            cin >> despesas[a][t]; // inserção de dados no array
37.            total = total + despesas[a][t];

```

```

38.  }
39.  exibeDespesas(despesas, total);
40.  system("PAUSE");
41.  return 0;
42.  }

```

Listagem 7.5

O programa da Listagem 7.5 cria um array bidimensional que serve para armazenar valores de despesas de uma empresa e solicita que o usuário digite valores de despesas por trimestre. A diferença dessa implementação em relação à anterior é que aqui os dados do array são passados como argumento na chamada a uma função que exibe os resultados. Executando o programa, você obterá a saída mostrada na Figura 7.5.

```

F:\campus\elsevier\programasCpp\programasCap7\programa7_5.exe
Entre com as despesas do 1o. trimestre do ano 1 : 1000
Entre com as despesas do 2o. trimestre do ano 1 : 2000
Entre com as despesas do 3o. trimestre do ano 1 : 3000
Entre com as despesas do 4o. trimestre do ano 1 : 4000
Entre com as despesas do 1o. trimestre do ano 2 : 2000.50
Entre com as despesas do 2o. trimestre do ano 2 : 3000.75
Entre com as despesas do 3o. trimestre do ano 2 : 4000
Entre com as despesas do 4o. trimestre do ano 2 : 5000

Despesas do ano 1      1000.00  2000.00  3000.00  4000.00
Despesas do ano 2      2000.50  3000.75  4000.00  5000.00

Despesa total em 2 anos = 24001.25
Pressione qualquer tecla para continuar. . .

```

Figura 7.5 – Saída do programa da Listagem 7.5.

■ Note que usar um endereço para um argumento de array é similar a usar um argumento de referência, uma vez que os valores não são duplicados ou copiados para função. Isso é feito porque se pode ter arrays de tamanho muito grande, o que causa desperdício de memória e consome mais tempo de processamento.

Na declaração de função, os argumentos do array são representados pelo tipo de dado e tamanho(s) do array. No programa temos a chamada de função `exibeDespesas(despesas, total)`. O nome `despesas` representa o endereço de memória do array.

7.4. ARRAYS COMO DADO DE CLASSE

Os arrays também podem ser declarados como membros de dados de uma classe. Quando os arrays são utilizados como membros de dados privados de classes,

eles se tornam automaticamente acessíveis às funções-membros daquela classe. Em tal situação, as funções-membros públicas podem modificar os arrays sempre que necessário.

Para entender mais, nada melhor do que explorar um exemplo. Nesse exemplo, o programa deve modelar uma pilha como estrutura de dado (ou seja, devemos criar uma classe para pilha). Mas o que é uma pilha?

7.4.1. Pilha – Estrutura de Dado

A pilha consiste em um array *aPilha*. Além disso, a pilha deve ter uma variável, por exemplo, do tipo inteiro (*topo*) que indica o índice do último item colocado na pilha. A localização desse item é no topo da pilha. Assim, quando um item é colocado na pilha, o endereço em *topo* é incrementado para apontar para o novo topo da pilha. Quando um item é removido da pilha, o valor em *topo* é decrementado.

Praticando um Exemplo. Escreva um programa que implemente uma classe de pilha. Essa classe deve possuir dois membros de dados: um array *aPilha[tamanhoArray]* e uma variável *topo* que serve para indicar o último elemento colocado na pilha. Para colocar um item na pilha, você deve chamar a função-membro *push()* com o valor armazenado como argumento. Para remover um item da pilha, deve usar a função-membro *pop()*, a qual retorna o valor do item. Note que as funções *push()* e *pop()* são funções-membros da classe *Pilha* e devem ser chamadas a partir de *main()*. Agora, feche o livro e tente implementar sua solução. Depois, consulte a solução do exemplo apresentada na Listagem 7.6.

```

1. #include <iostream>
2. const int MAX = 100; // tamanho do array
3. using namespace std;
4. // Programa para ilustrar array bidimensional
5.
6. class Pilha
7. {
8.     private:
9.         int aPilha[MAX]; // array pilha
10.        int topo; // elemento do topo da pilha
11.    public:
12.        Pilha() // construtor
13.        { topo = 0; }
14.        void push(int x) // push - colocar dado na pilha
15.        {
16.            aPilha[++topo] = x;

```

```
17.     }
18.     int pop() // pop - remove take um dado da pilha
19.     {
20.         return aPilha[topo--];
21.     }
22. };
23. int main()
24. {
25.     Pilha p;
26.     cout << "Dados na pilha: " << endl << endl;
27.     p.push(100);
28.     p.push(200);
29.     p.push(300);
30.     cout << "1: " << p.pop() << endl;
31.     cout << "2: " << p.pop() << endl;
32.     cout << "3: " << p.pop() << endl;
33.
34.     p.push(400);
35.     p.push(500);
36.     cout << "4: " << p.pop() << endl;
37.     cout << "5: " << p.pop() << endl << endl;
38.     system("PAUSE");
39.     return 0;
40. }
```

Listagem 7.6

O programa da Listagem 7.6 cria um objeto *p* da classe *Pilha* que tem um array dado-membro (da classe). Um conjunto de valores inteiros é inserido na pilha fazendo uso da função-membro *push()* e depois removido com a função *pop()* exibida. Executando o programa, você obterá a saída mostrada na Figura 7.6.

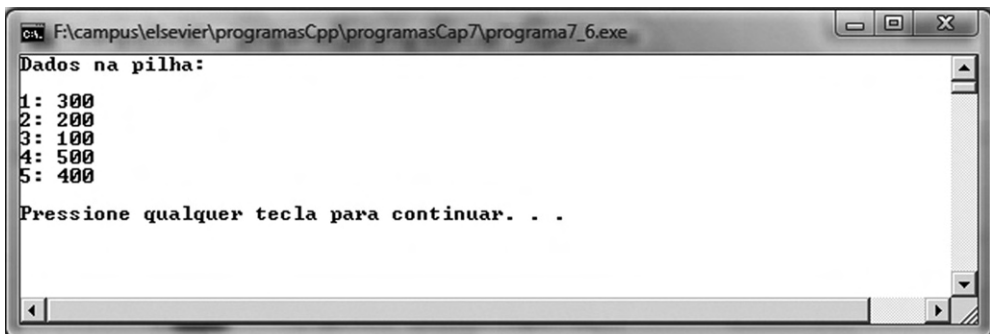


Figura 7.6 – Saída do programa da Listagem 7.6.

O programa da Listagem 7.6 usa a classe *Pilha* criando um objeto (*s1*) dessa classe. Ele coloca três itens na pilha; depois a função *pop()* é chamada e esses três itens são mostrados.

7.4.2. Operação da Pilha

Como se percebe, os itens são recuperados da pilha em ordem reversa. A política de funcionamento da pilha é: o último que entra é o primeiro que sai.

Observe, ainda, o uso dos operadores de incremento. A instrução

```
aPilha[++topo] = x;
```

na função-membro *push()* primeiro incrementa *topo* para que ele aponte para o próximo elemento disponível do array. Já a instrução

```
return aPilha[topo--];
```

primeiro retorna o valor que está no topo da pilha e depois decrementa *topo* para que ele aponte para o elemento precedente.

7.5. ARRAYS DE OBJETOS

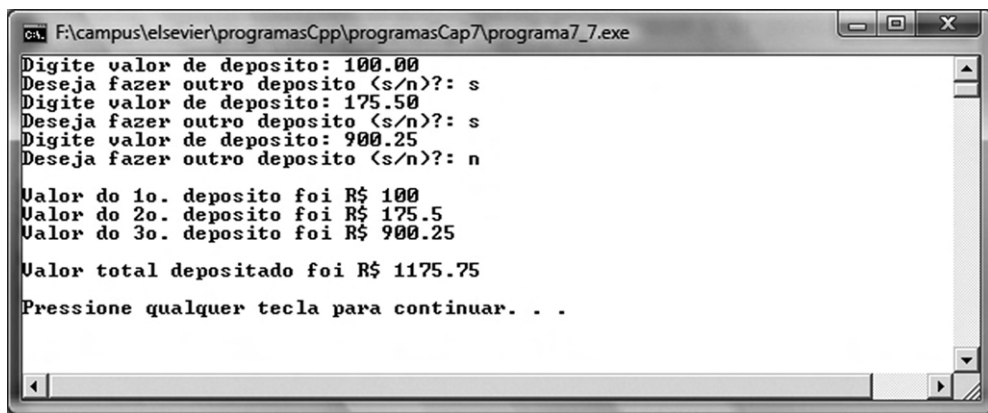
Você viu como um objeto pode conter um array (como dado-membro). Agora, você verá como podemos ter a situação inversa, isto é, ter um array de objetos.

Observe que você pode criar um array de objetos da mesma classe. Perceba que, embora possa criar um objeto independente, é muito mais interessante criar um array de objetos. Para tanto, você pode declarar um array de objetos da mesma forma que iria declarar um array de outro tipo. Para entender mais, vamos examinar um exemplo.

Praticando um Exemplo. Escreva um programa em C++ que crie um array de objetos. Para tanto, inicialmente você deve definir uma classe chamada *Conta* (conta-corrente bancária), a qual possui apenas um item de dado: *saldo*, similar ao exemplo feito no Capítulo 6. Essa classe *Conta* tem duas funções: *creditar(x)*, que faz um crédito de *x* na conta referenciada, e *getSaldo()*, que retorna o saldo da conta referenciada. Para elaborar esse programa, você deve criar um array de objetos, declarando: *Conta c[MAX]*. Essa classe deve ter um construtor *Conta()* { *saldo* = 0; } para inicializar os saldos das duas contas com valor 0. No programa, você deve informar a quantia que deseja creditar na conta *c*. O programa permite que você faça vários depósitos até o momento em que digita ‘n’, e ele encerra e lista os valores de depósito e o somatório do total de depósitos. Note ainda que você deve criar essas funções-membros e chamá-las a partir da função *main()* juntamente com a criação de objetos da classe *Conta*. Agora, feche o livro e tente implementar sua solução. Depois, consulte a solução do exemplo mostrada na Listagem 7.7.

```
1. #include <iostream>
2. const int MAX = 1000;
3. using namespace std;
4. // Programa para ilustrar criacao de array de objetos
5.
6. class Conta
7. {
8.     private:
9.         double saldo;
10.    public:
11.        Conta() { saldo = 0; } // construtor
12.        void creditar(double quantia)
13.        {
14.            saldo = saldo + quantia;
15.        } // funcao credito
16.        double getSaldo()
17.        {
18.            return saldo;
19.        } // funcao getSaldo
20. };
21. int main()
22. {
23.     Conta c[MAX]; // cria objeto Conta
24.     int num = 0; // contador de entrada de dados
25.     double total = 0.0; // valor total depositado
26.     char resposta; // resposta do usuario ('s' ou 'n')
27.     double quantia; // quantia de deposito
28.
29.     do // obtem do usuario valores de deposito
30.     {
31.         cout << "Digite valor de deposito: ";
32.         cin >> quantia;
33.         c[num++].creditar(quantia); // armazena quantia no array
34.         cout << "Deseja fazer outro deposito (s/n)?: ";
35.         cin >> resposta;
36.     } // encerrar se usuario digitar 'n'
37.     while( resposta!= 'n' );
38.
39.     for(int j = 0; j < num; j++) // exhibe valores depositados
40.     {
41.         cout << "\nValor do " << j + 1 << "o. deposito foi R$ ";
42.         cout << c[j].getSaldo();
43.         total += c[j].getSaldo();
44.     }
45.     cout << "\n\nValor total depositado foi R$ " << total <<
endl << endl;
46.     system("PAUSE");
47.     return 0;
48. }
```

O programa da Listagem 7.7 cria um array de objetos *c* da classe *Conta* que tem tamanho MAX. Esse programa permite digitar valores que você tenha interesse em depositar em uma conta e, em seguida, ele exibe os valores creditados e a soma total de depósito. Executando o programa, você obterá a saída mostrada na Figura 7.7.



```

F:\campus\elsevier\programasCpp\programasCap7\programa7_7.exe
Digite valor de deposito: 100.00
Deseja fazer outro deposito (s/n)?: s
Digite valor de deposito: 175.50
Deseja fazer outro deposito (s/n)?: s
Digite valor de deposito: 900.25
Deseja fazer outro deposito (s/n)?: n
Valor do 1o. deposito foi R$ 100
Valor do 2o. deposito foi R$ 175.5
Valor do 3o. deposito foi R$ 900.25
Valor total depositado foi R$ 1175.75
Pressione qualquer tecla para continuar. . .
  
```

Figura 7.7 – Saída do programa da Listagem 7.7.

Entretanto, se desejar evitar a entrada de dados fora do limite do array, poderá inserir o seguinte fragmento de código no início do loop do programa da Listagem 7.7:

```

if (n >= MAX)
{
    cout << "\nO array está no limite!";
    break;
}
  
```

7.6. STRINGS E ARRAYS

Agora, que você já está familiarizado com arrays, podemos examinar *strings*, que compreendem uma forma especial de usar arrays do tipo `char`. Em outras palavras, *string* é um array de caracteres (tipo `char`) que termina com o caractere ASCII nulo (`\0`).

A linguagem C++ oferece duas formas de fazer uma atribuição de uma *string*. Você pode atribuir uma *string* literal (ou simplesmente literal) para uma variável do tipo *string* no momento em que inicializa a variável. Para tanto, você utiliza o operador de atribuição (=) seguindo a sintaxe a seguir.

```
char variavelString[tamanhoString] = literal
```

Como exemplos, você pode ter:

```

char nome[20] = "Antonio Mendes";
char endereco[ ] = "Avenida Paulista, S/N";
  
```


Um outro método faz uso da função `strcpy`. Essa função assume que a string a ser copiada termina com caractere nulo. A função `strcpy` copia os caracteres de uma string de origem (fonte) para uma string de destino. Para tanto, a função string de destino possui espaço suficiente para acomodar todos os caracteres da string fonte. O exemplo a seguir ilustra a sintaxe.

```
char endereco[40];  
strcpy(endereco, "Avenida Paulista, S/N");
```

Assim como com os outros tipos de dados, as strings podem ser variáveis ou constantes. Inicialmente, vamos examinar um exemplo com variável string.

Praticando um Exemplo. Escreva um programa que defina uma variável do tipo string. Para tanto, você deve usar um array do tipo char. Por exemplo, suponha que você leia um nome e declare: `char nome[MAX]`, onde `MAX = 100`. O programa deve solicitar ao usuário digitar uma string (por exemplo, um nome), e depois mostra a string digitada. Agora, feche o livro e tente implementar sua solução. Depois, consulte a solução do exemplo apresentada na Listagem 7.8.

```
1. #include <iostream>  
2. const int MAX = 100;  
3. using namespace std;  
4. // Programa para ilustrar criacao de array de char  
5.  
6. int main()  
7. {  
8.     char nome[MAX]; // declara array do tipo char - string  
9.     cout << "\nDigite uma string: ";  
10.    cin >> nome; // ler string digitada pelo usuario  
11.    cout << "Voce digitou: " << nome; // exhibe string digitada  
12.    cout << endl << endl;  
13.    system("PAUSE");  
14.    return 0;  
15. }
```

Listagem 7.8

O programa da Listagem 7.8 cria um array de char `nome[MAX]` que tem tamanho `MAX`. Esse programa permite digitar uma string qualquer como, por exemplo, um endereço e, em seguida, exhibe a string digitada. Executando o programa, você obterá a saída mostrada na Figura 7.8.

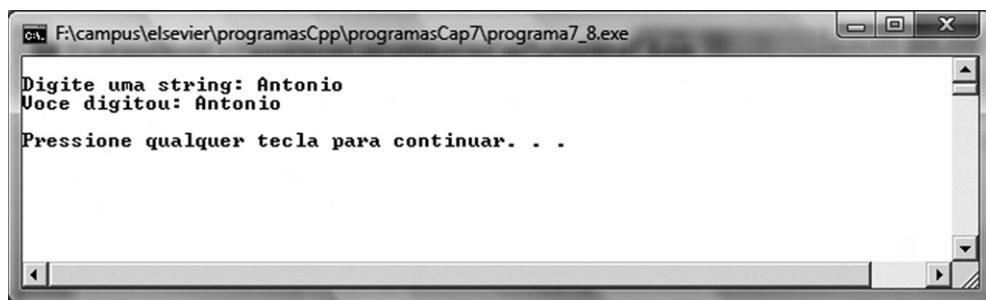


Figura 7.8 – Saída do programa da Listagem 7.8.

Observe que a definição da variável `string` é similar à definição de qualquer array do tipo `char`. O programa da Listagem 7.8 permite que você digite uma string de tamanho 100, como especificado. Agora, o que acontece se você digitar uma string maior do que o array usado para contê-la?

Em C++, contudo, não existe mecanismo embutido para evitar o programa de inserir elementos além do tamanho do array. Para lidar com esse problema, vamos examinar outro exemplo.

Praticando um Exemplo. Escreva um programa que defina uma variável do tipo `string`. Neste exemplo, você deve usar um array do tipo `char`, declarando: `char nome[MAX]`, onde `MAX = 30`. O programa deve solicitar ao usuário digitar uma string (por exemplo, um nome), e depois mostra a string digitada. Entretanto, se você digitar mais do que 30 caracteres, o operador `>>` não vai inserir os excedentes no array. Apenas 29 caracteres, no máximo, são inseridos. Agora, feche o livro e tente implementar sua solução. Depois, consulte a solução do exemplo apresentada na Listagem 7.9.

```
1. #include <iostream>
2. #include <iomanip>
3. const int MAX = 30;
4. using namespace std;
5. // Programa para ilustrar criação de array de char
6.
7. int main()
8. {
9.     char nome[MAX]; // declara array do tipo char - string
10.    cout << "\nDigite uma string: ";
11.    cin >> setw(MAX) >> nome; // ler string digitada, limitando-
    ao tamanho MAX
12.    cout << "Voce digitou: " << nome; // exibe string digitada
13.    cout << endl << endl;
```

```
14.  system("PAUSE");  
15.  return 0;  
16. }
```

Listagem 7.9

O programa da Listagem 7.8 cria um array de char *nome*[*MAX*] que tem tamanho *MAX* = 30. Esse programa permite digitar uma string qualquer desde que ela não exceda a 29 caracteres. Já o programa da Listagem 7.9 usa o manipulador *setw*(*MAX*) na linha 11 para especificar o número máximo de caracteres que o buffer de entrada pode aceitar. Assim, se você digitar mais caracteres que o máximo permitido, o operador *>>* não vai inseri-los no array. No programa anterior, apenas um máximo de 29 caracteres são inseridos. Executando o programa, você obterá a saída mostrada na Figura 7.9.

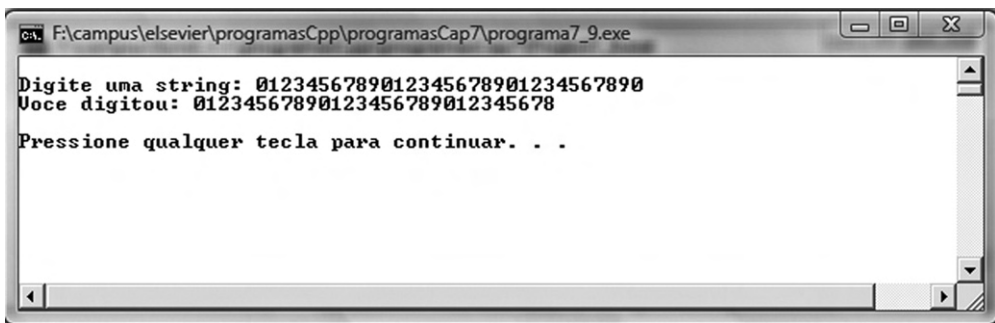


Figura 7.9 – Saída do programa da Listagem 7.9.

Após ter aprendido a utilizar variáveis strings, é o momento de ver como trabalhar com constantes strings. Você pode inicializar uma string com um valor constante, como ilustrado a seguir.

```
char nome[] = "Antonio Mendes";
```

Observe que, para inicializar uma string, você deve colocar a string entre aspas (“ ”). Para examinar melhor como funciona, vejamos o próximo exemplo.

Praticando um Exemplo. Escreva um programa que defina uma constante do tipo string. Neste exemplo, você deve usar um array do tipo char, declarando, por exemplo: *char nome[] = "Antonio Mendes"*. Uma vez inicializada a constante string, o programa deve exibir a string contida na constante declarada. Agora, feche o livro e tente implementar sua solução. Depois, consulte a solução do exemplo apresentada na Listagem 7.10.

```

1. #include <iostream>
2. using namespace std;
3. // Programa para ilustrar uso de constante string
4. int main()
5. {
6.     char nome[] = "Antonio Mendes"; // declara e inicializa
        constante string
7.     cout << "Constante string: " << nome; // exibe constante string
8.     cout << endl << endl;
9.     system("PAUSE");
10.    return 0;
11. }

```

Listagem 7.10

O programa da Listagem 7.10 declara e inicializa uma constante string, criando um array de char `nome[]` e lhe atribuindo uma string. Executando o programa, você obterá a saída mostrada na Figura 7.10.

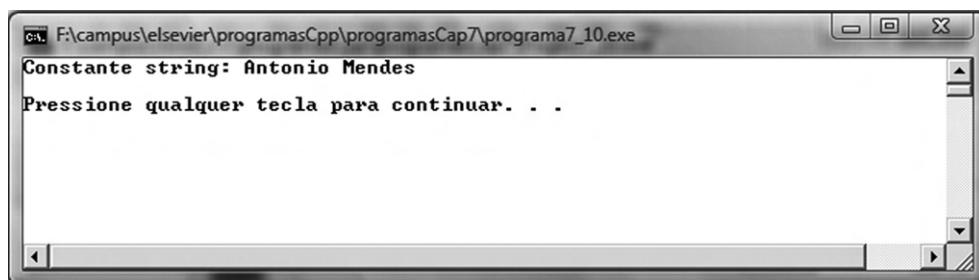


Figura 7.10 – Saída do programa da Listagem 7.10.

Vale ressaltar que você também pode inicializar uma string fazendo:

```
char nome[] = {'A', 'n', 't', 'o', 'n', 'i', 'o'};
```

É importante observar que ambas as formas têm o mesmo efeito. Todavia, a primeira é comumente usada.

Agora, se você quiser ou precisar ler, por exemplo, uma frase ou outro texto qualquer com espaços em branco entre as strings, como proceder?

■ *Note que, se você tentasse digitar uma frase (palavras com espaço em branco entre elas) no programa da Listagem 7.9, apenas a primeira palavra seria mostrada. Mas o que acontece com o resto? Em tal situação, o operador de extração >> considera um espaço como sendo um caractere de terminação. Portanto, ler strings consiste em ler uma única palavra. Assim, qualquer coisa digitada após o espaço é perdida.*

7.6.1. Função `cin::get()`

Em tal situação, para ler um texto com espaços em branco, você precisará usar outra função, `cin::get()`. Essa sintaxe informa que a função-membro `get()` da classe (stream) na qual `cin` é um objeto. Para utilizar a função `get()`, você deve usar a instrução `cin.get(sentença, MAX)`, onde `sentença[]` é uma variável do tipo `char` que pode ler uma quantidade de `MAX-1` caracteres. Para entender mais, vamos examinar o próximo exemplo.

Praticando um Exemplo. Escreva um programa que defina uma variável do tipo `string`. Neste exemplo, você deve usar um array do tipo `char`, declarando, por exemplo: `char sentenca[MAX]` de tamanho `MAX`. Você deve solicitar que o usuário digite uma frase qualquer e depois tecele Enter para encerrar. Uma vez que o usuário tecele Enter, o programa deve exibir toda a frase digitada. Note que você deve usar a função `cin.get(sentenca, MAX)` para ler e inserir o conteúdo digitado pelo usuário em `sentenca`. Agora, feche o livro e tente implementar sua solução. Depois, consulte a solução do exemplo apresentada na Listagem 7.11.

```
1. #include <iostream>
2. const int MAX =100;
3. using namespace std;
4. // Programa para ilustrar uso de constante string
5.
6. int main()
7. {
8.     char sentenca[MAX]; // declara variavel string
9.     cout << "Digite uma frase e depois tecle 'Enter':\n" << "-->> ";
10.    cin.get(sentenca, MAX); // coloca caracteres lidos em sentenca
11.    cout << "\nVoce digitou a frase: \n" << "-->> " << sentenca
12.    << endl << endl;
13.    system("PAUSE");
14.    return 0;
15. }
```

Listagem 7.11

Na linha 10 da Listagem 7.11, o primeiro argumento para `cin::get()` é o endereço do array onde a string será colocada (isto é, `sentenca`). O segundo argumento especifica o tamanho máximo do array (`MAX`). O programa da Listagem 7.10 declara uma variável `string` criando um array de `char` `sentenca[]` de tamanho `MAX`. Executando o programa, você obterá a saída mostrada na Figura 7.11.

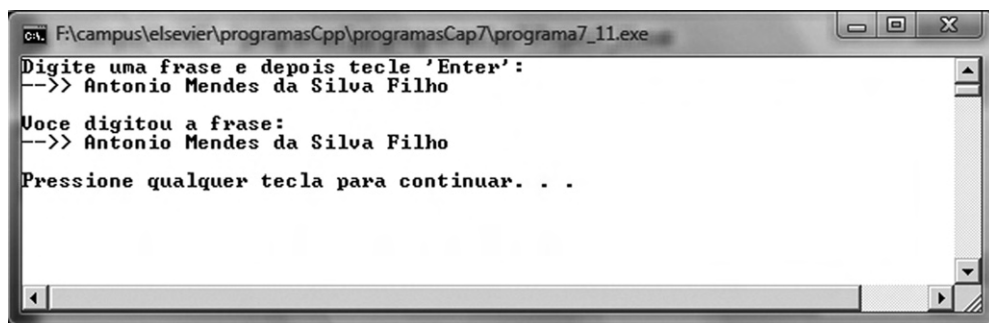


Figura 7.11 – Saída do programa da Listagem 7.11.

Embora o problema de espaços em branco tenha sido resolvido, ainda assim precisa-se tratar o caso de múltiplas linhas. Nesse caso, a função *get()* pode ter um terceiro argumento a fim de resolver essa situação. Esse argumento informa à função qual caractere ela deve encontrar para parar de ler.

7.6.2. Usando *get()* para Ler Múltiplas Linhas

Da mesma forma que a função *get()* foi utilizada na Listagem 7.11, o valor default desse terceiro argumento é o caractere `'\n'` (newline). Todavia, você pode usar outro caractere (o qual substituirá o caractere default pelo caractere especificado). Por exemplo, substitua a linha do programa anterior pela linha a seguir:

```
cin.get(sentenca, MAX, '#');
```

Utilizando `#` como terceiro argumento, você pode digitar quantas linhas desejar. A função continuará a aceitar os caracteres até que o caractere de terminação (`#`) seja digitado. Observe que é preciso teclar Enter após o caractere `#`. Entretanto, se você digitar mais caracteres do tamanho MAX, o programa também encerrará.

7.6.3. Função *strcpy*

A função *strcpy* copia caracteres de uma string de origem para uma string de destino. A função *strcpy* assume que a string de destino possui espaço suficiente para armazenar o conteúdo da string fonte. Você pode utilizar a função *strcpy*, seguindo a sintaxe do exemplo a seguir:

```
char sentenca[100];
strcpy(sentenca, "Antonio Mendes");
```

Utilizando essas instruções, o conteúdo da string “Antonio Mendes” é copiado na variável string *sentenca*. Perceba que a cópia de strings é tratada caractere por

caractere. Você também pode copiar um conteúdo de uma string para outra fazendo uso do operador de atribuição como no exemplo a seguir:

```
sentenca1[i] = "Frase qualquer";  
...  
sentenca2[i] = sentenca1[i];
```

■ *Note que o programa pode ser encerrado se você digitar o caractere de terminação (especificado no terceiro argumento da função `cin::get()`) ou se a quantidade máxima de caracteres (MAX) especificada for atingida.*

Esse fragmento de programa cria uma constante string (*sentenca1*) e uma variável string (*sentenca2*). Em seguida, o programa usa um loop for (com índice *i*) para copiar a constante string para a variável string, como mostrado. A cópia é feita caractere por caractere.

7.6.4. Função `strlen()`

Muitas operações com string exigem informação sobre a quantidade de caracteres de uma string. Em tal situação, você pode utilizar a função `strlen()`, que retorna o número de caracteres que a string possui, excluindo-se o caractere de terminação nulo (`\0`).

Praticando um Exemplo. Escreva um programa que defina e inicialize uma constante string *sentenca1* e também uma variável string *sentenca2*. Para a variável string, você deve usar um array do tipo `char`, declarando, por exemplo: `char sentenca2[MAX]` de tamanho *MAX*. Você deve escrever um laço for para controlar a cópia caractere por caractere da string *sentenca1* para *sentenca2*. Aqui, você pode utilizar a função `strlen(nomeString)` que retorna o tamanho de string *nomeString*. Uma vez que você tenha copiado o conteúdo da *sentenca1* para *sentenca2*, deve exibir o conteúdo de *sentenca2*, além dos tamanhos das strings usando a função `strlen()`. Agora, feche o livro e tente implementar sua solução. Depois, consulte a solução do exemplo apresentada na Listagem 7.12.

A fim de entender mais a operação com strings, vamos examinar o próximo exemplo.

```
1. #include <iostream>  
2. const int MAX =200;  
3. using namespace std;  
4. // Programa para ilustrar copia de uma string para outra  
5.  
6. int main()  
7. {
```

```

8.  char sentenca1[] = "\"Do not worry about your difficulties
    in Mathematics.\n"
9.      " I can assure you mine are still greater\".\n"
10.     " Albert Einstein\n\n";
11.  char sentenca2[MAX]; // declara variavel string
12.
13.  for(int i = 0; i < strlen(sentenca1); i++) // usa strlen
    para obter
14.  sentenca2[i] = sentenca1[i]; // numero de caracteres em
    sentenca1
15.  sentenca2[116] = '\0'; // insere o caractere nulo no final
    da string
16.  cout << sentenca2;
17.  cout << "\nTamanho de sentenca 1: " <<strlen(sentenca1);
18.  cout << "\nTamanho de sentenca 2: " <<strlen(sentenca2) <<
    endl << endl;
19.  system("PAUSE");
20.  return 0;
21. }

```

Listagem 7.12

O programa da Listagem 7.12 declara e inicializa uma constante string `sentenca1` e copia seu conteúdo para a variável string `sentenca2`. Executando o programa, você obterá a saída mostrada na Figura 7.12.

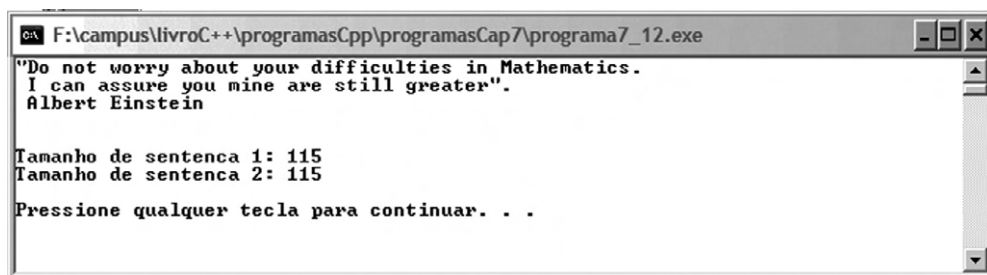


Figura 7.12 – Saída do programa da Listagem 7.12.

Note, no programa da Listagem 7.12, que foi feito uso da função `strlen()` para controlar o laço que realiza a cópia da constante string. Uma boa prática é terminar a string com o caracter *null* (`'\0'`), como feito na linha 15.

Entretanto, você pode minimizar todo esse trabalho para copiar uma string e simplificar seu programa fazendo uso da função `strcpy()`, como discutido. Veja o próximo exemplo, que ilustra o uso do `strcpy()`.

Praticando um Exemplo. Escreva um programa que defina e inicialize uma constante string *sentenca1* e também uma variável string *sentenca2*. Para a variável string, você deve usar um array do tipo *char*, declarando, por exemplo: *char sentenca2[MAX]* de tamanho *MAX*. Você deve utilizar a função *strcpy(stringDestino, stringOrigem)* para copiar os caracteres da string *sentenca1* para *sentenca2*. Uma vez que tenha copiado o conteúdo da *sentenca1* para *sentenca2*, você deve exibir o conteúdo de *sentenca2*, além dos tamanhos das strings usando a função *strlen()*. Agora, feche o livro e tente implementar sua solução. Depois, consulte a solução do exemplo apresentada na Listagem 7.13.

```
1. #include <iostream>
2. const int MAX =200;
3. using namespace std;
4.
5. // Programa para ilustrar copia de uma string para outra
6.
7. int main()
8. {
9.     char sentenca1[] = "\"Do not worry about your difficulties
        in Mathematics.\n"
10.         " I can assure you mine are still greater\".\n"
11.         " Albert Einstein\n\n";
12.
13.     char sentenca2[MAX]; // declara variavel string
14.     strcpy(sentenca2, sentenca1);
15.     cout << sentenca2;
16.     cout << "\nTamanho de sentenca 1: " <<strlen(sentenca1);
17.     cout << "\nTamanho de sentenca 2: " <<strlen(sentenca2) <<
        endl << endl;
18.     system("PAUSE");
19.     Return 0;
20. }
```

Listagem 7.13

O programa da Listagem 7.13 declara e inicializa uma constante string *sentenca1* e copia seu conteúdo para a variável string *sentenca2* usando a função *strcpy()*. Executando o programa, você obterá a saída mostrada na Figura 7.13.

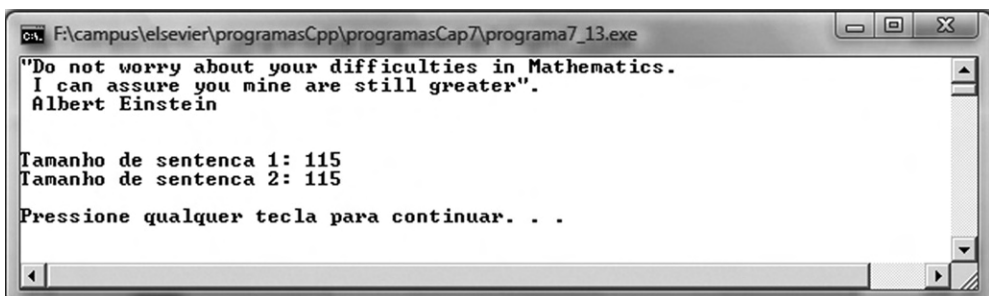


Figura 7.13 – Saída do programa da Listagem 7.13.

7.6.5. Array de strings

Já vimos que podemos ter um array de arrays. Então, podemos ter um array de strings. Desde que uma string é um array, *array*[][] (um array de strings) é um array bidimensional. Para entender mais, vamos examinar um exemplo.

Praticando um Exemplo. Escreva um programa que defina e inicialize uma variável string *dias*. Para a variável string, você deve usar um array do tipo char, declarando, por exemplo: *char dias[DIAS][MAX]* onde *DIAS* = 7 e tamanho *MAX* = 100. Note como cada string é acessada, ou seja, *dias[j]*, onde *j* varia de 0 a *DIAS*-1. Apenas o array mais externo é acessado. Você deve inicializar o array *dias* e, em seguida, o conteúdo do array *dias* deve ser exibido. Agora, feche o livro e tente implementar sua solução. Depois, consulte a solução do exemplo apresentada na Listagem 7.14.

```

1. #include <iostream>
2. const int DIAS = 7; // numero de strings no array
3. const int MAX = 100; // tamanho maximo de cada string
4. using namespace std;
5.
6. // Programa para ilustrar um array de strings
7. int main()
8. {
9.     char dias[DIAS][MAX] = { "Segunda", "Terca", "Quarta",
10.                               "Quinta", "Sexta",
11.                               "Sabado", "Domingo" };
12.     for(int i = 0; i < DIAS; i++) // display every string
13.         cout << dias[i] << endl;
14.     cout << endl;
15.     system("PAUSE");
16.     return 0;
17. }
```

Listagem 7.14

O programa da Listagem 7.14 declara e inicializa uma variável string *dias* com os nomes dos dias da semana e depois exibe seu conteúdo. Executando o programa, você obterá a saída mostrada na Figura 7.14.

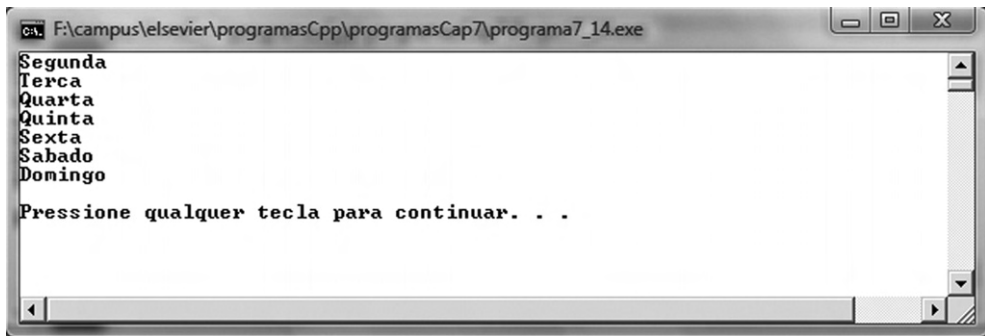


Figura 7.14 – Saída do programa da Listagem 7.14.

7.6.6. Strings como Membros de Classes

Strings, geralmente, aparecem como membros de classes. Para entender isso, nada melhor do que explorar um exemplo.

Praticando um Exemplo. Escreva um programa que defina dois objetos da classe *estoqueCelular* que possuem os atributos nome do fabricante (*nomeFab*) e código do modelo (*codigoModelo*) *custo*. Também é feita a atribuição de valores a esses objetos, fazendo uso da função-membro *setDados()*. Em seguida, ele mostra os valores com a função-membro *mostraDados()*. Na função *setDados()*, utilizamos a função *strcpy()* para copiar a string do argumento *f* para o dado-membro da classe (*nomeFab*). Note como cada string é acessada, ou seja, *dias[j]*, onde *j* varia de 0 a DIAS-1. Apenas o array mais externo é acessado. Você deve inicializar o array *dias* e, em seguida, o conteúdo do array *dias* deve ser exibido. Agora, feche o livro e tente implementar sua solução. Depois, consulte a solução do exemplo apresentada na Listagem 7.15.

```
1. #include <iostream>
2. using namespace std;
3. // Programa para ilustrar uso de strings como membro de classe
4.
5. class estoqueCelular // especificacao de classe
6. {
7.     private:
8.         char nomeFab[100];
9.         int codigoModelo;
10.        double custo;
11.    public:
12.        void setDados(char f[], int m, double c) // definicao de dados
13.        {
```

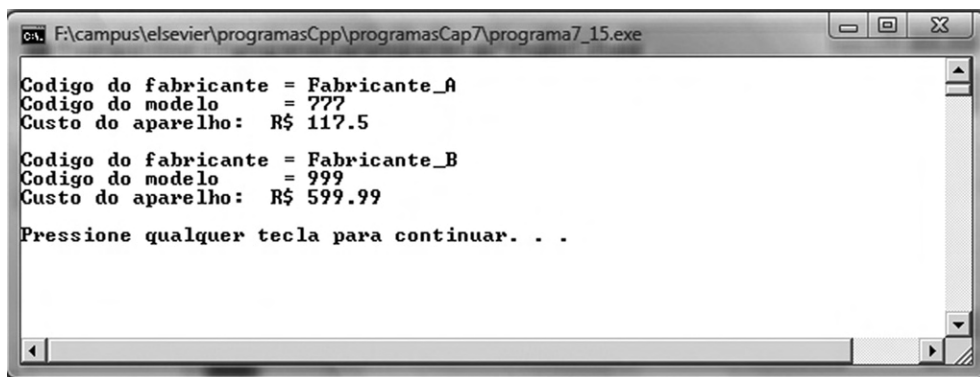
```

14.     strcpy(nomeFab, f);
15.     codigoModelo = m;
16.     custo = c;
17. }
18. void mostraDados() // mostra dados
19. {
20.     cout << "\nCodigo do fabricante = " << nomeFab;
21.     cout << "\nCodigo do modelo = " << codigoModelo;
22.     cout << "\nCusto do aparelho: R$ " << custo;
23.     cout << endl;
24. }
25. };
26. int main()
27. {
28.     estoqueCelular obj1,obj2; // declaracao de 2 objetos
29.     obj1.setDados("Fabricante_A", 777, 117.5 ); // chamada a
        funcao membro para definir valores
30.     obj2.setDados("Fabricante_B",999, 599.99);
31.
32.     obj1.mostraDados(); // chamada a funcao membro para mostrar
        valores
33.     obj2.mostraDados();
34.     cout << endl;
35.     system("PAUSE");
36.     return 0;
37. }

```

Listagem 7.15

O programa da Listagem 7.15 declara e define os valores de dois objetos da classe *estoqueCelular*. Executando o programa, você obterá a saída mostrada na Figura 7.15.



```

F:\campus\elsevier\programasCpp\programasCap7\programa7_15.exe
Codigo do fabricante = Fabricante_A
Codigo do modelo = 777
Custo do aparelho: R$ 117.5

Codigo do fabricante = Fabricante_B
Codigo do modelo = 999
Custo do aparelho: R$ 599.99

Pressione qualquer tecla para continuar. . .

```

Figura 7.15 – Saída do programa da Listagem 7.15.

RESUMO

Neste capítulo, você aprendeu sobre arrays e strings. Você teve a oportunidade de aprender que os arrays compreendem uma coleção contínua de dados do mesmo tipo. Essa coleção agrupa os dados em posições contínuas de memória e esses dados são do mesmo tipo, o que os diferencia das estruturas que agrupam dados de tipos distintos. Além disso, você também aprendeu a inicializar e acessar os dados de um array. Você ainda aprendeu a criar um array de objetos e a utilizar string que consiste em um array do tipo char. Diversos exemplos foram usados para a apresentação do conteúdo. No próximo capítulo, você estudará e explorará o uso de sobrecarga de operadores e como eles podem ser usados na programação orientada a objetos.

QUESTÕES

1. Qual a diferença entre arrays e estruturas? Use exemplos para ilustrar sua resposta.
2. O que são strings? Como definir uma string usando array?
3. Como você pode determinar o tamanho de uma string? Use um exemplo para ilustrar sua resposta.
4. Strings podem ser dados-membros de uma classe? Em que situações isso pode ser empregado? Use um exemplo para ilustrar sua resposta.

EXERCÍCIOS

1. Faça uma pesquisa visando responder à seguinte questão: é possível usar array para implementar uma pilha? Apresente um exemplo para ilustrar sua resposta
2. Escreva um programa que cria um array de double de tamanho N. Chame esse array de nota[N]. Adicione também ao seu código um laço for para que você possa ler até N notas. Você deve usar um sentinela (−1) para controlar o término da entrada de notas, quando o programa deve exibir a média das notas digitadas.
3. Modifique o programa anterior de modo que você possa ler as notas do primeiro e do segundo exercícios, que têm pesos 2 e 3, respectivamente. Em seguida, seu programa deve calcular a média ponderada de cada aluno e da turma.
4. Escreva um programa que use arrays para implementar a soma de duas matrizes de tamanho 3×3 de inteiros.
5. Escreva um programa que crie um array de inteiros de tamanho N. Seu programa pode ler até N valores. Você deve usar um sentinela (−1) para controle do término da entrada de valores, quando o programa deve colocar os valores em ordem ascendente e exibir o conteúdo do array ordenado.