

Componentes da Linguagem C++

It is a miracle that curiosity survives formal education.

Albert Einstein

OBJETIVOS

- Aprender a elaborar programas simples usando C++.
- Identificar como declarar variáveis, ler dados de entrada e exibir resultados de processamento.
- Fazer conversão de tipos de dados e utilizar operadores numéricos.
- Utilizar funções da biblioteca e diretivas do pré-processador.

A linguagem C++ possui um conjunto de elementos que permite aos programadores implementar determinadas funcionalidades. Para tanto, é preciso ficar sabendo dos recursos oferecidos pela programação orientada a objetos (POO) e, adicionalmente, conhecer os componentes da linguagem de programação OO. Aqui nosso foco recai sobre a linguagem C++.

Dentro desse contexto, há questões para as quais você busca respostas para: Como declarar variáveis? Quais os tipos suportados pela linguagem C++? Como ler dados de entrada? Como exibir os resultados computados? Como fazer uso da biblioteca da linguagem?

Responder a essas e outras questões compreende os propósitos deste capítulo para que você saiba como identificar situações nas quais pode empregar C++ adequadamente.

2.1. INTRODUÇÃO

Em qualquer linguagem de programação, existe um conjunto de princípios básicos (fundamentando a linguagem) que, juntamente com os elementos que a

compõem, a caracterizam, tornando-a adequada na solução de problemas. Ter esse conhecimento é essencial para escolher de modo eficaz a ferramenta ou linguagem de programação apropriada para o problema que se tem em mãos. Isso permite a elaboração de programas naquela linguagem.

O Capítulo 1 apresentou características e princípios da programação orientada a objetos, os quais são detalhados no livro. Este capítulo, especificamente, apresenta os componentes da linguagem de programação C++ fazendo uso de exemplos ilustrativos. Todos os exemplos neste e nos demais capítulos têm seu código completo apresentado tanto no livro quanto disponibilizado no site www.elsevier.com.br.

2.2. CONSTRUÇÃO DE UM PROGRAMA

Para conhecer uma ferramenta, não há nada melhor do que utilizá-la. E é justamente isso o que você vai fazer agora. Para tanto, deve ter instalado em seu computador algum compilador C++, dentre aqueles informados no site da editora. Neste livro, usarei o Dev-C++. Veja também, no site, orientações de onde fazer o download do ambiente Dev-C++ de desenvolvimento de programas em C e C++.

2.2.1. Criando um Projeto no Ambiente Dev-C++

Para começar, você vai elaborar um programa simples usando a linguagem C++. Para tanto, deve inicializar o Dev-C++ instalado em seu computador. Entretanto, antes de escrever seu primeiro programa, crie um projeto no qual poderá incluí-lo. Depois, clique em **File** no menu principal e, em seguida, selecione **New** e, finalmente, escolha **Project**, conforme ilustrado na Figura 2.1.

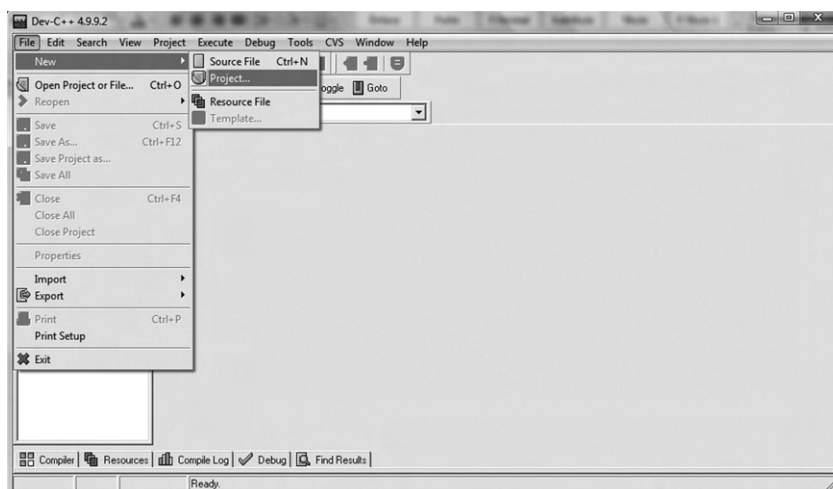


Figura 2.1 – Criando um projeto no Dev-C++.

Dessa forma, uma caixa de diálogo, conforme mostrado na Figura 2.2, vai aparecer solicitando que você selecione o tipo de projeto e informe o nome dele. Você, então, seleciona o ícone **Console Application** e digita o nome do projeto. Em seguida, seleciona a linguagem a ser utilizada (**C++ Project**) e, por fim, clica no botão **Ok**.

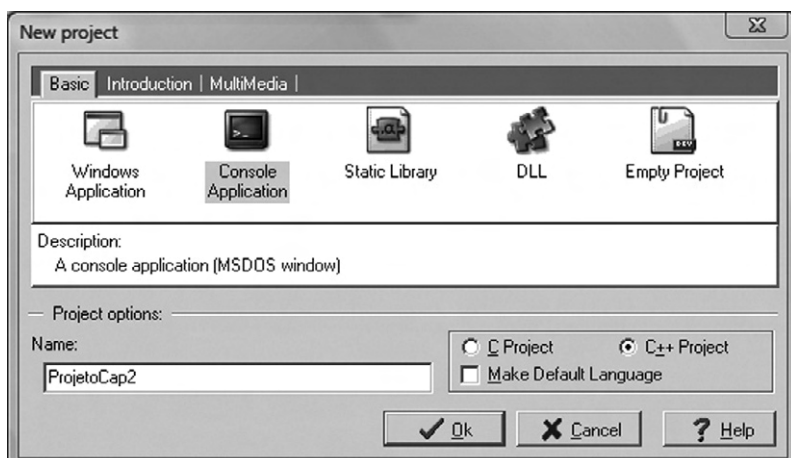


Figura 2.2 – Definindo o tipo de projeto no Dev-C++.

2.2.2. Criando um Arquivo no Dev-C++

Após fazer isso, o ambiente Dev-C++ exibe o projeto recém-criado e, ao lado do nome do projeto, haverá um sinal de **+**, como indicado na Figura 2.3. Clicando sobre esse sinal de **+**, o Dev-C++ exibe o seu conteúdo, que consiste em um arquivo **main.cpp**, o qual é exibido do lado direito do ambiente. Trata-se de um arquivo que é automaticamente criado pelo ambiente Dev-C++ quando você cria um projeto.

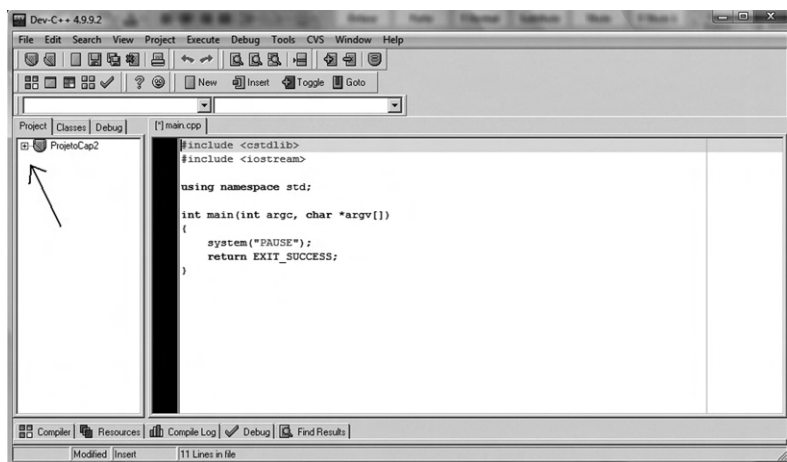


Figura 2.3 – Visualizando o conteúdo de projeto no Dev-C++.

2.2.3. Removendo ou Renomeando um Arquivo

Nesse momento, você pode remover ou renomear esse arquivo, escolhendo o nome que quiser. Aqui, recomendo que remova esse arquivo clicando com o botão direito do mouse sobre o arquivo **main.cpp** e selecionando a opção Remove file, conforme ilustrado na Figura 2.4.

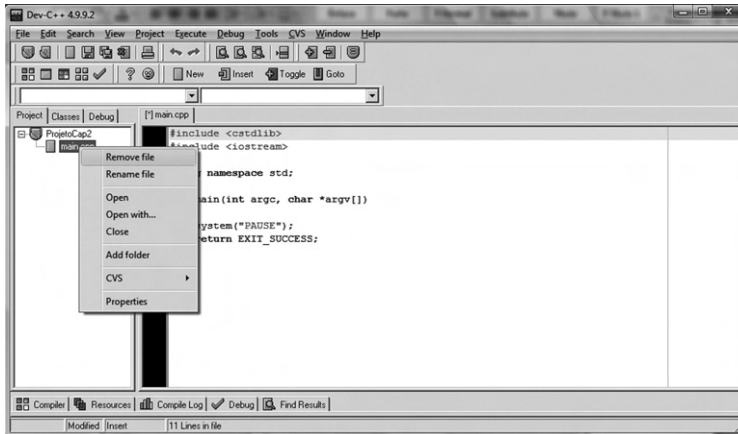


Figura 2.4 – Removendo um arquivo de um projeto no Dev-C++.

Depois que fizer isso, poderá criar um programa e atribuir o nome que quiser. Para tanto, basta clicar em **projetoCap2** com o botão direito do mouse e selecionar **New File**, como mostrado na Figura 2.5.

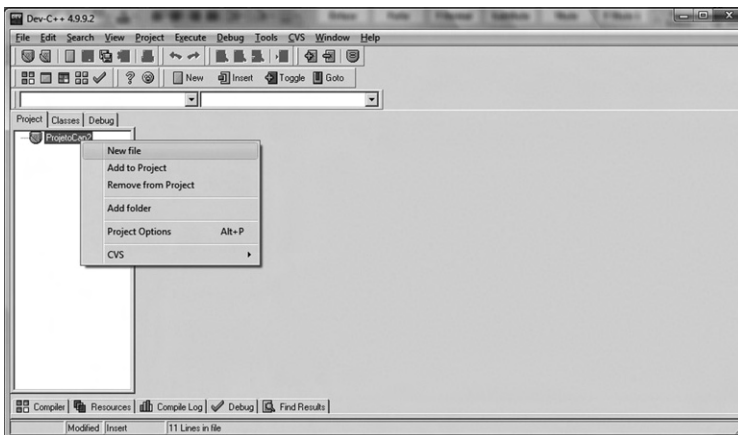


Figura 2.5 – Criando um arquivo novo em um projeto do Dev-C++.

Uma vez que tenha criado um novo arquivo, você pode digitar o seu primeiro programa, cuja listagem do código é apresentada na Listagem 2.1.

```
#include <iostream>
// Primeiro programa em C++
int main()
{
    std::cout << "Ola mundo ! \n\n"; // Exibe na tela OLA MUNDO !
    system("PAUSE");
    return 0;
}
```

Listagem 2.1

Em seguida, usando **CTRL F12**, salve o arquivo com o nome que desejar. Opcionalmente, você pode clicar em **File** no menu principal do Dev-C++ e depois clicar na opção **Save As** para salvar o arquivo que digitou.

2.2.4. Compilando e Executando um Programa no Dev-C++

O código-fonte desse exemplo, assim como todos os demais programas apresentados neste livro, está disponível no site da editora. Uma vez que tenha digitado todo o programa da Listagem 2.1, você estará pronto para compilar o código-fonte e executar o programa. Para tanto, pode clicar no ícone indicado na Figura 2.6 ou clicar na opção **Execute** do menu principal do Dev-C++ e, em seguida, selecionar **Compile & Run** ou, simplesmente, teclar **F9**. Qualquer uma dessas ações faz o Dev-C++ compilar e executar o programa que está sendo trabalhado.

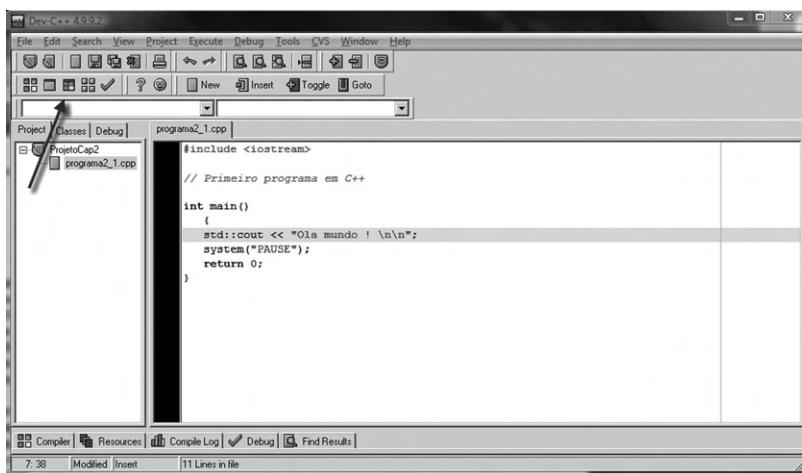


Figura 2.6 – Compilando e executando um programa no Dev-C++.

Como resultado, uma janela é aberta, exibindo o resultado do que o programa executa, como ilustrado na Figura 2.7.

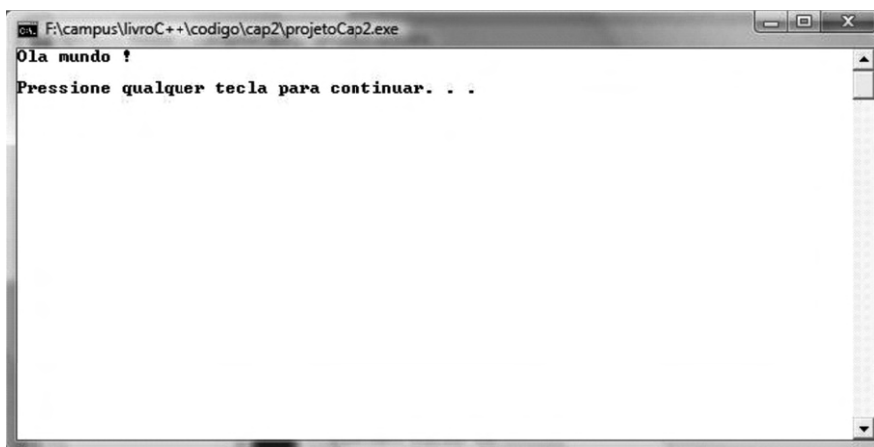


Figura 2.7 – Visualizando o resultado de um programa no Dev-C++.

2.2.5. Entendendo o Programa em C++

Agora que você já está familiarizado com o ambiente Dev-C++, é hora de entender o programa da Listagem 2.1.

Em C++, as funções compõem uma das partes fundamentais da linguagem. O programa da Listagem 2.1 consiste quase totalmente em uma única função chamada `main()`. A única parte desse programa que não é parte da função é a primeira linha, que inicia com `#include`.

Você já aprendeu no Capítulo 1 que uma função é parte de uma classe. Nesse caso, ela é chamada de função-membro. No entanto, as funções podem existir independentes das classes. Como ainda não é o momento para apresentar programas com classes, você verá primeiro as funções como entidades independentes, como `main()` no exemplo da Listagem 2.1.

Os parênteses seguindo a palavra `main` servem para caracterizar uma função. Sem os parênteses, o compilador iria interpretar `main` como uma variável ou algum outro elemento do programa. Além disso, você verá depois que nem sempre os parênteses estão vazios. Eles são usados para conter os argumentos da função. A palavra `int` que precede a função `main` indica que essa função retorna um valor inteiro.

O corpo da função é delimitado com o uso de chaves. Essas chaves têm a mesma função que `BEGIN` e `END` em outras linguagens, ou seja, elas delimitam um bloco de instruções do programa.

No exemplo da Listagem 2.1, há três instruções. A instrução `std::cout` faz com que o conteúdo entre aspas ("Olá Mundo! \n\n") seja exibido na tela. O

caractere de controle `\n` significa new line ou quebra de linha (isto é, ele vai para a linha seguinte). Como existe `\n\n`, isso significa que ele avança duas linhas. A linha seguinte, `system("PAUSE")`, faz com que o programa pare, causando uma pausa no sistema. Com isso, o programa para a execução e aparece na tela a mensagem: Pressione qualquer tecla para continuar.

Já a instrução seguinte, `return 0`, faz o controle retornar para o sistema operacional. Embora essa função possua apenas três instruções, o corpo de uma função pode possuir muitas instruções.

2.2.6. A Função `main()`

Observe também que, quando você executa um programa C++, a primeira instrução executada é o início da função `main()`. O programa pode consistir em muitas funções, classes e outros elementos. Todavia, no começo, o controle sempre vai para `main()`. Se essa função não existe, um erro durante o processo de compilação será sinalizado.

Mais adiante no livro, você verá que em um programa C++ a função `main()` pode chamar funções membros de outros objetos para executar tarefas do programa. Também a função `main()` pode possuir chamadas para outras funções independentes. Essas situações são ilustradas na Figura 2.8.

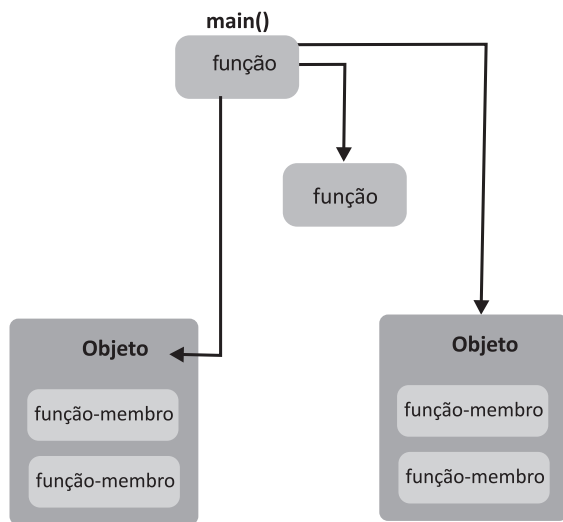


Figura 2.8 – Interação entre a função `main()` e outras funções.

Você deve perceber que a instrução é um item fundamental em qualquer linguagem de programação. Cada instrução diz ao computador qual tarefa ele deve

executar. Em C++, o *ponto-e-vírgula* (;) é utilizado para indicar o final de uma instrução. Se você, por acaso, esquecer dele, o compilador indicará erro.

Adicionalmente, o compilador ignora espaços em brancos quase completamente. Entretanto, há algumas exceções. A primeira linha do programa, iniciando com `#include`, é uma diretiva para o pré-processador, a qual deve ser escrita em uma linha (para mais detalhe, consulte a Seção 2.4). Também constantes do tipo *string*, tais como “Olá mundo!”, não podem ser *quebradas* em linhas separadas. Caso você necessite de um dado do tipo *string*, como, por exemplo, uma frase longa, insira uma contrabarra (\n) na *quebra da linha* (ou, então, divida a *string* em duas *strings* separadas, ficando cada uma circundada com aspas).

2.2.7. Comentários

Os comentários são outra parte importante em qualquer programa. Eles ajudam uma pessoa a escrever um programa e outra a entender o código-(programa) fonte (escrito pela anterior), compreendendo o que está sendo feito. O compilador, contudo, ignora os comentários e, assim, não os adiciona ao arquivo gerado no final da compilação. A seguir, o primeiro programa visto é repetido com a incorporação de comentários.

Os comentários iniciam com o símbolo de *duas barras* (//) e terminam com o fim da linha. Um comentário pode iniciar no início da linha como, por exemplo, // Primeiro programa em C++ na Listagem 2.1, ou logo após uma instrução, como em:

```
std::cout << "Ola mundo! \n\n"; // Exibe na tela OLA MUNDO!
```

Empregar comentários constitui uma boa prática de programação, uma vez que, ao fazer uso deles, o programador explica o que um programa faz. Outra forma de inserir um comentário em um programa C++ é mostrada a seguir:

```
[TD]/* este  
comentário  
é muito longo  
*/
```

Esse estilo de comentário possui vantagens em situações especiais quando, por exemplo, você necessita escrever várias linhas de comentários.

2.3. USO DE COUT

Como visto na Listagem 2.1, a instrução `std::cout << "Ola mundo! \n\n";` faz a frase entre aspas ser exibida na tela. Como isso ocorre?

2.3.1. Entendendo o Uso do Objeto cout

O entendimento completo dessa instrução depende do conhecimento sobre objetos, namespaces e outros tópicos que ainda serão discutidos. Todavia, o identificador cout é na realidade um objeto do fluxo de saída padrão. Em outras palavras, cout é predefinido em C++ para corresponder ao fluxo de saída. O fluxo é uma abstração que se refere a um fluxo de dados. O fluxo de saída padrão, normalmente, flui para a tela, embora possa ser redirecionado para outro dispositivo de saída.

2.3.2. Operador <<

Por outro lado, o operador << é denominado operador de inserção. Dessa forma, a instrução

```
[TD] std::cout << "Ola mundo! \n\n";
```

usa o operador de inserção << para direcionar a string para o dispositivo de saída, isto é, a tela. Se você conhece a linguagem C, deve estar familiarizado e reconhecerá que << é o operador de deslocamento de bits para a esquerda (na linguagem C), e pode querer saber por que ele também é usado para direcionar uma string para a saída (tela do computador).

É importante ressaltar que, em C++, os operadores podem ser *sobrecarregados*, ou seja, eles podem executar diferentes tarefas dependendo do contexto. Todavia, isso será discutido mais adiante no livro, especificamente no Capítulo 8.

A frase que está entre aspas ("Ola mundo! \n\n";), na quinta linha da Listagem 2.1, é uma constante do tipo string. Isto é, uma constante, ao contrário de uma variável, não pode receber novos valores à medida que o programa é executado. O valor de uma constante do tipo string é sempre determinado quando o programa está sendo escrito e, portanto, tem o mesmo valor durante toda a execução do programa.

2.4. DIRETIVAS DO PRÉ-PROCESSADOR

2.4.1. Diretivas do Pré-processador

A primeira linha do programa da Listagem 2.1 (`#include <iostream>`) poderia parecer uma instrução do programa mas não é. Ela não é parte do corpo da função nem termina com ponto-e-vírgula, assim como as instruções. As diretivas do pré-processador são instruções para o compilador, as quais são executadas antes que o programa seja compilado. A linha `#include <iostream>` é denominada diretiva do pré-processador.

Você deve lembrar que as instruções de programa operam sobre dados em um computador. Por outro lado, uma diretiva do pré-processador é uma instrução para o compilador. Nesse caso, parte do compilador, chamada de pré-processador, trata primeiro essas diretivas antes de iniciar o real processo de compilação.

A diretiva do pré-processador `#include` diz ao compilador para inserir um outro arquivo no código-fonte. A diretiva `#include` é substituída pelos conteúdos do arquivo indicado. Usar uma diretiva `#include` para inserir outro arquivo no código-fonte é similar a colar um bloco de texto em um documento com o processador de texto.

No programa da Listagem 2.1, a diretiva do pré-processador `#include <iostream>` diz ao compilador para adicionar o arquivo-fonte `iostream` ao arquivo-fonte do programa antes de compilá-lo. Por quê?

`iostream` é um exemplo de *arquivo cabeçalho* ou *arquivo include*. Ele contém declarações que são necessárias ao objeto `cout` e ao operador `<<`. Sem essas declarações, o compilador não reconhecerá `cout` e `<<` e pensará que ambos estão sendo usados incorretamente.

2.4.2. Diretiva `#define`

Essa diretiva é comum em programas na linguagem C. Embora tal construção não seja muito comum em C++, constantes também podem ser especificadas fazendo-se uso de diretiva do pré-processador `#define`. Um exemplo seria `#define PI 3.14159`. Para entender melhor, você deve praticar.

Praticando um Exemplo. Escreva um programa em C++ que calcule o volume de uma esfera. Suponha que o raio da esfera seja 5 cm e que o valor da constante `PI` seja 3,14159. Você sabe que a fórmula que determina o volume de uma esfera é $V = (4/3) \cdot PI \cdot R^3$. Para elaborar esse programa, você deve seguir os passos realizados quando o primeiro programa, mostrado na Listagem 2.1, foi feito. A solução deste exercício é mostrada na Listagem 2.2.

■ *Note que, na Listagem 2.2, fizemos o uso do tipo `double`, o qual é apresentado na próxima seção. `Double` e `float` são tipos utilizados quando se trabalha com números reais.*

Praticando um Exemplo. Modifique o programa da Listagem 2.2 de modo que ele calcule a área de um círculo. Suponha que o raio do círculo seja 4 cm. Você já sabe que o valor da constante `PI` é 3,14159. A fórmula que determina a área de um círculo é $A = PI \cdot R^2$. Na elaboração desse programa, você deve seguir os passos realizados. Faça a modificação solicitada e teste seu novo programa.

```
1. #include <iostream>
2. #define PI 3.141593
3. // Programa para determinar o volume de uma esfera de raio 5
   cm.
4. int main()
5. {
6.     double volume, raio;
7.     raio = 5.0; //definição do valor do raio
8.     volume = 4.0/3.0*PI*raio*raio*raio; // fórmula do volume da
   esfera
9.     // Exibe na tela o valor calculado do volume da esfera
10.    std::cout << "O volume de uma esfera de raio 5 cm = " <<
   volume << "\n\n";
11.    system("PAUSE");
12.    return 0;
13. }
```

Listagem 2.2

Agora, vamos entender o código da Listagem 2.2. A linha 1 é uma diretiva do pré-processador que instrui o compilador a incluir o arquivo `iostream` quando compilar o código da Listagem 2.2. Esse arquivo contém funcionalidades de entrada e saída que podem ser utilizadas no programa.

A linha 2, `#define PI 3.141593`, é outra diretiva do processador que define o valor da constante `PI` como sendo 3,141593. Já a linha 3 consiste em um comentário. Na linha 4, você declara a função principal `main()`, e nas linhas 5 e 13 delimita o início e o final do escopo dessa função. Como você necessita trabalhar com os valores de raio e volume, essas variáveis são declaradas na linha 6 como sendo do tipo `double` e na linha 7 você já inicializa o valor do raio como sendo 5,0.

A fórmula para calcular o volume de uma esfera é descrita na linha 8, seguida de um breve comentário. Na linha 10, você faz uso do objeto `cout` para exibir o valor computado do volume da esfera.

2.5. USO DE VARIÁVEIS

Variável é um componente essencial em qualquer linguagem de programação. Declarar uma variável implica atribuir um nome qualquer a esse componente, o qual pode receber uma variedade de valores. Quando uma variável recebe um valor, esse valor é armazenado no espaço de memória alocado a essa variável. A maioria das linguagens provê suporte a tipos como inteiros e reais (ou números de ponto flutuante) e strings.

2.5.1. Variáveis do Tipo Inteiro

O tipo mais comumente usado é `int`. Esse tipo exige 2 bytes de memória e permite armazenar números no intervalo de $-2.147.483.648$ a $2.147.483.647$. A seguir, você vai explorar um programa que define e faz uso de variáveis do tipo `int`. A palavra-chave `int` declara o tipo de variável. Observe que é necessário definir a variável antes de usá-la.

Praticando um Exemplo. Agora você vai explorar um programa em C++ que calcula a soma de duas variáveis do tipo inteiro. Para fins de exemplo você pode considerar as variáveis `x1` e `x2` e assumir que os valores são 10 e 20, respectivamente. Para elaborar esse programa, você deve declarar essas duas variáveis do tipo inteiro, inicializá-las, somá-las e, em seguida, exibir o resultado da soma. Seguir esses passos resulta em um programa, como mostrado na Listagem 2.3.

```
1. #include <iostream>
2. // Programa que solicita a entrada de dois valores inteiros
   para somá-los
3. int main()
4. {
5.     int x1; // define variavel x1
6.     int x2; // define variavel x2
7.     x1 = 10; // atribui 10 a variavel x1
8.     x2 = 20; // atribui 20 a variavel x2
9.     // Exibe na tela o valor calculado da soma das variaveis x1
       e x2
10.    std::cout << "x1 + x2 = " << x1 + x2 << "\n\n";
11.    system("PAUSE");
12.    return 0;
13. }
```

Listagem 2.3

O código da Listagem 2.3 implementa o exemplo anterior. A linha 1 é uma diretiva do pré-processador que instrui o compilador a incluir o arquivo `iostream` quando compilar o código da Listagem 2.3. A linha 2 contém um comentário, enquanto na linha 3 você declara a função principal `main()` delimitada nas linhas 4 e 13. As variáveis do tipo `int` `x1` e `x2` são declaradas nas linhas 5 e 6 e inicializadas nas linhas 7 e 8, respectivamente. A linha 10 contém o objeto `cout`, que é usado para exibir o resultado da soma `x1 + x2`.

2.5.2. Nomes de Variáveis

Na hora de escolher o nome das variáveis, você pode utilizar nomes com letras maiúsculas ou minúsculas. Pode, também, usar o caractere *underscore* (`_`) e números. Note que o compilador diferencia letras maiúsculas das minúsculas e que o primeiro caractere tem de ser uma letra ou underscore (`_`). Palavras-chave da linguagem C++, como `int`, `class` e `if`, não podem ser usadas.

2.6. ENTRADA USANDO CIN

Toda vez que precisa enviar dados para serem exibidos na tela, você faz uso de objeto do fluxo de saída padrão `cout`, como visto nos exemplos anteriores. Agora, se você necessita ler dados de entrada, pode utilizar o objeto `cin`. Como isso ocorre?

2.6.1. Entendendo o Uso do Objeto `cin`

`cin` é um objeto do fluxo de saída padrão predefinido em C++ para corresponder ao fluxo de saída. Esse fluxo é uma abstração que se refere a um fluxo de dados. O fluxo de entrada padrão, normalmente, flui do teclado, por onde o usuário entra com dados.

2.6.2. Operador `>>`

O fluxo representa os dados vindos do teclado, e o símbolo `>>` é um operador de *extração*. Dessa forma, a instrução

```
std::cin >> x1;
```

usa o operador de extração `>>` para aguardar o dado a ser digitado pelo usuário (por exemplo, um valor inteiro) e colocá-lo na variável inteira `x1`. Para entender melhor como isso funciona, vamos ver um exemplo.

Praticando um Exemplo. Agora você vai explorar um programa em C++ que calcula a soma de duas variáveis do tipo inteiro. Entretanto, diferentemente do exemplo da Listagem 2.3, aqui você deve solicitar que o usuário digite os dois valores inteiros que ele deseja somar. Para elaborar esse programa, você necessita declarar duas variáveis do tipo inteiro, exibir mensagem na tela solicitando que o usuário entre com os valores, somá-los e, em seguida, exibir o resultado da soma. Seguir esses passos resulta em um programa como mostrado na Listagem 2.4.

```
1. #include <iostream>
2. // Programa para ler dois números inteiros e calcular soma
3. int main()
4. {
5.     int x1, x2; // define as variaveis x1, x2, soma
6.     // Solicita que o usuário entre com dois valores inteiros
7.     std::cout << "\nDigite o primeiro valor inteiro e tecle
        Enter: \n";
8.     std::cin >> x1; // ler primeiro valor inteiro
9.     std::cout << "\nDigite segundo valor inteiro e tecle Enter:
        \n";
10.    std::cin >> x2; // ler segundo valor inteiro
11.    std::cout << "Soma = " << x1 + x2 << "\n\n";
12.    system("PAUSE");
13.    return 0;
14. }
```

Listagem 2.4

O código da Listagem 2.4 implementa o exemplo de um programa que soma dois inteiros fornecidos pelo usuário. A linha 1 é uma diretiva do pré-processador para incluir o arquivo `iostream` no momento da compilação do código. A linha 3 declara a função principal `main()` delimitada nas linhas 4 e 14. As variáveis do tipo `int` `x1` e `x2` são declaradas na linha 5 e lidas nas linhas 8 e 10, respectivamente. A linha 11 contém o objeto `cout`, que é usado para exibir o resultado da soma `x1 + x2`.

A instrução da linha 8, `std::cin >> x1`, faz o programa esperar a entrada de dados por parte do usuário. O dado (valor) digitado é colocado na variável `x1`. A palavra-chave `cin` é um objeto predefinido em C++ que corresponde ao fluxo de entrada de dados. Note que esse fluxo representa os dados vindos do teclado. O símbolo `>>` é o operador de *extração*.

Praticando um Exemplo. Foi observado que tanto o operador de inserção quanto o de extração podem ser utilizados em cascata. No programa da Listagem 2.4, é feito uso do operador de inserção `>>`. Agora, você é solicitado a modificar o programa da Listagem 2.4 de modo que também utilize o operador de extração em cascata para realizar a mesma funcionalidade do programa anterior. A solução é apresentada na Listagem 2.5.

```
1. #include <iostream>
2. // Programa para ler dois números inteiros e calcular soma
3. int main()
4. {
```

```
5.   int x1, x2; // define as variaveis x1, x2, soma
6.   // Solicita que o usuário entre com dois valores inteiros
7.   std::cout << "\nDigite um valor inteiro, tecle Enter e di-
    gite outro valor inteiro: \n";
8.   std::cin >> x1 >> x2;
9.   int soma = x1 + x2; // definição da variável soma
10.  std::cout << "Soma = " << soma << "\n\n";
11.  system("PAUSE");
12.  return 0;
13. }
```

Listagem 2.5

Você deve perceber que a principal diferença entre o código da Listagem 2.5 e a anterior está na entrada de dados (valores inteiros) que ocorre na linha 8, utilizando-se dois operadores de extração em cascata. Além disso, na linha 9, foi criada a variável soma e, na mesma linha, também foi definido o valor que ela deveria receber, ou seja, a variável soma foi inicializada com os valores que você digitou na entrada. Por exemplo, se você digitar os valores 5 e 65, o programa exibirá na tela o resultado Soma = 70.

■ *Note também que a definição de uma variável pode ocorrer no meio de um programa. Por exemplo, você poderia ter definido uma variável soma como inteiro e já determinado seu valor logo após a linha 10, fazendo: `int soma = x1 + x2`. Além disso, você pode usar os operadores de inserção e extração em cascata.*

2.7. MANIPULADORES

Em diversas situações, você pode necessitar de manipular ou alterar a forma pela qual os dados serão exibidos na tela de seu computador. Quando precisar fazer isso, poderá utilizar um recurso oferecido pela linguagem C++, denominado manipuladores. Como utilizar?

2.7.1. Entendendo Manipuladores

Manipuladores são operadores usados com o operador de inserção << para modificar (ou manipular) a forma como os dados são mostrados. A seguir, veremos os manipuladores *endl* e *setw*.

2.7.2. Operador endl

A última instrução cout do programa da Listagem 2.5 termina com o caractere '\n' (que significa quebra de linha ou nova linha). Entretanto, você pode substituir a linha 10 da Listagem 2.5, isto é:

```
std::cout << "Soma = " << soma << "\n\n";
```

por

```
std::cout << "Soma = " << soma << std::endl << std::endl;
```

Essa palavra endl, com a qual você ainda não está familiarizado, é um manipulador que causa o mesmo efeito do caractere '\n', ou seja, ela envia um caractere de nova linha '\n' para a saída padrão (monitor). Todavia, é um tanto mais clara. Agora, vamos analisar o programa da Listagem 2.6.

As linhas 2 e 3 contêm declarações using que eliminam a necessidade de utilizar o prefixo std em todos os locais no programa onde aparece cout e endl. Alternativamente, você pode substituir as linhas 2 e 3 da Listagem 2.6 por:

```
using namespace std;
```

Ao fazer isso, você estará utilizando a diretiva using, que informa ao compilador para fazer uso dos arquivos da biblioteca declarados no namespace std. Namespace é parte do padrão C++ ANSI.

palavra em tipo diferente

```
1. #include <iostream>
2. using std::cout;
3. using std::endl;
4. // Programa que mostra regiões do mundo e número de usuários
   da Internet
5. int main()
6. {
7.     long usuarios1 = 4514400, usuarios2 = 114304000,
8.         usuarios3 = 105096093, usuarios4 = 3284800,
9.         usuarios5 = 108096800, usuarios6 = 18068919,
10.        usuarios7 = 7620480, usuariosTotal = 360985492;
11. cout << "Região do mundo " << "Número de usuários da Inter-
    net em 2008" << endl
12. << "Africa " << usuarios1 << endl
13. << "Asia " << usuarios2 << endl
14. << "Europa " << usuarios3 << endl
15. << "Oriente Medio " << usuarios4 << endl
16. << "America do Norte " << usuarios5 << endl
17. << "America Latina " << usuarios6 << endl
18. << "Oceania e Australia " << usuarios7 << endl
19. << "Total de usuarios " << usuariosTotal << endl << endl;
```



```
20. system("PAUSE");
21. return 0;
22. }
```

Listagem 2.6

Continuando a análise do programa da Listagem 2.6, nas linhas 7 a 10 são definidas oito variáveis inteiras do tipo long que são inicializadas com quantidade de usuários da Internet em sete regiões do mundo, mais o valor total de usuários em 2008.

■ *Namespace é parte do padrão C++ ANSI. Ao fazer a declaração `using namespace std;` para o compilador, ele deve aceitar o uso de `cin`, `cout` e `endl` sem o prefixo `std`, como em `std::cout`. Você pode usar a diretiva `using namespace` quando precisar trabalhar com namespaces predefinidos, como `std`.*

Observe que `cout` na linha 11 e `endl` das linhas 11 a 19 não fazem uso do prefixo `std::`. Isso é possível porque você utilizou `using std` nas linhas 2 e 3 para `cout` e `endl`. Todavia, há um problema com a saída desse programa. Quando você o compila e executa com F9 no ambiente do Dev-C++, ele exibe a saída mostrada na Figura 2.9.

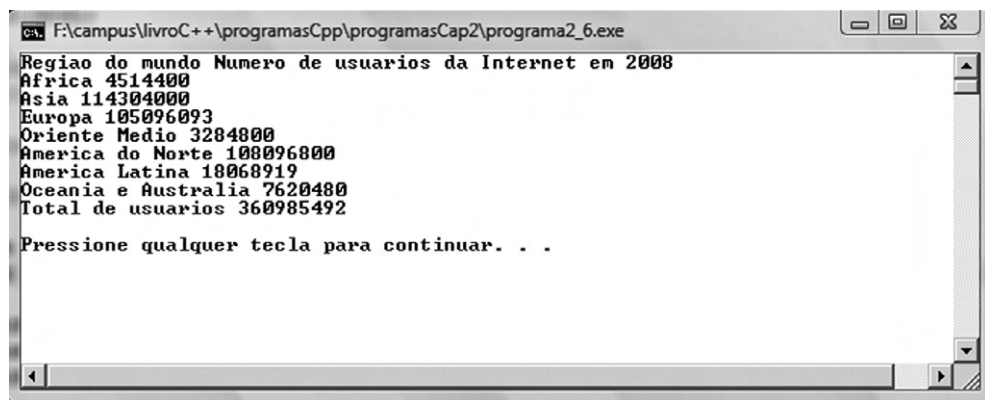


Figura 2.9 – Saída da execução do programa da Listagem 2.6.

Você percebe o inconveniente dessa saída? Na realidade, há dois inconvenientes. Primeiro, as quantidades de usuários estão completamente juntas às respectivas regiões do mundo; segundo, os números estão alinhados à esquerda. Como resolver isso?

Para resolver essa questão, precisamos de algum recurso para permitir a formatação da saída. É o que você irá explorar agora.

2.7.3. Operador setw

Outro manipulador oferecido pela linguagem C++ é `setw`. Assim, para começar, imagine que o espaço ocupado por `cout` corresponde a um *campo* com determinada largura. O valor default deve ser largo o suficiente para conter o valor que você quer exibir.

Por exemplo, o inteiro 12345 ocupará um campo de cinco caracteres de largura e a string “Pernambuco” ocupará um campo com 10 caracteres de largura. Contudo, em certas situações isso pode gerar resultados não desejados, como o que aconteceu no programa da Listagem 2.6.

Observe que, com esse formato, torna-se difícil comparar os números. Não seria melhor se eles (os números) estivessem alinhados à direita? Além disso, você pode inserir espaço entre os nomes das cidades e as respectivas quantidades de usuários da Internet para separá-los.

Na Listagem 2.7, você encontrará o programa que elimina esse problema utilizando o manipulador `setw`. O manipulador `setw` faz o número (ou string) que o segue no fluxo ser impresso dentro de um campo com *n* caracteres de largura, onde *n* é o argumento da instrução `setw(n)`. O valor é justificado à direita dentro do campo, conforme mostrado na Figura 2.10, que ilustra a saída da Listagem 2.7.

```
1. #include <iostream>
2. using std::cout;
3. using std::endl;
4. #include <iomanip>
5. using std::setw;
6. // Programa que mostra regiões do mundo e número de usuários
   da Internet
7. int main()
8. {
9.     long usuarios1 = 4514400, usuarios2 = 114304000, usuarios3
       = 105096093,
10.    usuarios4 = 3284800, usuarios5 = 108096800, usuarios6 =
       18068919,
11.    usuarios7 = 7620480, usuariosTotal = 360985492;
12.    cout << setw(20) << "Regiao do mundo " << setw(10) << "Nu-
       mero de usuarios da Internet em 2008" << endl
13.    << setw(20) << "Africa " << setw(10) << usuarios1 << endl
14.    << setw(20) << "Asia " << setw(10) << usuarios2 << endl
15.    << setw(20) << "Europa " << setw(10) << usuarios3 << endl
16.    << setw(20) << "Oriente Medio " << setw(10) << usuarios4 <<
       endl
17.    << setw(20) << "America do Norte " << setw(10) << usuarios5
       << endl
18.    << setw(20) << "America Latina " << setw(10) << usuarios6
       << endl
```

```

19.  << setw(20) << "Oceania e Australia " << setw(10) << usuarios7 << endl
20<< setw(20) << "Total de usuarios " << setw(10) << usuarios-
    Total << endl << endl;
21.  system("PAUSE");
22.  return 0;
23. }

```

Listagem 2.7

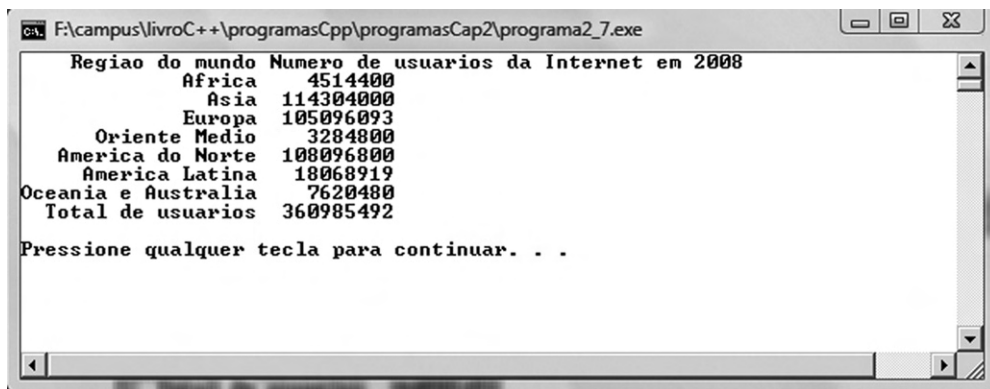


Figura 2.10 – Saída da execução do programa da Listagem 2.7.

Nas linhas 2 e 3, você faz uso de declarações `using` que eliminam a necessidade de utilizar o prefixo `std` em todos os locais no programa onde aparece `cout` e `endl`, isto é, nas linhas 12 a 20.

Na linha 5, outro manipulador de C++ `setw` é utilizado para definir a largura de um campo que será exibido na saída (monitor), como ocorre na linha 20, onde se tem o comando `setw(20)`, que determina o campo ocupará um campo de vinte caracteres de largura, sendo em parte ocupado pela string `Regiao do mundo`. ocupará um campo com dez caracteres de largura.

Observe que todo o conteúdo apresentado está alinhado à esquerda. Cabe ainda destacar que o uso do comando `setw` permite formatar o conteúdo que será exibido na saída (monitor).

2.8. TIPOS DE VARIÁVEIS (DADOS)

Você já verificou, na Seção 2.5, que variável é um elemento essencial nas linguagens de programação. As variáveis servem para receber e armazenar valor de um determinado tipo. Portanto, quando você declara uma variável, está atribuindo um nome a esse elemento que poderá receber uma variedade de valores. Vejamos agora outras alternativas que você tem quando elabora um programa.

2.8.1. Constantes do Tipo Inteiro

Constantes são elementos que não mudam seus valores durante a execução de um programa. Por exemplo, uma constante do tipo inteiro consiste em valores numéricos inteiros e, portanto, elas não possuem ponto decimal. Por exemplo, você pode definir uma constante do tipo inteiro como:

```
const int c = 100; // c é uma constante inteira que armazena o
valor 100.
```

Vale ainda ressaltar que uma constante é similar a uma variável, com a única diferença de que ela não tem o valor alterado. Além disso, há a obrigatoriedade de atribuir um valor (a constante). Isso será ilustrado no exemplo adiante.

■ *Note que você também pode ter um constante tipo real. Por exemplo se você quiser definir uma constante real você deveria declarar.*

2.8.2. Variáveis do Tipo char

Trata-se de outro tipo de variável, ou seja, do tipo char. Esse tipo serve para declarar e armazenar valores do tipo char. Uma variável do tipo char pode armazenar qualquer um dos caracteres da tabela ASCII. Exemplos desses caracteres são: 'a', 'B', '\$' etc. O exemplo da Listagem 2.8 ilustra o seu uso.

2.8.3. Constante do Tipo char

Constantes de caracteres são elementos colocados entre apóstrofos, conforme visto. Quando o compilador encontra tal constante de caractere, ele traduz em seu correspondente código ASCII. Por exemplo, a constante 'C' será armazenada como 67. Um programa ilustrando o uso de variáveis de caracteres e constante de caracteres é ilustrado na Listagem 2.8.

```
1. #include <iostream>
2. using std::cout;
3. using std::endl;
4. // Programa que mostra o uso de variáveis tipo char
5. int main()
6. {
7.     char varChar1 = 'A'; // define variável varChar1 como char
8.     char varChar2 = '\t'; // define variável varChar2 como char
9.     char varChar3 = 'C'; // define variável varChar3 como char
10.    cout << varChar1; // exibe conteúdo de varChar1
11.    cout << varChar2; // exibe conteúdo de varChar2
12.    varChar1 = 'B'; // atribui o caractere B a varChar1
```

```

13. cout << varChar1;      // exhibe conteúdo de varChar1
14. cout << '\n';        // exhibe caractere newline ou quebra de linha
15. cout << varChar3 << endl << endl; // exhibe conteúdo de var-
    Char3 e pula duas linhas
16. system("PAUSE");
17. return 0;
18. }

```

Listagem 2.8

Observe que as variáveis `varChar1`, `varChar2` e `varChar3` foram inicializadas, respectivamente, nas linhas 7, 8 e 9, ao mesmo tempo que foram definidas. Adicionalmente, note que a segunda variável do tipo `char`, `'\t'`, é um caractere de *tabulação*, fazendo com que a impressa continue na próxima para do *tab*. Isso é o que ocorre após a execução do programa, como ilustrado na Figura 2.11.

Perceba que, na linha 10, `cout` faz com que o conteúdo da variável `varChar1` seja exibido (isto é, A). Em seguida, `cout` na linha 11 faz com que um *tab* seja exibido. Isso resulta no avanço do espaço de tabulação, como mostrado na Figura 2.11. Posteriormente, na linha 12, o caractere B é atribuído a `varChar1` e depois exibido com `cout` na linha 13. Já na linha 14, o caractere *newline* (ou quebra de linha) faz o cursor de saída pular para a próxima linha, onde é exibido o conteúdo de `varChar3`.

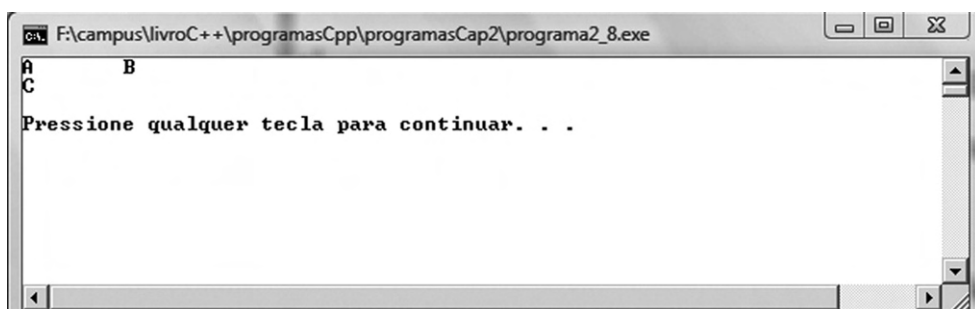


Figura 2.11 – Saída do programa da Listagem 2.8.

2.8.4. Variáveis do Tipo Ponto Flutuante ou Real

Variáveis do tipo ponto flutuante representam números reais com casa decimal, como 3,1415 e 0,0034. Elas têm a parte à esquerda inteira (denominada mantissa) e a parte à direita fracional. Variáveis de ponto flutuante representam números reais e podem ser declaradas como:

```

float x; // declara x como do tipo float
double y; // declara y como do tipo Double

```

Praticando um Exemplo. Para entender mais, nada melhor do que um exemplo para explorar como usar os recursos da linguagem C++. Então, calcule o volume de uma esfera. Solicite que o usuário digite o valor do raio da esfera. Note que o valor do raio deve ser um número real (ou de ponto flutuante); considere o valor da constante $PI = 3,14159$. Você sabe que a fórmula que determina o volume de uma esfera é $V = (4/3) \cdot PI \cdot R^3$. Para elaborar esse programa, deve seguir os passos realizados nos programas anteriores. A solução deste exercício é mostrada na Listagem 2.9.

```
1. #include <iostream>
2. using std::cout;
3. using std::cin;
4. using std::endl;
5. // Programa para determinar o volume de uma esfera de raio
   informado pelo usuário.
6. int main()
7. {
8.     const float PI = 3.14159;
9.     float volume, raio;
10.    cout << "Digite o raio da esfera: ";
11.    cin >> raio; // solicita valor do raio
12.    volume = 4,0/3.0*PI*raio*raio*raio; // fórmula para volume
       da esfera
13.    // Exibe o valor calculado do volume da esfera
14.    std::cout << "O volume de uma esfera de raio " << raio <<
       "cm = " << volume << "\n\n";
15.    system("PAUSE");
16.    return 0;
17. }
```

Listagem 2.9

Se executar o programa da Listagem 2.9, o programa solicitará que você digite o valor do raio da esfera. Vamos supor que você tenha digitado 3.5. Quando pressionar a tecla Enter, o programa exibirá o resultado ilustrado na Figura 2.12.

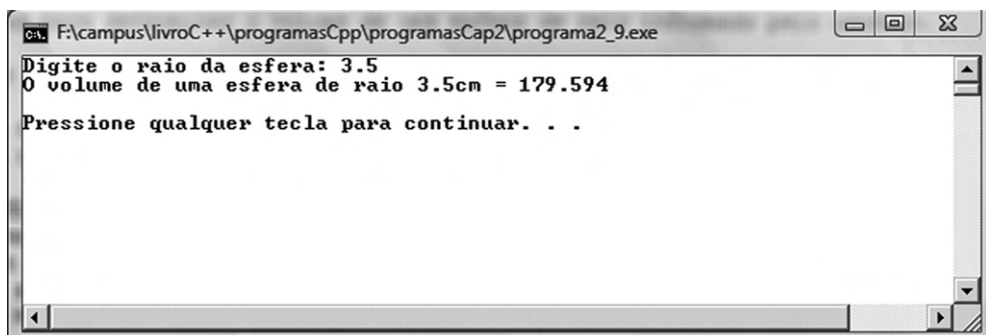


Figura 2.12 – Saída do programa da Listagem 2.9.

Nesse programa, é feito o uso de constante do tipo ponto flutuante na linha 8, onde se declara:

```
const float PI = 3.14159;
```

Cabe também destacar que, se você tiver outro valor constante, por exemplo, 314,15, do tipo float, que queira declarar, pode declarar como:

```
const float z = 314.15;
```

ou

```
const float z = 3.1415E2.
```

o qual estará escrito na notação exponencial como 3,1415E2 (ou seja, 3,1415 vezes 10^2).

2.8.5. Qualificador const

No exemplo da Listagem 2.9, além de mostrar o uso de variáveis do tipo ponto flutuante (para representar números reais), o programa também apresenta o qualificador const. A palavra-chave const (para constante) deve preceder o tipo de dado de uma variável.

const serve para especificar que o valor da variável não mudará durante a execução do programa. Em outras palavras: manterá o valor constante. Qualquer tentativa de alterar o valor de uma variável definida com esse qualificador gerará uma mensagem de erro do compilador. Isso assegura que uma variável não seja alterada inadvertidamente.

2.8.6. Tipo de Dados long

Outro tipo de inteiro (além de char e int) é long. Também pode ser escrito como long int. Ele pode conter inteiros, conforme a Tabela 2.1, que apresenta sumário de tipos. Adicionalmente, se você desejar criar uma constante do tipo long, bastará usar a letra L seguida do valor numérico, como mostrado no exemplo a seguir.

```
long x = 99999L; // declara x como do tipo long int e atribui o
                // valor 99999 à variável x
```

■ É importante observar que long é, na realidade, um modificador para o tipo int. Assim, quando você declara uma variável do tipo inteiro, essa variável está na faixa long int. Similarmente, você também pode reduzir a faixa de valores declarando, por exemplo:

```
short int y = 123;
```

2.8.7. Definições Múltiplas

Observe que, na linha 9 do programa da Listagem 2.9, as variáveis volume e raio foram declaradas como sendo do tipo float. Note ainda que elas estão separadas por vírgulas. Isso permite economizar espaço.

2.8.8. Sumário de Tipos de Variáveis

A Tabela 2.1 sumariza os tipos de variáveis discutidos neste capítulo.

Tabela 2.1

| Tipo de dado | Intervalo numérico | | Bytes de memória |
|--------------|-------------------------|-----------------------|------------------|
| | Inferior | Superior | |
| Char | -128 | 127 | 1 |
| Short | -32.768 | 32.767 | 2 |
| Int | -2.147.483.648 | 2.147.483.647 | 4 |
| Long | -2.147.483.648 | 2.147.483.647 | 4 |
| Float | -1.2*10 ⁻³⁸ | 3.4*10 ³⁸ | 4 |
| Double | -2.2*10 ⁻³⁰⁸ | 1.7*10 ³⁰⁸ | 8 |

Praticando um Exemplo. Escreva um programa que liste os limites inferiores e superiores dos tipos de dados estudados, bem como o número de bytes alocados para cada variável desses tipos. Para tanto, você deve fazer uso da diretiva `#include <limits>` que permite checar os limites inferiores e superiores dos tipos de dados, conforme solução deste exemplo mostrada na Listagem 2.10.

```
1. #include <iostream>
2. #include <limits>
3. using namespace std;
4. // Programa para obter os limites dos tipos de dados
5. int main()
6. {
7.     cout << "Limite inferior de <char>: " << (int)numeric_
8.         limits<char>::min() << endl;
9.     cout << "Limite superior de <char>: " << (int)numeric_
10.        limits<char>::max() << endl;
11.     cout << "Numero de bytes de <char>: " << sizeof(char) <<
12.        endl;
13.     cout << "Limite inferior de <short>: " << numeric_
14.        limits<short>::min() << endl;
15.     cout << "Limite superior de <short>: " << numeric_
16.        limits<short>::max() << endl;
```



```

12. cout << "Numero de bytes de <short>: " << sizeof(short) <<
    endl;
13. cout << "Limite inferior de <int>: " << numeric_
    limits<int>::min() << endl;
14. cout << "Limite superior de <int>: " << numeric_
    limits<int>::max() << endl;
15. cout << "Numero de bytes de <int>: " << sizeof(int) << endl;
16. cout << "Limite inferior de <long>: " << numeric_
    limits<long>::min() << endl;
17. cout << "Limite superior de <long>: " << numeric_
    limits<long>::max() << endl;
18. cout << "Numero de bytes de <long>: " << sizeof(long) <<
    endl;
19. cout << "Limite inferior de <float>: " << numeric_
    limits<float>::min() << endl;
20. cout << "Limite superior de <float>: " << numeric_
    limits<float>::max() << endl;
21. cout << "Numero de bytes de <float>: " << sizeof(float) <<
    endl;
22. cout << "Limite inferior de <double>: " << numeric_
    limits<double>::min() << endl;
23. cout << "Limite superior de <double>: " << numeric_
    limits<double>::max() << endl;
24. cout << "Numero de bytes de <double>: " << sizeof(double)
    << endl;
25. cout << "\n";
26. system("PAUSE");
27. return 0;
28. }

```

Listagem 2.10

Qual o propósito da Listagem 2.10? Verificar e exibir os limites inferiores e superiores dos diversos tipos de dados como char, int e float, além de retornar a quantidade de bytes alocados a cada um deles. Por exemplo, para o tipo char, na linha 7 da Listagem 2.10, você tem:

```
cout << "Limite inferior de <char>: " << (int)numeric_limits<char>::min()
<< endl;
```

que retornará o limite inferior de char com a instrução `(int)numeric_limits<char>::min()`. Para tanto, você adicionou a diretiva `#include <limits>` na linha 2 da Listagem 2.10. `limits` é um arquivo de cabeçalho que contém classes definindo os limites dos diversos tipos de dados.

Procedimento similar você faz na linha 8 para obter o limite superior de char. Já para obter a quantidade de bytes de char, você utiliza o operador sizeof(), como na linha 9. O mesmo procedimento se repete para os demais tipos. Agora, se você executar o programa da Listagem 2.10, ele exibirá a saída mostrada na Figura 2.13.

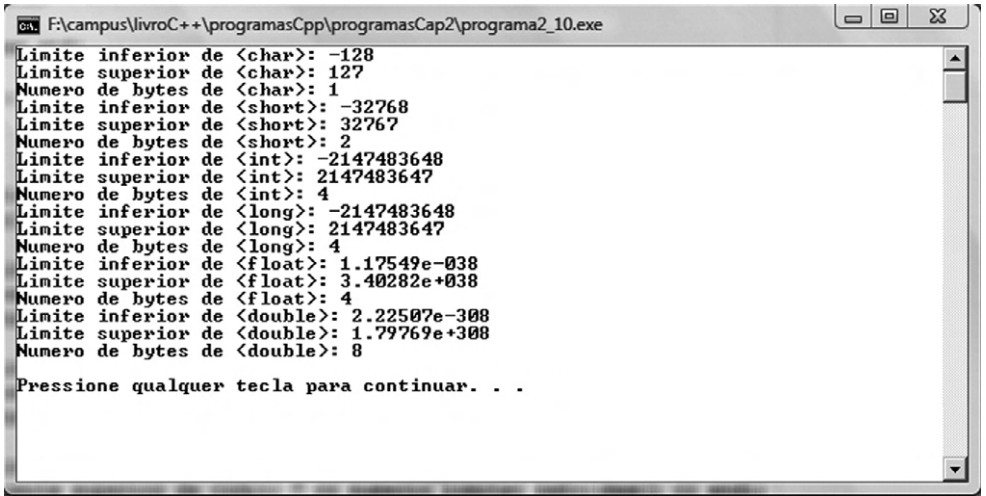


Figura 2.13 – Saída do programa da Listagem 2.10.

■ É importante observar que o inteiro do tipo long reduz a performance, se comparado ao tipo int. Isso ocorre porque ele é mais lento nas operações aritméticas e também requer mais espaço de memória. Fato similar é observado nos tipos double e float.

2.8.9. Tipos de Dados unsigned

Cabe destacar que, se você eliminar o sinal do tipo inteiro, o intervalo inferior resultante iniciará em 0, incluindo apenas os números positivos. Disso resulta que os números representados na Tabela 2.1 podem ser duas vezes maiores do que os que possuem sinal. A Tabela 2.2 sumariza essa informação.

Tabela 2.2

| Tipo de dado | Limite superior | Bytes de memória |
|--------------|-----------------|------------------|
| Char | 255 | 1 |
| Short | 65.535 | 2 |
| Int | 4.294.967.295 | 4 |

O programa da Listagem 2.11 exemplifica o uso de unsigned. Nesse exemplo, você declara e utiliza variáveis do tipo unsigned de modo que o intervalo compreende apenas valores positivos, como mostrado na Tabela 2.2.

Ao executar esse programa, você obtém a saída apresentada na Figura 2.14. No programa da Listagem 2.11, observe que o valor da variável do tipo `short int` excede o intervalo numérico após o cálculo efetuado na linha 8. Perceba que ambas as variáveis (`varComSinal` e `varSemSinal`) são multiplicadas por 5 e divididas por 3, conforme linhas 8 e 9, respectivamente. Essa multiplicação cria um resultado (150.000) que excede o intervalo da variável `short int`, gerando um resultado incorreto.

```

1. #include <iostream>
2. using namespace std;
3. // Programa para verificar faixa de valores suportados pelos
   tipos de dados.
4. int main()
5. {
6.     short int varComSinal = 30000; // signed short: -32768 to
       32767
7.     unsigned short int varSemSinal = 30000; // unsigned short:
       0 to 65535
8.     varComSinal = (varComSinal * 5)/3; // calculo de varComSinal
       excede intervalo
9.     varSemSinal = (varSemSinal * 5)/3; // calculo de varSemSinal
       não excede intervalo
10.    cout << "varComSinal = " << varComSinal << endl; // errado:
       -15536
11.    cout << "varSemSinal = " << varSemSinal << endl; // certo:
       50000
12.    system("PAUSE");
13.    Return 0;
14. }
```

Listagem 2.11

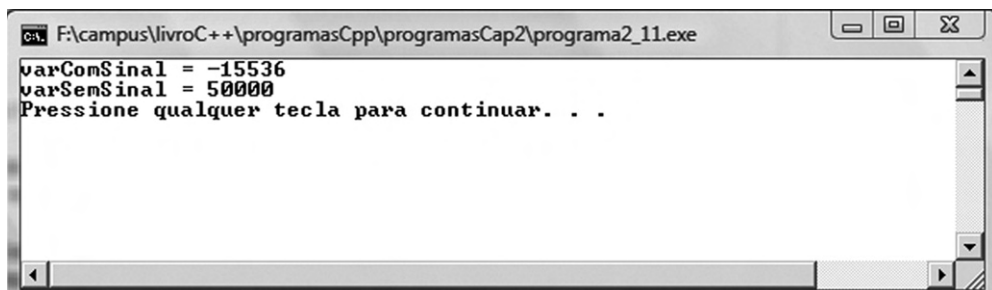


Figura 2.14 – Saída do programa da Listagem 2.11.

■ Exceder o intervalo do tipo `short int` com sinal (ilustrado na Listagem 2.11) pode levar a erros, como observado na Figura 2.14. O mesmo pode ocorrer para outros tipos, como `int` e `long`. Portanto, é preciso atenção durante o desenvolvimento de programas.

2.9. CONVERSÃO DE TIPOS

Vamos agora revisitar o programa da Listagem 2.9. No entanto, nessa nova versão daquele programa, você verá como podemos fazer uso da conversão de tipos suportada pela linguagem C++. O programa da Listagem 2.9, que solicita que o usuário digite o valor do raio de uma esfera e em seguida calcule o volume da esfera, foi reescrito de modo a ilustrar a conversão de tipos. O programa resultante é apresentado na Listagem 2.12. A Figura 2.15 ilustra a saída do programa da Listagem 2.12.

```
1. #include <iostream>
2. using namespace std;
3. // Programa para determinar o volume de uma esfera de raio
   informado pelo usuário.
4. int main()
5. {
6.     system("cls"); // Limpa a tela
7.     system("color F0"); // Torna a cor de fundo branca e as
   letras pretas
8.     const float PI = 3.14159;
9.     int raio;
10.    cout << "Digite o raio (de valor inteiro) da esfera: ";
11.    cin >> raio; // solicita valor do raio
12.    double volume = 4.0/3.0*PI*raio*raio*raio; // fórmula para
   volume da esfera
13.    cout << "O volume de uma esfera de raio " << raio << "cm =
   " << volume << "\n\n"; // Exibe o valor calculado do volume
   da esfera
14.    system("PAUSE");
15.    return 0;
16. }
```

Listagem 2.12

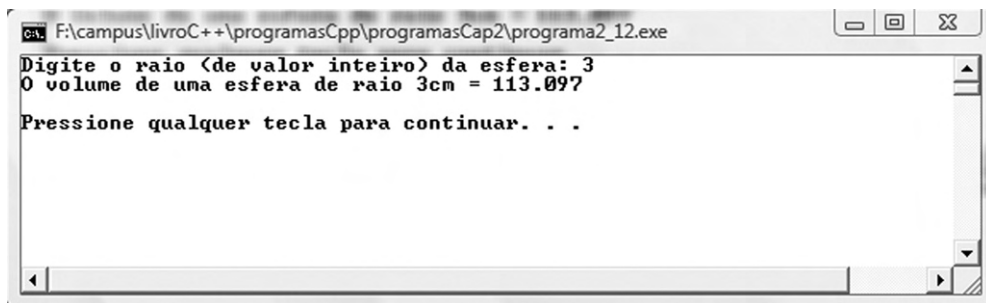


Figura 2.15 – Saída do programa da Listagem 2.12.

No programa da Listagem 2.12, a variável `raio` é declarada como sendo do tipo `int` e a constante `PI` como sendo do tipo `float`, conforme linhas 9 e 8, respectivamente.

Na linha 12, uma variável `int` é multiplicada por uma variável do tipo `float` e gera um resultado do tipo `double`. O compilador compila sem acusar qualquer erro. Por quê?

Tanto a linguagem C++ quanto a C assumem que o programador deve ter uma boa razão para fazer isto e, assim, procuram ajudá-lo nesse sentido. Essa é uma das razões da popularidade das linguagens C++ e C.

2.9.1. Conversões Automáticas

Quando o compilador se depara com uma situação similar à do programa `mixed.cpp`, ele segue a ordem de conversão, conforme mostrado na Tabela 2.3.

Tabela 2.3

| Tipo de dado | char int long float double long double |
|--------------|--|
| Ordem | Mais baixa ← → Mais alta |

Observe que, quando dois operadores do mesmo tipo são encontrados em uma expressão, a variável do tipo mais baixo é convertida para a do tipo mais alto. Assim, no programa da Listagem 2.12, o valor `int` de `raio` é convertido para o tipo `float` e, em seguida, armazenada em uma variável temporária antes de ser multiplicada pela constante `float PI`. O resultado, que é `float`, é convertido para `double` para que depois possa ser atribuído para a variável do tipo `double volume`.

2.9.2. Casts

Casts é um termo usado em C++ para designar a conversão de dados especificada pelo programador. Para que serve? Considere a situação em que o programador deseja fazer uma conversão de dados que não seria feita automaticamente pelo compilador. Para tanto, vamos explorar um exemplo.

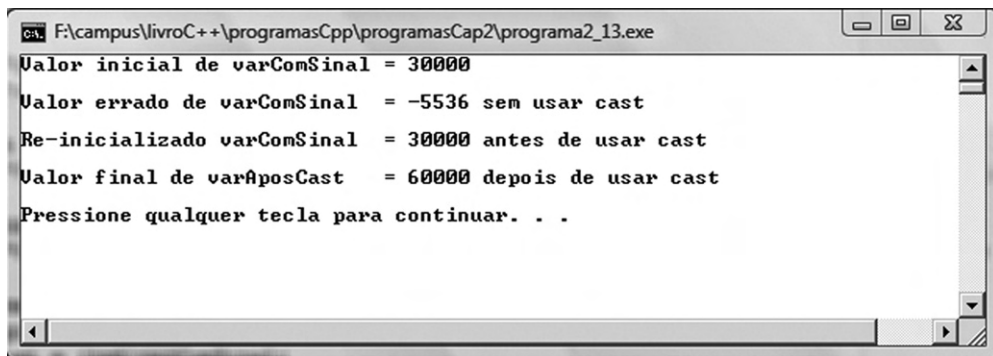
Praticando um Exemplo. Lembra-se do programa da Listagem 2.11? Vamos explorar aquele programa agora, com o objetivo de somar 30.000 à variável `varComSinal`, que foi declarada como do tipo `short int`. Isso vai gerar um erro porque o valor numérico resultante extrapola a faixa suportada por `short int`. Então, você deve forçar essa variável a ser do tipo `int` ou `long`, fazendo uso de *cast* e depois atribuindo o resultado a uma outra variável (do tipo `int`), como ilustrado na Listagem 2.13.

```
1. #include <iostream>
2. using namespace std;
3. // Programa para forçar a conversão de tipos de dados.
```

```
4. int main()
5. {
6.     short varComSinal = 30000; // signed short: -32768 to 32767
7.     cout << "Valor inicial de varComSinal = " << varComSinal <<
        endl << endl; // valor inicial: 30000
8.     varComSinal = varComSinal + 30000; // calculo de varComSinal
        excede intervalo
9.     cout << "Valor errado de varComSinal = " << varComSinal <<
        " sem usar cast\n" << endl; // errado: -5536
10.    varComSinal = 30000; // signed short: -32768 to 32767
11.    int varAposCast;
12.    varAposCast = (int)varComSinal;
13.    cout << "Re-inicializado varComSinal = " << varComSinal <<
        " antes de usar cast\n" << endl; // valor: 30000
14.    varAposCast = varAposCast + 30000; // calculo de varComSinal
        nao excede intervalo com uso de cast
15.    cout << "Valor final de varAposCast = " << varAposCast << "
        depois de usar cast\n" << endl; // certo: 60000
16.    system("PAUSE");
17.    return 0;
18. }
```

Listagem 2.13

No programa da Listagem 2.13, o uso de cast é feito na linha 12, precedendo a variável `varComSinal` com `int`., e depois a variável `varAposCast` é adicionada ao valor 30.000, resultando o valor 60.000, muito maior do que o que poderia conter uma variável do tipo `short int`. Isso gera um erro se cast não tiver sido feito. Você pode checar isso na linha 8, em que a variável `varComSinal` tem a ela somado o valor 30.000, e na linha 9 você pode verificar que o valor resultante extrapola a faixa numérica suportada para `short int`. A saída do programa da Listagem 2.13 é mostrada na Figura 2.16.

**Figura 2.16** – Saída do programa da Listagem 2.13.

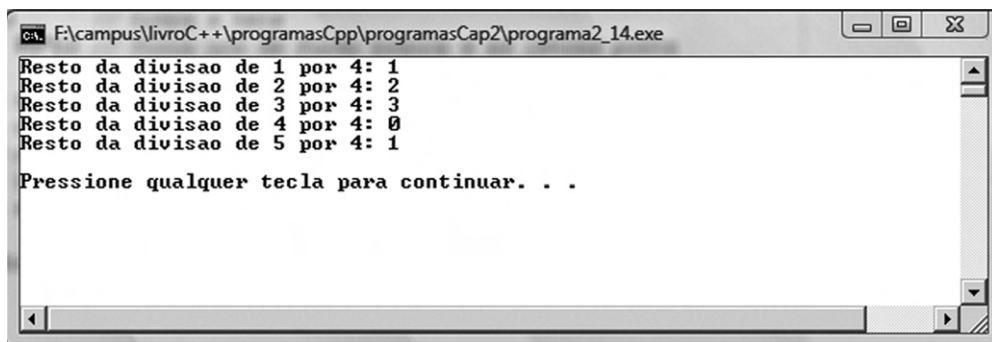
Assim, o que fizemos foi forçar uma variável `varComSinal` a ser do tipo `int` fazendo um *cast*, como ocorre na linha 12. Em outra situação, você pode declarar uma variável como sendo do tipo `float`, e o compilador dá uma mensagem de alerta informando que a conversão pode causar a perda de dígitos. Isso ocorre quando ele é transformado de um tipo (mais alto), como `float`, para um tipo menor, como `int`.

2.10. OPERADORES ARITMÉTICOS

Você recorda dos operadores aritméticos que usa no seu cotidiano para fazer as mais variadas operações? Então, ficará contente em saber que a linguagem C++ possui quatro operadores aritméticos: `+` (soma), `-` (subtração), `*` (multiplicação) e `/` (divisão). Além deles, há o operador `%` (módulo), o qual acha o resto de uma divisão. Esses operadores são binários, uma vez que atuam sobre dois operandos.

```
1. #include <iostream>
2. using namespace std;
3. // Programa para ilustrar o uso do operador módulo (%)
4. int main()
5. {
6.     cout << "Resto da divisao de 1 por 4: " << 1 % 4 << endl;
7.     cout << "Resto da divisao de 2 por 4: " << 2 % 4 << endl;
8.     cout << "Resto da divisao de 3 por 4: " << 3 % 4 << endl;
9.     cout << "Resto da divisao de 4 por 4: " << 4 % 4 << endl;
10.    cout << "Resto da divisao de 5 por 4: " << 5 % 4 << endl <<
        endl;
11.
12.    system("PAUSE");
13.    return 0;
14. }
```

Listagem 2.14



```

F:\campus\livroC++\programasCpp\programasCap2\programa2_14.exe
Resto da divisao de 1 por 4: 1
Resto da divisao de 2 por 4: 2
Resto da divisao de 3 por 4: 3
Resto da divisao de 4 por 4: 0
Resto da divisao de 5 por 4: 1

Pressione qualquer tecla para continuar. . .
  
```

Figura 2.17 – Saída do programa da Listagem 2.14.

Praticando um Exemplo. Neste exemplo, o interesse recai na utilização do operador módulo, que retorna o resto da divisão de dois operandos. O uso do operador módulo é feito na Listagem 2.14, onde há um conjunto de números de 1 a 5, os quais foram divididos por 4 e depois se obteve o resto da divisão por 4 (com a aplicação do operador módulo %). A saída do programa da Listagem 2.14 é exibida na Figura 2.17.

2.10.1. Operadores de Atribuição Aritmética

Os operadores de atribuição aritmética são, geralmente, usados para reduzir e/ou tornar mais claro o código. Esse tipo de operador é, na realidade, uma combinação de duas operações. Por exemplo, considere que você queira somar o inteiro 10 a uma variável x que foi inicializada com o valor 10. Para fazer isso, você escreveria:

Praticando um Exemplo. Neste exemplo, você verá como utilizar os operadores de atribuição aritmética. Para tanto, vamos fazer uso de vários desses operadores e verificar os resultados obtidos. O uso desses operadores é apresentado na Listagem 2.15, onde o valor 10 é adicionado à variável x utilizando o operador `+=` e subsequentemente usando outros operadores de atribuição aritmética. O resultado da execução do programa da Listagem 2.15 é exibido na Figura 2.18.

```
int x = 10;  
x = x + 10;
```

Observe que a instrução `x = x + 10` possui duas operações (soma e atribuição). Com o uso do operador de atribuição aritmética, você pode efetuar essas duas operações de modo mais conciso, simplesmente escrevendo: `x +=10;`

```
1. #include <iostream>  
2. using namespace std;  
3. // Programa para ilustrar o uso do operador módulo (%)  
4. int main()  
5. {  
6.     int x = 10;  
7.     cout << "Valor inicial de x: " << x << endl;  
8.  
9.     x += 10; // x = x + 10;  
10.    cout << "Resultado apos x +=10: " << x << endl;  
11.    x -= 5; // x = x - 10;  
12.    cout << "Resultado apos x -= 5: " << x << endl;  
13.    x *=2; // x = x * 2;  
14.    cout << "Resultado apos x *= 2: " << x << endl;  
15.    x /= 5; // x = x / 5;  
16.    cout << "Resultado apos x /= 5: " << x << endl;
```



```

17. x %= 5; // x = x % 5;
18. cout << "Resultado apos x %= 5: " << x << endl << endl;
19. system("PAUSE");
20. return 0;
21. }

```

Listagem 2.15

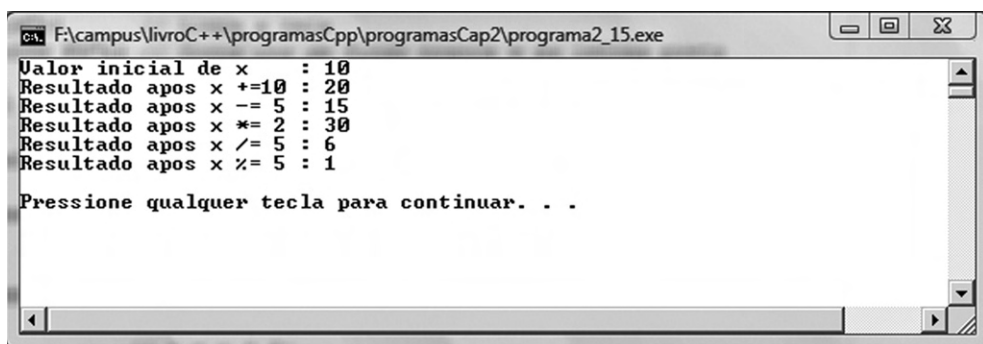


Figura 2.18 – Saída do programa da Listagem 2.15.

■ Utilizar o operador de atribuição aritmética é uma boa prática de programação. Como resultado, o código produzido fica mais conciso em trechos do programa como, por exemplo, num loop (ou laço) em que você precise atualizar uma variável ou contador. Portanto, você contribui para melhorar o entendimento do programa.

2.10.2. Operadores de Incremento

Você deve ter observado que, nos operadores de atribuição aritmética apresentados anteriormente, é possível adicionar 1 à variável x . Para tanto, você escreveria: $x += 1$. Perceba que aqui você está incrementando a variável x de 1. Em tais situações em que você necessita adicionar o valor 1 a uma variável, você tem duas opções:

```

x = x + 10;
x += 1;

```

Todavia, você pode fazer uso dos operadores de incremento, além dos operadores de decremento, conforme ilustrado na Listagem 2.16. O resultado da execução do programa desse programa é mostrado na Figura 2.19.

```

1. #include <iostream>
2. using namespace std;
3. // Programa para ilustrar o uso do operador módulo (%)
4. int main()
5. {
6.     int x = 10;
7.     cout << "Valor inicial de x: " << x << endl; // mostra x = 10

```

```
8.   cout << "Resultado apos ++x: " << ++x << endl; // mostra x
    = 11
9.   cout << "Valor seguinte de x: " << x << endl; // mostra x
    = 11
10.  cout << "Resultado apos x++: " << x++ << endl; // mostra x
    = 11
11.  cout << "Valor seguinte de x: " << x << endl; // mostra x
    = 12
12.  cout << "Resultado apos --x: " << --x << endl; // mostra x
    = 11
13.  system("PAUSE");
14.  return 0;
15. }
```

Listagem 2.16

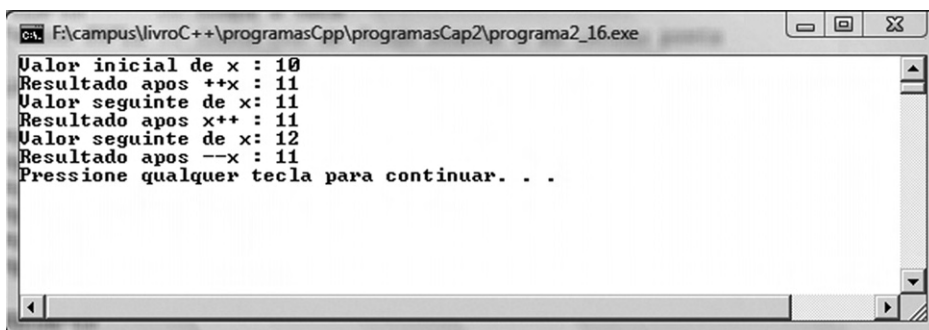


Figura 2.19 – Saída do programa da Listagem 2.16.

■ Utilizar o operador de incremento permite você um código mais conciso, geralmente em partes do programa onde você um loop (ou laço). Note também que existem duas formas de fazer o incremento (prefix ou postfix). No incremento pré-fixado como `++x`, o valor de `x` é primeiro incrementado e depois usado na instrução. Já no incremento pós-fixado, o valor (atual) de `x` é primeiro utilizado na iteração atual da instrução e apenas depois incrementado.

2.11. FUNÇÕES DE BIBLIOTECA

Muitas das funcionalidades oferecidas pela linguagem C++ estão disponíveis nos arquivos da biblioteca. Esses arquivos contêm funções de biblioteca. Dessa forma, quando você escreve um programa que inclui esses arquivos (da biblioteca), o compilador busca os arquivos de biblioteca, os quais contêm funções (de biblioteca) que serão utilizadas no programa.

Essas funções executam, dentre outras coisas, acesso a arquivos, cálculos matemáticos e conversões de dados. Neste capítulo, esse conteúdo não será detalhado. A apresentação será mais completa no Capítulo 5, quando você estudará funções, bem

como nos capítulos subsequentes. Embora esse conteúdo seja tratado em detalhes posteriormente, é sempre bom ver um exemplo para entender melhor os conceitos.

Praticando um Exemplo. Aqui, você irá utilizar o arquivo `math.h` da biblioteca de C++, o qual provê suporte a várias funções e, dentre elas, a função `sqrt(n)`, que retorna a raiz quadrada do valor n que você passa como parâmetro. Neste exemplo, o usuário é solicitado a digitar um valor numérico, e o programa calcula e exibe o valor da raiz quadrada desse valor. O código que implementa essa funcionalidade está na Listagem 2.17.

```

1. #include <iostream>
2. #include <math.h> // para a funcao sqrt()
3. using namespace std;
4. // Programa para ilustrar o uso de biblioteca
5. int main()
6. {
7.     double n, raiz;
8.     cout << "Digite um numero: "; // solicita o usuario digitar
        um numero
9.     cin >> n; // ler o numero digitado
10.    raiz = sqrt(n); // obtem a raiz quadrada de n
11.    cout << "Raiz quadrada = " << raiz << endl << endl;
12.    system("PAUSE");
13.    return 0;
14. }
```

Listagem 2.17

A Listagem 2.17 mostra o exemplo que faz uso de uma função da biblioteca `sqrt(n)` para calcular a raiz quadrada de um número n . A saída do programa da Listagem 2.17 é ilustrada na Figura 2.20. Você deve perceber que, para fazer uso da função `sqrt()`, é necessário saber os tipos de dados corretos a serem entrados para o argumento, bem como o tipo de valor retornado pela função. Para tanto, faz-se necessário consultar a documentação da biblioteca usada.

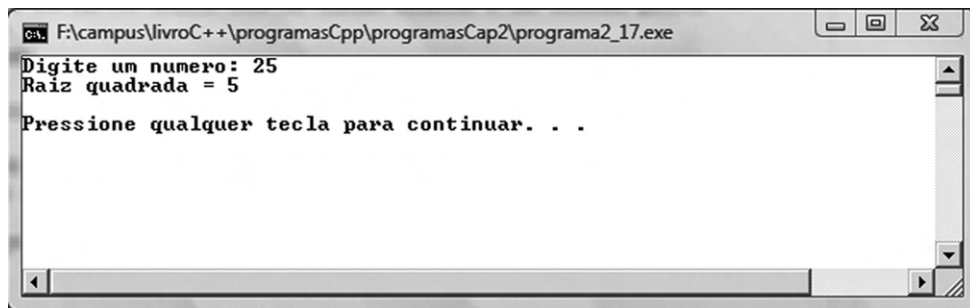


Figura 2.20 – Saída do programa da Listagem 2.17.

■ É importante observar que os símbolos `<e>` devem ser utilizados para arquivos do tipo `#include` que você encontra no subdiretório `include` do DevC++. Todavia, se você tem um arquivo do tipo `#include` que escreveu e que também está no mesmo subdiretório onde se encontra o programa, deve usar aspas como, por exemplo, em:

```
#include "meuArquivo.h";
```

RESUMO

Neste capítulo, você teve a oportunidade de conhecer e explorar os principais componentes de um programa na linguagem C++. Dentre os componentes apresentados, você explorou exemplos de entrada e saída de dados, declaração, uso e conversão de variáveis de diversos tipos, além de conhecer vários operadores e funções da biblioteca C++. Juntamente com esses componentes da linguagem, novos conceitos foram apresentados de modo a proporcionar mais recursos que você pode utilizar na solução de problemas. No próximo capítulo, você irá explorar outros operadores e mecanismos de controle da execução de um programa C++ que engloba laços e decisões.

QUESTÕES

1. Quais os principais componentes de um programa orientado a objetos?
2. O que são diretivas do pré-processador? Para que servem?
3. Qual a diferença de variáveis e constantes? Como elas podem ser definidas em C++?

EXERCÍCIOS

1. Faça uma pesquisa visando responder à seguinte questão: o que são manipuladores? Em que situações seu uso é interessante?
2. Escreva um programa em C++ que gere a saída a seguir. Utilize operador de incremento para incrementar 10 e decremento na geração do último número.
 - a) 10
 - b) 20
 - c) 30
 - d) 19x
3. Escreva um programa em C++ que gere a saída a seguir. Utilize operador de incremento para incrementar 10 e decremento na geração do último número.
 - a) 10
 - b) 20
 - c) 30

4. Escreva um programa em C++ que gere a saída a seguir. Utilize operador de incremento para incrementar 10 e decremento na geração do último número.
 - a) 2008 100
 - b) 2009 800
 - c) 2010 1.200
 - d) 2011 1.500
 - e) 2012 1.800
5. Escreva um programa em C++ que solicite ao usuário para entrar com um valor de temperatura em Fahrenheit, faça a conversão desse valor para Celsius e mostra o resultado. Use a fórmula a seguir considerando ftemp e ctemp como temperatura em Fahrenheit e Celsius, respectivamente:
$$ctemp = (ftemp - 32) * 5 / 9;$$
6. Modifique o programa do Exercício 5 de modo que ele solicite ao usuário para entrar com uma quantia em dólares e faça a conversão desse valor para reais.