CIN/COUT

A linguagem C++ possui uma ótima biblioteca de classes relacionadas ao controle de entrada e saídas de dados. Desde o início da apostila temos feito uso de algumas facilidades fornecidas por esta biblioteca, especificamente, os comandos cout e cin. Como você deve ter percebido, a classe cout serve para exibir valores - seja o valor de uma variável ou uma frase – enquanto que cin serve para armazenar valores recebidos através do teclado em variáveis. Tínhamos na linguagem C as funções printf e scanf para executar estas mesmas funções. Na verdade, printf e scanf também estão presentes em C++ (assim como todas as funções padrões de C), e podemos utilizá-las caso desejemos. Porém, os comandos – ou, utilizando um termo mais tecnicamente apropriado, classes – cin e cout facilitam muito a vida do programador, por serem mais “inteligentes” que printf e scanf.

**5.1 – Utilização de cout**

Como já dissemos, cout exibe valores na tela. A sintaxe utilizada é:

cout << <valor, string, variável, ponteiro, etc>;

Utilizamos cout em conjunto com o operador de inserção <<. Note que símbolo << também é utilizado pelo operador de bitwise shift para a esquerda (move bits de uma variável para a direção esquerda), entretanto não precisamos nos preocupar com isto: C++ sabe diferenciar quando estamos utilizando um operador ou o outro, através do contexto.

O operador << indica ao comando cout que um dado deve ser exibido na tela, além de identificar automaticamente qual o tipo deste dado e como ele deve ser formatado para exibição na tela. Assim, não precisamos informar à cout que estamos enviando um inteiro, um real ou uma string, como fazíamos em C: o operador << se encarrega desta identificação, bastando para o operador indicar o nome da variável. Abaixo temos a lista de todos os tipos básicos de C++ reconhecidos pelo operador de inserção:

* unsigned char
* signed char
* char
* short
* unsigned short
* int
* unsigned int
* long
* unsigned long
* float
* double
* long double

O operador de inserção também fornece facilidades para a exibição de strings. Além dos tipos básicos mostrados acima, o operador de inserção também reconhece os seguintes tipos de ponteiros:

* const signed char \*
* const unsigned char \*
* const char \*
* void \*

Ponteiros serão explicados com maior propriedade no módulo 8, mas precisamos saber que C e C++ representam strings utilizando ponteiros para o endereço de memória da string. Este ponteiro pode ser o nome de uma variável matriz de tipo char, um ponteiro de tipo char ou então uma frase entre aspas. O operador de inserção reconhece cada um destes casos e exibe na tela a string de texto. Por exemplo:

char nome[20] = “Jose das Couves”;  
char \* nome2 = “Jose das Galinhas”;  
cout << “Jose das Dores”;  
cout << nome;  
cout << nome2;

Estas 3 utilizações de cout exibem as strings de texto na tela. Veremos mais adiante que toda string possui um caractere de término, “\0”, que indica para o compilador que a string terminou. Nestes três casos, o operador de inserção orienta-se por este “\0” para terminar a exibição das strings.

É importante ressaltar que o operador de inserção não reconhece automaticamente matrizes numéricas e não as exibe automaticamente na tela. Para fazer isso, precisaremos utilizar os métodos tradicionais envolvendo loops para mostrar cada membro da matriz de uma vez. Este assunto será discutido no módulo 7, dedicado para matrizes.

**5.2 – Overload do operador de inserção**

“Overload do operador de inserção” nada mais é do que utilizar o operador de inserção várias vezes na mesma chamada de cout. Por exemplo:

cout << ”O valor da variável X é :“ << X;

A linha de comando acima exibe dois valores: a string “O valor da variável X é :” e a variável X. Note que utilizamos duas vezes o operador de inserção, sempre antes do valor a ser exibido. Podemos repetir o operador de inserção quantas vezes precisarmos na mesma linha de cout.

**5.3 – Formatação de exibição com cout**

A função printf fornecia aos usuários de C múltiplas maneiras de formatar a exibição dos dados na tela do computador. A classe cout também fornece as mesmas facilidades para os usuários de C++. Nas subseções abaixo descreveremos como fazer vários tipos de formatação de dados utilizando cout e o operador de inserção.

**5.3.1 – Escolhendo a Base Numérica**

Podemos escolher a base numérica que utilizaremos para representar números inteiros. Para isto, devemos utilizar os comandos:

cout << hex;  
cout<< oct;  
cout<< dec;

Após utilizar qualquer um destes comandos, sempre que pedirmos para cout exibir um número inteiro na tela, o comando automaticamente converterá o número para a base definida previamente. Por exemplo:

int numero = 10;  
cout << hex;  
cout << numero;

Dessa forma, cout exibirá na tela não o número 10 como estamos acostumados, mas a letra a, que representa 10 na base hexadecimal. Note que podemos utilizar o operador de inserção duas vezes para deixar o código mais compacto e obter o mesmo resultado:

int numero = 10;  
cout << hex << numero;

Não se esqueça de retornar para a base decimal com “cout << dec;” após exibir valores na base hexadecimal ou octal!

**5.3.2 – Formatação de números reais**

Podemos escolher também a notação utilizada para exibição de números reais. Com o comando:

cout << fixed;

Instruímos o programa a exibir valores reais usando a notação de ponto fixo ( por exemplo, 3.1214). Da mesma forma, com o comando:

cout << scientific;

Instruímos o programa a utilizar a notação científica (por exemplo, 3.21E-2).

**5.3.3 – Espaçamento de Texto**

O comando cout permite também escolher um número mínimo de caracteres para ser exibido na tela. Isto é feito utilizando o método:

cout.width ( x );;

Onde substituímos x pelo número mínino de caracteres a ser exibido na tela. Após a utilização deste método, utilizamos o comando cout para exibir o valor desejado, como no exemplo abaixo:

int variavel = 10;  
cout.width ( 5 );  
cout << variavel;

Neste exemplo, foi especificado cout.width (5); e o valor a ser exibido é 10. Assim, cout predecerá o valor 10 com três espaços em branco.

Observe que o valor especifica o número mínimo de caracteres que a saída consumirá. Se o valor a ser exibido requer mais caracteres do que o especificado, será usado o número de caracteres necessários para exibir o valor corretamente.

É importante observar também que o método cout.width só é válido para a próxima utilização de cout: após isto, o número mínimo de caracteres volta a ser zero.

Podemos também determinar o caractere a ser utilizado para preencher os espaços em branco de um campo de exibição. Isto é feito com o seguinte método:

cout.fill ( “caractere”);

Onde substituímos “caractere” pelo caractere que será exibido. É necessário utilizar aspas entre o caractere, para indicar para o compilador que não se trata de uma variável.

O exemplo abaixo mostra a utilização conjunta destes dois métodos:

int variavel = 10;  
cout.width ( 8 );  
cout.fill(“0”);  
cout << variavel;

Este exemplo fará a seguinte exibição na tela, preenchendo os espaços em branco determinados por cout.width com o caractere 0, determinado por cout.fill:

00000010

**5.3.4 – Precisão de Variáveis Reais**

O seguinte método é utilizado para fixar a precisão de variáveis reais, ou seja, o número mínimo de casas decimais à serem exibidas após a vírgula em um valor real:

cout.precision ( valor );

Por default, C++ utiliza 6 casas decimais após a vírgula. Quando alteramos o valor da precisão, este novo valor vale para todas as utilizações futuras de cout.

**5.3.5 – Alinhamento de Texto**

A escolha da direção de alinhamento de texto é feita da seguinte forma utilizando cout:

Alinhamento à direita: cout << right << <valor a ser exibido>;  
Alinhamento à esquerda: cout << left << <valor a ser exibido>;

Por default, todos os valores exibidos em um programa são automaticamente alinhados à direita. Quando mudamos o modo do alinhamento de texto, ele permanecerá dessa forma até que o alteremos novamente.

**5.4 – Utilização de cin**

Utilizamos o comando cin para obter valores do usuário através do teclado. A sintaxe utilizada é a seguinte:

cin >> variavel\_destino;

Assim como cout, cin utiliza um operador (nesse caso, o operador de extração >> ) para identificar o tipo de variável onde o valor será armazenado e encontrar o endereço de memória correto. Ao contrário da função scanf, utilizada na linguagem C, não é preciso especificar qual o tipo de valor será enviado pelo teclado pois o operador de extração faz as conversões necessárias. Podemos utilizar cin para ler valores inteiros, reais e strings de caracteres.

Na maioria dos casos, o comando cin cobre nossas necessidades de entrada de dados via teclado. Entretanto, quando precisamos ler strings com mais de uma palavra, como por exemplo frases ou nomes, cin apresenta certos “problemas”. Isto acontece por causa da maneira que C++ trata os espaços em branco em uma entrada via teclado.

Espaços em branco são considerados fim de entrada pelo comando cin; ao invés de descartar os caracteres que vierem após o espaço em branco, C++ os guarda em um buffer (uma espécie de “reserva” ou pilha de dados). Quando cin for chamado novamente, antes de ler a nova entrada do teclado, o programa primeiro utiliza os dados que estão nesse buffer. Assim, temos a impressão que a nova entrada de dados foi descartada pelo programa, mas na verdade ela foi jogada no buffer, esperando uma nova chamada de cin. Para solucionar este problema, utilizamos o métdo de cin cin.getline.

**5.5 – Método de cin: cin.getline**

O método cin.getline é muito útil para receber strings de caracteres com espaços, como frases. Este método lê uma linha inteira, marcando o fim da entrada de dados pelo uso da tecla <ENTER> indicando a entrada de uma nova linha. Abaixo temos a sintaxe do método:

cin.getline ( <matriz\_destino>, <limite de caracteres>);

O primeiro argumento é a matriz de caracteres para onde serão enviados os dados recebidos. É necessário declarar uma matriz de caracteres previamente ao uso deste método. O segundo argumento é o número máximo de caracteres que será lido pelo método, menos o caractere \0 indicando o fim da string. Assim, se especificarmos um número máximo igual a 20, este comando lerá 19 caracteres e descartará os próximos caracteres entrados pelo usuário, até que a tecla <ENTER> seja pressionada. Um espaço será sempre utilizado para marcar o fim da string através do caractere \0.

Um exemplo da utilização de cin.getline:

char matriz[60];  
cin.getline ( matriz, 50 );  
cout >> matriz;

Nesse caso, o método cin.getline lerá os próximos 49 caracteres (lembre-se do espaço reservado para o caractere fim\_de\_string \0) que o usuário entrar através do teclado. A leitura será feita até que ele aperte a tecla <ENTER> interrompendo o comando. Caso o usuário entre mais do que 50 caracteres , os próximos serão descartados pelo programa.