Material para uso exclusivo de aluno matriculado em curso de Educação a Distância da Rede Senac EAD, da disciplina correspondente. Proibida a reprodução e o compartilhamento digital, sob as penas da Lei. © Editora Senac São Paulo.

Capítulo 5

Linguagem de programação imperativa

Uma linguagem de programação é uma notação formal que utilizamos para descrever precisamente soluções para problemas. Essa solução é alcançada em duas etapas. Na primeira delas, a solução é representada pela escrita ou desenvolvimento do algoritmo, que pode envolver um pensamento considerável por parte do solucionador. Na segunda etapa, uma vez que um algoritmo de solução foi pensado, ele precisa ser codificado em uma notação formal (uma linguagem de programação) para que possa ser executado em um computador.

A dificuldade envolvida na segunda etapa é determinada pela linguagem de programação que se usa para codificar o algoritmo de solução.

As linguagens de programação devem ser cuidadosamente projetadas para garantir que essa etapa seja fácil. A facilidade de programar o algoritmo de solução na linguagem de programação escolhida depende de quão estreita é a lacuna semântica entre essa linguagem de programação e o nível de abstração no qual o procedimento de solução foi concebido pelo usuário. Em geral, quanto mais baixo nível for uma linguagem, maior será a lacuna semântica.

Uma linguagem de baixo nível é a linguagem de máquina, representada por bits zeros e uns. Esta é muito mais complexa para o programador, além de ter o aprendizado dificultado. Por outro lado, tem melhor aproveitamento da arquitetura da máquina.

Uma linguagem de alto nível é aquela escrita em inglês formal, com palavras e instruções mais próximas de nosso entendimento, o que requer um maior nível de abstração, porém, o aprendizado é menos complexo.

Diferentes linguagens de programação têm estruturas para oferecer suporte aos recursos necessários para resolver problemas em domínios específicos. Por exemplo, a linguagem Cobol tem suporte para resolver problemas relacionados a negócios; já a linguagem C, para problemas de programação de sistemas; e Lisp/Prolog, para aplicativos de inteligência artificial.

A linguagem de programação escolhida para expressar o procedimento de solução para um determinado problema tem um grande impacto na qualidade do código escrito (sua legibilidade, escrita e manutenção) e na produtividade do programador.

Neste capítulo, conheceremos o conceito de paradigmas de linguagem de programação. Começaremos a estudar alguns dos principais paradigmas, bem como a sintaxe para a escrita de códigos imperativos. Além disso, veremos na prática como escrever os primeiros códigos na linguagem C# (CSharp).

Material para uso exclusivo de aluno matriculado em curso de Educação a Distância da Rede Senac EAD, da disciplina correspondente. Proibida a reprodução e o compartilhamento digital, sob as penas da Lei. © Editora Senac São Paulo.

Material para uso exclusivo de aluno matriculado em curso de Educação a Distância da Rede Senac EAD, da disciplina correspondente. Proibida a reprodução e o compartilhamento digital, sob as penas da Lei. © Editora Senac São Paulo.

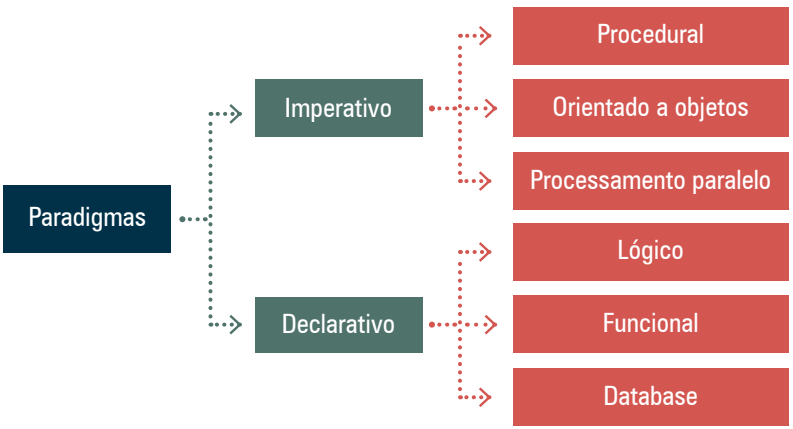
1 Programação imperativa

Antes de apresentar o conceito de programação imperativa, vamos entender o conceito de paradigma de linguagem de programação.

Não há uma consideração ou documentação oficial apontando exatamente quantas linguagens de programação existem, porém, mais delas estão sendo criadas exatamente para resolver problemas, sejam para fins gerais ou específicos. Portanto, é útil organizar essas linguagens em categorias que nos ajudem a entender as semelhanças e as diferenças entre elas, de modo que facilite, inclusive, o aprendizado.

Embora não haja uma maneira definitiva de classificar ou agrupar os tipos de linguagens de programação, uma abordagem comum é dividi-los em estilos de programação ou, formalmente definido, em paradigmas centrais, conforme ilustra a figura 1. Isso determina como seu código é estruturado e organizado.

Figura 1 – Paradigmas de linguagem de programação



Na figura 1, é possível observar que as linguagens de programação podem ser divididas em dois paradigmas ou estilos de programação distintos: o imperativo e o declarativo.

A programação imperativa expressa comandos para o computador executar – ela se concentra no processo explícito de o que um programa deve fazer e como esse programa deve fazer (FORBELLONE; EBERSPACHER, 2005). Se quisermos obter a soma de dois números, este código pode ser expresso conforme o exemplo a seguir:

```
1  int numA, numB, soma;
2  numA = 4;
3  numB = 5;
4  soma = numA + numB;
5  Console.WriteLine("O resultado desta operação é " + soma);
```

Observe que o código apresentado é baseado em declarações que alteram o estado do programa, dizendo ao computador o que fazer e como fazer as coisas. Em outras palavras, seu código é baseado na definição de variáveis e na alteração dos valores dessas variáveis.

A programação declarativa é de nível superior, focada na lógica e nos conceitos, em vez do fluxo imperativo. Em outras palavras, está preocupada somente com o que precisa ser feito, e não exatamente como deve ser feito. Por exemplo, o código HTML ** informa ao navegador para exibir a imagem de um carro sem dizer ao navegador como ele deve fazer.

Para entender melhor a diferença entre ambos os paradigmas, considere que André está prestes a visitar Bia. Ele não sabe onde a amiga mora, logo:

- pelo paradigma imperativo, Bia fornece instruções detalhadas para uma rota específica até a sua casa;
- pelo paradigma declarativo, Bia fornece o seu endereço ao amigo e o deixa decidir qual caminho seguir.

Material para uso exclusivo de aluno matriculado em curso de Educação a Distância da Rede Senac EAD, da disciplina correspondente. Proibida a reprodução e o compartilhamento digital, sob as penas da Lei. © Editora Senac São Paulo.

Compreendido o conceito e a diferença dos paradigmas, definimos a seguir cada um dos paradigmas do tipo imperativo e declarativo.

Entre os paradigmas imperativos, de acordo com a figura 1, temos:

- Paradigma procedural: está relacionado à chamada de procedimentos, os quais podem ser sub-rotinas, métodos ou funções. Esses procedimentos são coleções de instruções que são executadas sequencialmente para resolver um problema específico. Por exemplo, considere que você preparará uma macarronada. Então, há uma função chamada *preparar o molho* e outra função *cozinhar a massa*. Neste paradigma, você deve invocar uma função por vez. Desse modo, somente após uma tarefa for concluída é que a outra será executada. Entre as principais linguagens procedurais, destacam-se: C, C++, C#, Java e Pascal.

- Paradigma orientado a objetos: considera que o código pode ser dividido em objetos (como os objetos da vida real), os quais possuem propriedades e executam diferentes ações. Os objetos podem interagir uns com os outros para alcançar um objetivo ou resultado desejado. A popularidade e o alto uso deste paradigma estão no fato de que este imita a nossa visão do mundo real, tornando-o relativamente simples de entender. Entre as principais linguagens de programação orientada a objetos, destacam-se: C++, C#, Java e Python.

- Paradigma de processamento paralelo: usa o princípio de dividir diferentes partes de uma tarefa entre vários processadores, os quais trabalham simultaneamente para resolver um problema. Considere o exemplo anterior da tarefa *preparar a macarronada*. Agora, em vez de você fazer tudo sozinho, uma pessoa lhe ajudará a preparar o molho, outra irá ralar o queijo e, ao mesmo tempo, você cozinhará a massa – o tempo para preparar toda a tarefa é reduzido, uma vez que três pessoas fazem coisas diferentes.

Material para uso exclusivo de aluno matriculado em curso de Educação a Distância da Rede Senac EAD, da disciplina correspondente. Proibida a reprodução e o compartilhamento digital, sob as penas da Lei. © Editora Senac São Paulo.

Entre as principais linguagens de processamento paralelo, destacam-se: C, NESL e C++.

Já entre os paradigmas declarativos, de acordo com a figura 1, temos:

- Paradigma lógico: os programas são escritos como uma série de fatos e regras que seguem uma estrutura lógica. Por exemplo, podemos ter dois fatos: (fato 1) *Sócrates é homem*; (fato 2) *todo homem é mortal*. Se fizermos então uma pergunta do tipo: "Sócrates é mortal?", o sistema nos retornará "sim", uma vez que Sócrates é homem e todo homem é mortal. Entre as principais linguagens do paradigma lógico, destacam-se: Prolog, Absys, Alice e Ciao.
- Paradigma funcional: é baseado na execução de uma série de funções matemáticas, as quais formam um bloco de construção para a execução de diversos tipos de tarefas. Essas linguagens evitam estruturas de controle de fluxo, como loops. Entre as principais linguagens do paradigma lógico, destacam-se: Haskell e Scala.
- Paradigma database: é baseado em dados, logo, as instruções são definidas para manipular os dados em um banco, em vez de codificar uma sequência de instruções para o computador. A maioria dos bancos de dados utiliza a linguagem SQL (Structured Query Language) para manipular os dados (leitura e escrita).

2 Variáveis e constantes

As variáveis são estruturas para armazenamento de valores na memória do computador ou dispositivo (FORBELLONE; EBERSPACHER, 2005). Como seu nome indica, o valor pode ser alterado, além, é claro, de poder ser utilizado várias vezes durante um código.

A sintaxe para declaração de uma variável em C# é definida por seu tipo e nome da variável, sucedido por um ponto e vírgula para delimitar

Material para uso exclusivo de aluno matriculado em curso de Educação a Distância da Rede Senac EAD, da disciplina correspondente. Proibida a reprodução e o compartilhamento digital, sob as penas da Lei. © Editora Senac São Paulo.

o fim da instrução de declaração, por exemplo: *int saldo;* em que *int* é o tipo da variável, e *saldo* é o seu nome.

Existem muitos tipos de variáveis em C#, a saber:

- Variável local – uma variável declarada dentro de uma função/método. Ela será utilizada apenas dentro da função/método em que foi declarada.
- Variável global – uma variável declarada fora de uma função/método. Ela pode ser utilizada em todo o programa.
- Variável estática – uma variável utilizada para reter seu valor entre várias chamadas de função. Ela é declarada usando a palavra-chave *static*.

Existe também um tipo especial de variável, cujo valor não é alterado durante a execução de um programa, logo, o seu valor é fixo, a ela damos o nome de constantes. Em C#, uma constante pode ser definida em duas categorias: primárias (para tipos simples, como números e cadeia de caracteres) e secundárias (para tipos não primitivos, como array e outras estruturas de dados).

2.1 Tipos de dados

A linguagem C# é fortemente tipada, isso quer dizer que é necessário declarar o tipo de dado que uma variável ou constante armazenará. A tabela 1 a seguir apresenta os principais tipos de dados predefinidos em C#.

Tabela 1 – Tipos de dados em C#

TIPO	DESCRIÇÃO	EXEMPLO
int	Número inteiro de 32 bits	int idade = 48; int velocidade = 96;
long	Número inteiro de 64 bits	long valorx; valorx = 45L;

(*cont.*)

Material para uso exclusivo de aluno matriculado em curso de Educação a Distância da Rede Senac EAD, da disciplina correspondente. Proibida a reprodução e o compartilhamento digital, sob as penas da Lei. © Editora Senac São Paulo.

TIPO	DESCRIÇÃO	EXEMPLO
float	Número ponto flutuante de 32 bits	float altura; altura = 1.98F;
double	Número ponto flutuante de 64 bits	double peso; peso = 0.85424;
string	Cadeia de caracteres	string nome; nome = "Ana Júlia";
bool	Valor lógico (true/false) de 8 bits	bool tentativa = false;

Observe na tabela 1 que alguns tipos como *long* e *float* devem possuir um sufixo, respectivamente *L* e *F*. Caso esses sufixos não sejam colocados, o código resultará em erro.

Observe também que existem dois tipos de números (FORBELLONE; EBERSPACHER, 2005; FEIJÓ; CLUA; SILVA, 2009): os inteiros que não possuem casas decimais e os que podem ser divididos em dois tipos diferentes:

- *int*: indica que o tipo de dados é um inteiro de 32 bits e que pode armazenar números de –2.147.483.648 a 2.147.483.647. Este tipo também pode ser usado para números hexadecimais e binários;
- *long*: indica que o tipo de dados é um inteiro de 64 bits e que pode armazenar números de –9.223.372.036.854.775.808 a 9.223.372.036.854.775.807.

Os números do tipo ponto flutuante são aqueles que possuem casas decimais (FORBELLONE; EBERSPACHER, 2005; FEIJÓ; CLUA; SILVA, 2009). Observe que o ponto flutuante pode ser de dois tipos:

- *float*: pode armazenar números fracionários de 3.4e − 038 a 3.4e + 038. Ocupa 4 bytes na memória e é utilizado quando o número fracionário não necessita de muita precisão;

Material para uso exclusivo de aluno matriculado em curso de Educação a Distância da Rede Senac EAD, da disciplina correspondente. Proibida a reprodução e o compartilhamento digital, sob as penas da Lei. © Editora Senac São Paulo.

Material para uso exclusivo de aluno matriculado em curso de Educação a Distância da Rede Senac EAD, da disciplina correspondente. Proibida a reprodução e o compartilhamento digital, sob as penas da Lei. © Editora Senac São Paulo.

- *double*: pode armazenar números fracionários de 1,7e − 308 a 1,7e + 308. Ocupa 8 bytes na memória e é utilizado quando o número fracionário necessita de muita precisão.

NA PRÁTICA

As variáveis do tipo numérica (sejam elas *int*, *float*, *double* ou *long*) são utilizadas principalmente para realizar cálculos. Em C#, assim como em outras linguagens, os operadores aritméticos são: + para somar, – para subtrair, / para dividir e * para multiplicar. Além deles, destaca-se o operador % (módulo), que é responsável por retornar o resto de uma divisão. Experimente!

3 Entrada e saída de dados

Na linguagem C#, bem como em outras linguagens de programação, é possível realizar entradas e saídas de dados, o que permite que um usuário interaja com o programa. Essencialmente, para que o programa apresente alguma informação na tela no formato de texto, é utilizado o método *WriteLine* ou *Write* da classe *console* do C# , observe o exemplo de código a seguir:

```
1  Console.Write("Vou escrever");
2  Console.Write("na mesma linha.")
3  Console.WriteLine("Agora eu vou escrever")
4  Console.WriteLine("em uma outra linha")
```

A principal diferença entre *WriteLine()* e *Write()* é que o método *Write()* apresenta na tela apenas a string fornecida a ele, enquanto o método *WriteLine()* apresenta a string e também se move para o início da próxima linha.

Também é possível imprimir na tela o valor de variáveis. Observe o código a seguir:

```
1  int idade = 26;
2  Console.WriteLine(idade);
3
4  Console.WriteLine("A idade de José é " + idade);
5  Console.WriteLine("A idade de José é {0} ", idade);
```

Vamos analisar linha a linha esse código apresentado:

- Na linha 1, foi declarada a variável do tipo inteira chamada *idade*, a qual recebe o valor 26.
- Na linha 2, o programa apresentará na tela o valor 26.
- Na linha 4, o operador aritmético de soma (+) é responsável por realizar a concatenação de uma string com a variável *idade*.
- Na linha 5, ocorre a concatenação da string e da variável *valor*, porém, agora utilizando uma string formatada. Note que, quando trabalhamos com string formatada, devemos colocar marcadores de posição para a variável. Observe que *{0}* é o espaço reservado na string para a variável *idade*.

A concatenação usando formatação de string pode considerar mais de uma variável, observe o exemplo a seguir em que temos três variáveis e o objetivo é somar dois números e apresentar o resultado desta operação.

```
1  int numA, numB, resultado;
2
3  numA = 8;
```

(*cont.*)

Material para uso exclusivo de aluno matriculado em curso de Educação a Distância da Rede Senac EAD, da disciplina correspondente. Proibida a reprodução e o compartilhamento digital, sob as penas da Lei. © Editora Senac São Paulo.

Material para uso exclusivo de aluno matriculado em curso de Educação a Distância da Rede Senac EAD, da disciplina correspondente. Proibida a reprodução e o compartilhamento digital, sob as penas da Lei. © Editora Senac São Paulo.

```
4  numB = 11;
5  resultado = numA + numB;
6
7  Console.WriteLine("{0} + {1} = {2}", numA, numB, resultado);
```

Ao executar o programa mencionado, a saída será *8 + 11 = 19*.

Neste código, o marcador *{0}* é substituído pelo valor da variável *numA*, *{1}* é substituído pelo valor da variável *numB*, e o valor de *{2}* é substituído pelo valor da variável *resultado*. Observe que, na linha 5, foi realizada a operação de adição, somando os valores das variáveis *numA* e *numB* e, em seguida, armazenando na variável *resultado*.

Realizar a concatenação da linha 7 é mais simples e legível, além de evitar possíveis erros que poderiam surgir se utilizássemos o operador *+* para concatenar.

3.1 Entrada de dados

Se por um lado o método *Write* ou *WriteLine* apresenta uma saída ao usuário, é possível utilizar o método *Read*, *ReadLine* ou *ReadKey* para obter uma entrada do usuário:

- *ReadLine*: lê a próxima linha do fluxo de entrada-padrão do sistema.
- *Read*: lê um caractere do fluxo de entrada-padrão do sistema.
- *ReadKey*: lê a tecla pressionada pelo usuário. É um método usado para segurar a tela até que o usuário digite uma tecla qualquer.

O código a seguir apresenta um exemplo utilizando o *ReadLine* e também o *ReadKey*.

```
1  string nome;
2  Console.WriteLine("Escreva seu nome");
3  nome = Console.ReadLine();
4  Console.WriteLine("Olá {0}", nome);
5  Console.ReadKey();
```

Neste código, temos:

- Na linha 1, é declarada a variável do tipo string chamada *nome*.
- Na linha 2, é apresentada na tela uma mensagem para o usuário digitar o nome.
- Na linha 3, o programa é responsável por capturar o conteúdo digitado pelo usuário e armazenar na variável *nome*.
- Na linha 4, o programa saúda o usuário, concatenando a string *Olá* com o valor digitado anteriormente e que está armazenado na variável *nome*.
- Na linha 5, temos o comando *ReadKey*, que segura a tela do prompt para que ela só seja fechada quando o usuário digitar uma tecla.

NA PRÁTICA

Todo valor recebido pelo *ReadLine()* é uma cadeia de caracteres, ou seja, é do tipo string. Logo, se você receber dois valores numéricos, o resultado será um erro. Desse modo, se você solicitar um valor do tipo inteiro para o usuário, é necessário que primeiramente você converta a entrada do *ReadLine()* para *int*, utilizando o método *ToInt32*. Um exemplo de código é: *int valor = Convert.ToInt32(Console.ReadLine())*. O método *Convert.ToInt32* converterá o conteúdo recebido pelo *Console.ReadLine()* para inteiro e armazenará na variável *valor*. Experimente!

Material para uso exclusivo de aluno matriculado em curso de Educação a Distância da Rede Senac EAD, da disciplina correspondente. Proibida a reprodução e o compartilhamento digital, sob as penas da Lei. © Editora Senac São Paulo.

Material para uso exclusivo de aluno matriculado em curso de Educação a Distância da Rede Senac EAD, da disciplina correspondente. Proibida a reprodução e o compartilhamento digital, sob as penas da Lei. © Editora Senac São Paulo.

Considerações finais

Neste capítulo, pudemos compreender o conceito de paradigma de linguagem de programação. Inicialmente, conhecemos os dois principais paradigmas: o declarativo e o imperativo. Em seguida, vimos o conceito relacionado a cada um dos paradigmas de linguagem de programação, bem como as linguagens a eles associadas.

Também pudemos compreender que uma variável em linguagem de programação em C# pode possuir diversos tipos, e as numéricas podem armazenar dados inteiros (*int* ou *long*) e pontos flutuantes (*float* e *double*), além dos tipos relacionados à cadeia de caracteres, como o caso de uma string.

Por fim, conhecemos os principais métodos para realizar a entrada e saída de dados em um programa C#. Nesse momento, é fortemente recomendável que se pratique os códigos apresentados, afinal, somente com a prática é possível alcançar a proficiência na programação.

Referências

FEIJÓ, Bruno; CLUA, Esteban; SILVA, Flávio S. C. da. **Introdução à ciência da computação com jogos**: aprendendo a programar com entretenimento. Rio de Janeiro: Campus, 2009.

FORBELLONE, André Luiz Villar; EBERSPACHER, Henri Frederico. **Lógica de programação**: a construção de algoritmos e estruturas de dados. 3. ed. São Paulo: Pearson, 2005.