algoritmos e programação

com exemplos em Pascal e C

nina edelweiss
maria aparecida castro livi

bookman

as autoras

Nina Edelweiss é engenheira eletricista e doutora em Ciência da Computação pela Universidade Federal do Rio Grande do Sul. Durante muitos anos, lecionou em cursos de Engenharia e de Ciência da Computação na UFRGS, na UFSC e na PUCRS. Foi, ainda, orientadora do Programa de Pós-Graduação em Ciência da Computação da UFRGS. É coautora de três livros, tendo publicado diversos artigos em periódicos e em anais de congressos nacionais e internacionais. Participou de diversos projetos de pesquisa financiados por agências de fomento como CNPq e FAPERGS, desenvolvendo pesquisas nas áreas de bancos de dados e desenvolvimento de software.

Maria Aparecida Castro Livi é licenciada e bacharel em Letras, e mestre em Ciência da Computação pela Universidade Federal do Rio Grande do Sul. Desenvolveu sua carreira profissional na UFRGS, onde foi programadora e analista de sistema, antes de ingressar na carreira docente. Ministrou por vários anos a disciplina de Algoritmos e Programação para alunos dos cursos de Engenharia da Computação e Ciência da Computação. Sua área de interesse prioritário é o ensino de Linguagens de Programação, tanto de forma presencial quanto a distância.

E22a Edelweiss, Nina.
Algoritmos e programação com exemplos em Pascal e C [recurso eletrônico] / Nina Edelweiss, Maria Aparecida Castro Livi. – Dados eletrônicos. – Porto Alegre : Bookman, 2014.

Editado também como livro impresso em 2014.
ISBN 978-85-8260-190-7

1. Informática. 2. Algoritmos – Programação. I. Livi, Maria Aparecida Castro. II. Título.

CDU 004.421

Catalogação na publicação: Ana Paula M. Magnus – CRB 10/2052

2.3.1 expressões aritméticas

Expressões aritméticas são expressões cujos resultados são valores numéricos, inteiros ou fracionários. A sintaxe de uma expressão aritmética é a seguinte:

```
<operando>  <operador aritmético>  <operando>
```

Na pseudolinguagem utilizada neste livro, os operadores que podem ser usados em expressões aritméticas são os mesmos utilizados nas expressões aritméticas comuns. Mas, da mesma forma que nas linguagens de programação, o símbolo utilizado para a multiplicação é o asterisco, e o símbolo de divisão é a barra inclinada. A Tabela 2.1 mostra os operadores que podem ser utilizados em expressões aritméticas, na forma adotada pela pseudolinguagem.

tabela 2.1 Operadores aritméticos na pseudolinguagem

Operador	Significado	Observação
+	Soma	-
-	Subtração	-
*	Multiplicação	-
/	Divisão	-
**	Potência	-
Div	Divisão inteira	Operandos inteiros
Mod	Resto da divisão inteira	Operandos inteiros

Os operadores aritméticos têm diferentes precedências na execução das operações: primeiro são calculadas as potências, depois as multiplicações e as divisões e, no final, as somas e as subtrações. Expressões com operadores de mesma precedência justapostos são avaliadas da esquerda para a direita. Essa ordem de precedência pode ser alterada através do uso de parênteses.

Os seguintes tipos de operandos podem ser utilizados:

1. valores numéricos literais;
2. variáveis numéricas;
3. chamadas a funções[1] que devolvem um valor numérico;
4. expressões aritméticas, as quais podem incluir partes entre parênteses.

Se uma expressão aritmética incluir funções, essas terão precedência maior na execução.

Exemplos de expressões aritméticas:

```
a + 1
a * 2 + 7,32
( x / 2 )  / C - ( valor + 1 / 2 )
2 + cos(x)                                  onde cos(x) é uma função
```

As expressões aritméticas devem ser escritas horizontalmente, em uma mesma linha, com eventuais valores fracionários expressos linearmente. Muitas vezes é necessário o emprego de parênteses para garantir a execução na ordem correta. A necessidade de linearização possibilita a uma expressão aritmética ter sua aparência inicial bastante modificada, como no caso da expressão a seguir:

$$a + \frac{(b-4)(\frac{a}{2}+z42)}{c+d}$$

A representação dessa expressão em pseudolinguagem fica:

```
a + ( ( b - 4 ) * ( a / 2  + 4 * z42) / ( c + d ) )
```

[1] Uma FUNÇÃO é um subprograma. Pode receber parâmetros (valores) para realizar sua tarefa e normalmente devolve um valor em seu nome, sendo o tipo do valor devolvido o próprio tipo da função. Mais detalhes sobre definição de funções são vistos no Capítulo 9.

Algumas funções básicas predefinidas já vêm embutidas nas linguagens de programação. Entre elas, funções matemáticas, como o cálculo do cosseno de um ângulo utilizado no exemplo anterior. Algumas dessas funções necessitam de alguma informação para calcular o que é pedido como, por exemplo, o valor do ângulo do qual se quer o cosseno. As informações requeridas são chamadas de parâmetros da função e são listadas logo após o nome da função, entre parênteses. Um parâmetro pode ser fornecido através de uma expressão cujo valor, depois de avaliado, será utilizado pela função.

Na Tabela 2.2 são listadas algumas funções que podem ser utilizadas na pseudolinguagem, definidas de forma idêntica ou similar àquela em que ocorrem na maioria das linguagens de programação.

tabela 2.2 Funções predefinidas na pseudolinguagem

Nome da função	Parâmetro	Significado
abs	valor	Valor absoluto do valor
sen	ângulo	Seno do ângulo
cos	ângulo	Cosseno do ângulo
tan	ângulo	Tangente do ângulo
arctan	valor	Arco cuja tangente tem o valor
sqrt	valor	Raiz quadrada do valor
sqr	valor	Quadrado do valor
pot	base, expoente	Base elevada ao expoente
ln	valor	Logaritmo neperiano
log	valor	Logaritmo na base 10

3.1 esquema básico dos algoritmos sequenciais

Os problemas puramente sequenciais geralmente incluem três atividades, que ocorrem normalmente na ordem indicada a seguir: entrada de dados, processamento realizado sobre esses dados (cálculos, comparações) e saída de dados ou apresentação dos resultados. Mesmo em problemas mais complexos, essas atividades constituem o esquema básico subjacente. O processamento e a saída de dados são elementos sempre presentes; já a entrada pode eventualmente não ocorrer, como nos casos em que o processamento se baseia em valores predefinidos e constantes.

Entradas são os dados fornecidos pelo usuário durante a execução do programa, sem os quais não é possível solucionar o problema. Existem casos em que alguns dados de entrada, pela sua constância e regularidade, podem ser utilizados como constantes em uma solução. No Exercício de Fixação 3.3 (discutido na Seção 3.6), em que um valor em reais é convertido para dólares, em um período de estabilidade da moeda americana a taxa de conversão para o dólar poderia ser colocada como um valor constante na expressão de cálculo. Mas caso fosse preciso alterar essa taxa, seria necessário alterar o código. Nesse, e em casos semelhantes, sugere-se optar pela solução mais genérica, em que mesmo os dados relativamente estáveis são sempre fornecidos como entradas.

As saídas de um problema são geralmente os elementos mais facilmente determináveis, uma vez que correspondem aos resultados esperados.

No problema apresentado no Algoritmo 1.1 (no Capítulo 1, Seção 1.1.4), em que é calculada a soma de dois valores, os dois valores a serem somados são as entradas que devem ser informadas; o processamento é o cálculo da soma; e a saída é a soma calculada.

Na sequência veremos os comandos que permitirão a leitura dos dados de entrada, a produção de resultados e sua apresentação em um meio externo.

3.2 → comandos de entrada e de saída

Os comandos de entrada e de saída de dados fazem a ligação entre o programa e o usuário. Toda a comunicação entre o mundo virtual e o mundo real é feita através desses comandos, sem os quais o usuário não ficaria ciente do que ocorre durante e ao término do processamento.

3.2.1 comando de entrada de dados

Através de um **comando de entrada de dados**, o programador solicita que um ou mais dados sejam obtidos (lidos) pelo computador a partir de um dispositivo de entrada como, por exemplo, o teclado. Os valores lidos devem ser armazenados em variáveis na memória para que essas possam depois ser utilizadas pelo programa. Para isso, o comando de entrada de dados deve, além de solicitar a operação de leitura, informar os nomes das variáveis que irão armazenar os valores lidos.

Na pseudolinguagem, um comando de entrada de dados é identificado pela palavra reservada `ler`, seguida da lista de variáveis que irão armazenar os valores lidos, as quais aparecem separadas por vírgulas e entre parênteses:

```
ler ( <lista de variáveis separadas por vírgulas> )
```

Por exemplo, o comando:

```
ler (nota)
```

pede que seja lido um valor no dispositivo de entrada de dados, dizendo ainda que o valor lido deve ser armazenado na variável denominada `nota`. Outro exemplo é o comando:

```
ler (a, b, c)
```

onde `a`, `b` e `c` são nomes de variáveis. Através desse comando serão lidos três valores de entrada, sendo o primeiro colocado na variável `a`, o segundo na `b` e o terceiro na `c`.

3.2.2 comando de saída de dados

Comandos de saída de dados são usados para transferir para fora do computador os resultados que foram solicitados a fim de que esses sejam vistos pelo usuário ou utilizados em futuro processamento. Um comando de saída inicia sempre pela palavra reservada `escrever`, seguida da lista de valores que deverão ser informados, os quais aparecem separados por vírgulas e entre parênteses:

```
escrever ( <lista de valores de saída separados por vírgulas> )
```

A lista de valores de saída pode conter:

- nomes das variáveis cujos conteúdos devem ser informados;
- expressões que serão avaliadas, sendo seu resultado informado na saída;
- *strings* formadas por cadeias de caracteres entre apóstrofos simples. As *strings* não são analisadas pelo computador, sendo simplesmente copiadas para o dispositivo de saída. Servem para explicar ao usuário o significado dos valores que são listados.

Por exemplo, o comando:

```
escrever (a, b)
```

vai transferir para a saída o valor contido na variável `a`, seguido do valor da variável `b`. Já o comando:

```
escrever (a * b + 1 / sin (c))
```

vai avaliar a expressão fornecida (`a * b + 1 / sin (c)`), transferindo para a saída somente seu resultado. Como exemplo da utilização de *strings* na saída, considere o comando:

```
escrever ('Maior valor: ', maior, ' Resultado: ' , soma)
```

Neste comando, `maior` e `soma` são nomes de variáveis. Supondo que o valor contido na variável `maior` seja 15 e que o valor de `soma` seja 20, a saída produzida seria a seguinte:

```
Maior valor: 15 Resultado: 20
```

3.2.3 formatação de entrada e saída

As linguagens de programação possibilitam definir como os dados deverão ser apresentados, ou seja, como as informações deverão ser fornecidas na entrada de dados e como os resultados deverão ser dispostos na saída. Ao escrever um algoritmo, deve-se analisar somente quais as ações que devem integrá-lo para que represente uma solução adequada e correta para o problema que se pretende solucionar. Não é aconselhável definir formatos de entrada e saída neste momento, deixando para fazê-lo na tradução do algoritmo para uma linguagem de programação específica. Por isso, a pseudolinguagem utilizada ao longo deste livro não define a formatação de entrada e de saída de dados, deixando para mostrá-la nas seções específicas das linguagens Pascal e C.

3.3 → comando de atribuição

No **comando de atribuição**, o resultado de uma expressão é atribuído a uma variável, ou seja, é colocado no espaço de memória reservado para essa variável. Se já existia algum valor armazenado na variável, ele é substituído pelo novo valor, e o valor anterior é perdido. Na pseudolinguagem, um comando de atribuição tem à esquerda o nome da variável que vai receber o valor, seguido de uma flecha direcionada para a esquerda, seguida à direita pela expressão cujo valor vai ser utilizado na atribuição. O sentido da flecha representa visualmente que o resultado da expressão é colocado na variável:

```
<variável> ← <expressão>
```

Somente um nome de variável pode ser colocado à esquerda em um comando de atribuição. A execução do comando inicia avaliando a expressão à direita, colocando depois seu resultado na variável à esquerda. Por exemplo, no comando:

```
a ← b + 1
```

o resultado da expressão `b + 1` é atribuído à variável `a`.

O tipo da variável que vai receber a atribuição deve ser compatível com o resultado da expressão à direita. Dependendo do tipo dessa variável, três tipos de atribuição são identificados, os quais são analisados a seguir.

3.3.1 atribuição numérica

Se a variável for numérica, o valor da expressão deve ser também um valor numérico. O valor a ser atribuído à variável pode ser (1) informado diretamente através de um número, (2) o valor armazenado em outra variável numérica ou (3) o resultado de uma expressão aritmética. Pode ser utilizada qualquer expressão aritmética, incluindo nomes de variáveis e chamadas a funções. O nome da variável à qual vai ser atribuído o valor pode também ser utilizado na expressão aritmética – nesse caso, a expressão é avaliada com o valor que a variável continha antes da execução do comando, sendo esse valor perdido ao ser feita a atribuição do novo valor.

O tipo do valor a ser atribuído à variável deve ser compatível com o tipo da variável que vai receber a atribuição. Assim, variáveis inteiras devem receber valores inteiros, e variáveis reais devem receber valores reais. Uma exceção a essa regra, entretanto, é a atribuição de valores inteiros a variáveis reais.

Vamos supor que foram declaradas as seguintes variáveis:

```
i, k (inteiro)
a, r (real)
```

Utilizando essas variáveis, os comandos de atribuição a seguir são válidos:

```
i ← 10            { i RECEBE O VALOR 10 }
i ← k             { i RECEBE O VALOR DA VARIÁVEL k }
i ← k - 2         { i RECEBE VALOR DA EXPRESSÃO ARITMÉTICA k - 2}
i ← i + k - 7     { VARIÁVEL PODE SER UTILIZADA NA EXPRESSÃO }
a ← 10            { VARIÁVEL REAL PODE RECEBER VALOR INTEIRO }
a ← 3,14 * sqr(r) { EXPRESSÃO π.r² UTILIZANDO CHAMADA À FUNÇÃO }
```

Os comandos de atribuição a seguir são inválidos:

```
i ← a             { VARIÁVEL INTEIRA NÃO PODE RECEBER VALOR REAL}
i ← i + a         { EXPRESSÃO TEM RESULTADO REAL POIS a É REAL }
i ← a > r         { EXPRESSÃO NÃO PODE SER LÓGICA }
a + b ← a         { NO LADO ESQUERDO NÃO PODE TER EXPRESSÃO }
a, b ← 0          { SOMENTE UMA VARIÁVEL NO LADO ESQUERDO }
```

Uma construção bastante comum é a variável que vai receber o valor da atribuição ser também utilizada na expressão à direita do sinal de atribuição. Nesses casos, a expressão é avaliada utilizando o valor contido na variável e, terminada a avaliação da expressão, seu resultado é colocado na variável, alterando então seu valor. Por exemplo, supondo que a variável `a` é inteira e contém o valor 5, o valor contido em `a`, após a execução do comando de atribuição a seguir, é 8:

```
a ← a + 3
```

3.3.2 atribuição lógica

Se a variável à esquerda do comando de atribuição for lógica, ela poderá receber somente um dos dois valores lógicos: `verdadeiro` ou `falso`. O valor lógico a ser atribuído pode (1) ser representado explicitamente através de uma dessas duas palavras reservadas, (2) ser o conteúdo de outra variável lógica ou (3) resultar da avaliação de uma expressão lógica.

Considerar as seguintes declarações:

```
i, k (inteiro)
x, y (lógico)
```

Os exemplos a seguir são comandos de atribuição lógica válidos:

```
x ← verdadeiro    { VALOR LÓGICO INFORMADO EXPLICITAMENTE }
x ← y             { x RECEBE VALOR LÓGICO DE y }
x ← i = k         { x verdadeiro SE i IGUAL A k  }
x ← i > 7 ou y    { x RECEBE RESULTADO DA EXPRESSÃO LÓGICA }
```

Os comandos a seguir são inválidos:

```
x ← i             { i É INTEIRO, NÃO É VALOR LÓGICO }
x ← x > 7         { ERRO NA EXPRESSÃO LÓGICA }
x ← k + 1         { EXPRESSÃO TEM VALOR INTEIRO, NÃO LÓGICO }
```

3.3.3 atribuição de caracteres

Se a variável for do tipo caractere, a expressão à direita deve resultar em um caractere; se a variável for do tipo `string`, a expressão deve resultar em uma *string*.

Considerar que foram declaradas as seguintes variáveis:

```
nome (string)
letra, letra2 (caractere)
```

Utilizando essas variáveis, os exemplos a seguir mostram comandos de atribuição válidos:

```
nome ← 'Ana Terra'  { STRING ATRIBUÍDA À VARIÁVEL nome }
letra ← 'Z'         { letra RECEBE CARACTERE 'Z' }
letra ← letra2      { letra RECEBE CARACTERE DE letra2 }
```

Ainda utilizando as mesmas variáveis, os comandos a seguir são inválidos:

```
nome ← 10           { STRING NÃO PODE RECEBER VALOR NUMÉRICO }
letra ← i > 2       { CARACTERE NÃO PODE RECEBER VALOR LÓGICO }
letra ← nome        { VARIÁVEL CARACTERE NÃO PODE RECEBER STRING }
letra ← letra + 10  { EXPRESSÃO À DIREITA ESTÁ INCORRETA}
```

Encerra aqui o trecho do livro disponibilizado para esta Unidade de Aprendizagem. Na Biblioteca Virtual da Instituição, você encontra a obra na íntegra.