

Notas de Aula de Algoritmos e Programação de Computadores

FLÁVIO KEIDI MIYAZAWA

com a colaboração de

TOMASZ KOWALTOWSKI

Instituto de Computação - UNICAMP

Versão 2000.1

Estas notas de aula não devem ser usadas como única fonte de estudo. O aluno deve ler outros livros disponíveis na literatura.

Nenhuma parte destas notas pode ser reproduzida, qualquer que seja a forma ou o meio, sem a permissão dos autores.

Os autores concedem a permissão explícita para a utilização e reprodução deste material no contexto do ensino de disciplinas regulares dos cursos de graduação sob a responsabilidade do Instituto de Computação da UNICAMP.

© Copyright 2000

Instituto de Computação
UNICAMP
Caixa Postal 6176
13083-970 Campinas-SP
{fkm,tomasz}@ic.unicamp.br

5 Desenvolvendo Programas

5.1 Programação por Etapas

Quando um programa começa a ficar maior ou mais complicado, é comum se fazer primeiro um esboço, contendo operações em alto nível, mesmo que sejam complexas. Nos passos seguintes este esboço é detalhado em operações mais específicas até que se chegue na codificação do programa na linguagem de programação.

Exemplo 5.1 *Faça um programa que lê três números a , b e c e verifica se estes números formam os lados de um triângulo. Em caso afirmativo, o programa verifica se o triângulo formado é equilátero (três lados iguais), isóceles (dois lados iguais) ou escaleno (três lados diferentes).*

Vamos desenvolver este programa por etapas. Podemos considerar os seguintes passos principais:

Passo 1: *Leia a , b e c .*

Passo 2: *Se (a, b, c) não formam um triângulo então*

Passo 3: *Escreva: Os valores lidos não formam um triângulo.*

Passo 4: *Senão se (a, b, c) formam um triângulo equilátero então*

Passo 5: *Escreva: Os valores lidos formam um triângulo equilátero.*

Passo 6: *Senão se (a, b, c) formam um triângulo isóceles então*

Passo 7: *Escreva: Os valores lidos formam um triângulo isóceles.*

Passo 8: *Senão*

Passo 9: *Escreva: Os valores lidos formam um triângulo escaleno.*

Esta estrutura descreve uma estratégia para resolver o problema, sem se preocupar com cada um dos testes mais complicados (condições que estão destacadas). A tradução dos comandos não destacados para a linguagem Pascal é praticamente direta. Assim, vamos nos preocupar com cada um dos testes destacados.

1. No passo 2, temos de verificar quando três números não formam os lados de um triângulo. Isto pode ser feito verificando se um dos valores é maior que a soma dos outros dois valores. Assim, podemos trocar a condição do passo 2 por:

$$(a > b + c) \text{ ou } (b > a + c) \text{ ou } (c > a + b)$$

2. No passo 4 devemos testar se os valores a , b e c formam um triângulo equilátero. Isto é verdadeiro se os três valores são iguais e pode ser feito usando duas relações de igualdade.

$$(a = b) \text{ e } (b = c)$$

3. No passo 6 devemos testar se os valores a , b e c formam um triângulo isóceles. Neste caso devemos verificar se qualquer dois dos três valores são iguais.

$$(a = b) \text{ ou } (a = c) \text{ ou } (b = c)$$

Substituindo estas condições no algoritmo acima e traduzindo para a Linguagem Pascal, temos:

```

program triangulo;
var a,b,c: real;
begin
  write('Entre com o primeiro lado: '); readln(a);
  write('Entre com o segundo lado: '); readln(b);
  write('Entre com o terceiro lado: '); readln(c);
  if ((a > b + c) or (b > a + c) or (c > a + b))
    then {não existe triângulo}
      writeln('Os números não formam um triângulo.')
    else
      if (a = b) and (a = c)
        then {é triângulo equilátero}
          writeln('O triângulo é equilátero')
        else
          if (a = b) or (a = c) or (b = c)
            then {é triângulo isóceles}
              writeln('O triângulo é isóceles')
            else {é triângulo escaleno}
              writeln('O triângulo é escaleno')
    end.

```

Exemplo 5.2 Programa para ler 3 números n_1 , n_2 e n_3 e imprimir (n_1, n_2, n_3) de maneira a ficarem em ordem não decrescente de valor.

Um primeiro esboço, que apresenta uma estratégia para se resolver o problema, é dada a seguir:

1. Ler os três números nas variáveis n_1, n_2, n_3 .
2. Trocar os valores das variáveis n_1, n_2, n_3 , de forma que n_3 contém um maior valor entre n_1, n_2, n_3 . Neste momento, n_3 contém um valor correto.
3. Trocar os valores das variáveis n_1, n_2 , de forma que n_2 contém um maior valor entre n_1, n_2 .
4. Imprimir (n_1, n_2, n_3) .

Em um segundo passo, podemos descrever a estratégia acima de forma mais detalhada, omitindo alguns passos ainda desconhecidos.

```

program ordena3;
var n1, n2, n3: real;
begin
  write('Entre com o primeiro número: '); readln(n1);
  write('Entre com o segundo número: '); readln(n2);
  write('Entre com o terceiro número: '); readln(n3);
  if (n3 < n1)
    then troca valores de n3 e n1
  if (n3 < n2)
    then troca valores de n3 e n2
  {Neste ponto, n3 contém um maior valor dos três valores lidos.}
  if (n2 < n1)
    then troca valores de n2 e n1
  {Neste ponto, n2 contém um segundo maior valor dos três valores lidos.}
  {Após acertar os valores em n3 e n2, observe que n1 contém um menor valor dos três lidos.}
  writeln(n1, n2, n3);
end.

```

Agora considere o problema de se trocar os valores contidos em apenas duas variáveis. Digamos que você queira trocar os valores das variáveis A e B . Não é possível passar o valor de A para B diretamente, já que isto faria com que perdêssemos o valor original de B . Da mesma forma, também não é possível passar o valor de B para A diretamente, sem perda do valor original de A . Isto nos induz a usar uma outra variável auxiliar, AUX , para primeiro guardar o valor de B , em seguida passar o valor de A para B , e depois passar o valor de AUX para A . Isto nos dá a seguinte seqüência de operações:

$$AUX \leftarrow B; \quad B \leftarrow A; \quad A \leftarrow AUX;$$

Substituindo este processo de troca dos valores em variáveis no pseudo programa acima, obtemos o seguinte programa final:

```
program ordena3;
var n1,n2,n3, AUX: real;
begin
  write('Entre com o primeiro número: '); readln(n1);
  write('Entre com o segundo número: '); readln(n2);
  write('Entre com o terceiro número: '); readln(n3);
  if (n3<n1) then begin
    AUX:= n3;   n3:= n1;   n1:= AUX;
  end;
  if (n3<n2) then begin
    AUX:= n3;   n3:= n2;   n2:= AUX;
  end;
  if (n2<n1) then begin
    AUX:= n2;   n2:= n1;   n1:= AUX;
  end;
  writeln(n1,n2,n3);
end.
```

Exercício 5.1 *Descreva um programa usando a mesma estratégia do exemplo 5.2 para ordenar quatro números nas variáveis $n1$, $n2$, $n3$ e $n4$.*

Exercício 5.2 *A estratégia do exemplo 5.2 foi a de colocar o maior dos valores lidos na variável correta (acertar o valor em $n3$) e em seguida aplicar o mesmo método para acertar as variáveis restantes (variáveis $n1$ e $n2$). Faça um programa para ordenar três números como no exemplo 5.2, mas usando a estratégia de colocar o menor dos valores lidos na variável correta (variável $n1$) e reaplicar o mesmo método para as demais variáveis (variáveis $n2$ e $n3$).*

5.2 Simulação de programas

Uma simulação de um programa nada mais é que simular o comportamento de um programa em um computador. As simulações que faremos tem por objetivo verificar se o programa está realmente correto ou encontrar o erro de um programa incorreto.

É aconselhável que se faça sempre uma simulação do programa antes de implementá-lo no computador. Isto permitirá que o programador possa tratar possíveis erros no momento em que estiver projetando o programa e saber como consertá-lo mais facilmente uma vez que as idéias e estratégias usadas no desenvolvimento do programa estarão mais “frescas” em sua memória.

Muitos compiladores que integram ambientes de editoração possibilitam a execução de um programa passo a passo e a observação de todo o processamento dos dados existentes na memória. Com isso, as simulações podem ser feitas diretamente neste software integrado. Outros compiladores podem inserir informações dentro do código executável do programa contendo as informações necessárias para se fazer uma execução passo a passo no programa fonte.

Para fazer uma simulação em papel, precisamos ter o programa e uma “memória” em papel. Nesta simulação você estará simulando o computador acompanhando o programa e processando os dados que estão na memória de papel. Isto é, você fará o trabalho da CPU. A seguir iremos descrever como fazer esta simulação.

Primeiramente, descreva uma tabela, colocando na primeira linha os nomes de todas as variáveis declaradas no programa, como apresentado abaixo. Caso seja uma função ou procedimento (veja seção 7) não esqueça de colocar também os parâmetros declarados nestas rotinas.

Variável-1	Variável-2	...	Variável-K	Parâmetro-1	...	Parâmetro-M

Em seguida coloque os valores iniciais destas variáveis e parâmetros. Caso não tenha sido atribuído nenhum valor a elas, coloque a palavra *lixo* (já que é um valor que você desconhece). No caso de parâmetros, coloque os valores que foram passados como parâmetros (alguns podem ser lixo também). Qualquer operação que utilize o valor *lixo* também resulta em *lixo*.

Variável-1	Variável-2	Variável-3	Parâmetro-1	Parâmetro-2
<i>lixo</i>	<i>lixo</i>	<i>lixo</i>	<i>valor-param1</i>	<i>lixo</i>

Vamos fazer a simulação de um programa para ler uma seqüência de números inteiros não negativos terminada com um número inteiro negativo. O programa deve imprimir a soma e quantidade dos números não negativos.

```

1. program Sequencia;
2. var x,soma,n : integer;
3. begin
4.     write('Entre com um número: ');
5.     readln(x);
6.     soma := 0;
7.     n := 0;
8.     while (x>=0) do begin
9.         soma := soma + x;
10.        n := n + 1;
11.        write('Entre com um número: ');
12.        readln(x);
13.    end;
14.    writeln('Soma=',soma,' Quantidade=',n);
15. end.

```

Marcaremos o comando que acabou de ser executado com uma marca → (inicialmente marcado no **begin** do bloco de comandos do programa principal). Vamos supor que a entrada é a seqüência: 1,2,3,-1.

<pre> 1. program Sequencia; 2. var x,soma,n : integer; → begin 4. write('Entre com um número: '); 5. readln(x); 6. soma := 0; 7. n := 0; 8. while (x>=0) do begin 9. soma := soma + x; 10. n := n + 1; 11. write('Entre com um número: '); 12. readln(x); 13. end; 14. writeln('Soma=',soma,' Quantidade=',n); 15. end.</pre>	<table border="1"> <thead> <tr> <th>x</th> <th>Soma</th> <th>n</th> </tr> </thead> <tbody> <tr> <td><i>lixo</i></td> <td><i>lixo</i></td> <td><i>lixo</i></td> </tr> <tr><td> </td><td> </td><td> </td></tr> <tr><td> </td><td> </td><td> </td></tr> <tr><td> </td><td> </td><td> </td></tr> <tr><td> </td><td> </td><td> </td></tr> <tr><td> </td><td> </td><td> </td></tr> <tr><td> </td><td> </td><td> </td></tr> <tr><td> </td><td> </td><td> </td></tr> <tr><td> </td><td> </td><td> </td></tr> <tr><td> </td><td> </td><td> </td></tr> <tr><td> </td><td> </td><td> </td></tr> <tr><td> </td><td> </td><td> </td></tr> </tbody> </table>	x	Soma	n	<i>lixo</i>	<i>lixo</i>	<i>lixo</i>																																		<p>Início da Simulação Seqüência a testar: 1,2,3,-1</p>
x	Soma	n																																							
<i>lixo</i>	<i>lixo</i>	<i>lixo</i>																																							
<pre> 1. program Sequencia; 2. var x,soma,n : integer; 3. begin 4. write('Entre com um número: '); → readln(x); 6. soma := 0; 7. n := 0; 8. while (x>=0) do begin 9. soma := soma + x; 10. n := n + 1; 11. write('Entre com um número: '); 12. readln(x); 13. end; 14. writeln('Soma=',soma,' Quantidade=',n); 15. end.</pre>	<table border="1"> <thead> <tr> <th>x</th> <th>Soma</th> <th>n</th> </tr> </thead> <tbody> <tr> <td>lixo</td> <td><i>lixo</i></td> <td><i>lixo</i></td> </tr> <tr> <td>1</td> <td> </td> <td> </td> </tr> <tr><td> </td><td> </td><td> </td></tr> <tr><td> </td><td> </td><td> </td></tr> <tr><td> </td><td> </td><td> </td></tr> <tr><td> </td><td> </td><td> </td></tr> <tr><td> </td><td> </td><td> </td></tr> <tr><td> </td><td> </td><td> </td></tr> <tr><td> </td><td> </td><td> </td></tr> <tr><td> </td><td> </td><td> </td></tr> <tr><td> </td><td> </td><td> </td></tr> <tr><td> </td><td> </td><td> </td></tr> </tbody> </table>	x	Soma	n	lixo	<i>lixo</i>	<i>lixo</i>	1																																	<p>Execução dos passos 4 e 5. Leitura do valor 1.</p>
x	Soma	n																																							
lixo	<i>lixo</i>	<i>lixo</i>																																							
1																																									
<pre> 1. program Sequencia; 2. var x,soma,n : integer; 3. begin 4. write('Entre com um número: '); 5. readln(x); 6. soma := 0; → n := 0; 8. while (x>=0) do begin 9. soma := soma + x; 10. n := n + 1; 11. write('Entre com um número: '); 12. readln(x); 13. end; 14. writeln('Soma=',soma,' Quantidade=',n); 15. end.</pre>	<table border="1"> <thead> <tr> <th>x</th> <th>Soma</th> <th>n</th> </tr> </thead> <tbody> <tr> <td>lixo</td> <td>lixo</td> <td>lixo</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> </tr> <tr><td> </td><td> </td><td> </td></tr> <tr><td> </td><td> </td><td> </td></tr> <tr><td> </td><td> </td><td> </td></tr> <tr><td> </td><td> </td><td> </td></tr> <tr><td> </td><td> </td><td> </td></tr> <tr><td> </td><td> </td><td> </td></tr> <tr><td> </td><td> </td><td> </td></tr> <tr><td> </td><td> </td><td> </td></tr> <tr><td> </td><td> </td><td> </td></tr> <tr><td> </td><td> </td><td> </td></tr> </tbody> </table>	x	Soma	n	lixo	lixo	lixo	1	0	0																															<p>Execução dos passos 6 e 7. Inicialização de <i>Soma</i> e <i>n</i>.</p>
x	Soma	n																																							
lixo	lixo	lixo																																							
1	0	0																																							
<pre> 1. program Sequencia; 2. var x,soma,n : integer; 3. begin 4. write('Entre com um número: '); 5. readln(x); 6. soma := 0; 7. n := 0; → while (x>=0) do begin 9. soma := soma + x; 10. n := n + 1; 11. write('Entre com um número: '); 12. readln(x); 13. end; 14. writeln('Soma=',soma,' Quantidade=',n); 15. end.</pre>	<table border="1"> <thead> <tr> <th>x</th> <th>Soma</th> <th>n</th> </tr> </thead> <tbody> <tr> <td>lixo</td> <td>lixo</td> <td>lixo</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> </tr> <tr><td> </td><td> </td><td> </td></tr> <tr><td> </td><td> </td><td> </td></tr> <tr><td> </td><td> </td><td> </td></tr> <tr><td> </td><td> </td><td> </td></tr> <tr><td> </td><td> </td><td> </td></tr> <tr><td> </td><td> </td><td> </td></tr> <tr><td> </td><td> </td><td> </td></tr> <tr><td> </td><td> </td><td> </td></tr> <tr><td> </td><td> </td><td> </td></tr> <tr><td> </td><td> </td><td> </td></tr> </tbody> </table>	x	Soma	n	lixo	lixo	lixo	1	0	0																															<p>Teste da condição do while. Elemento lido é ≥ 0 ? $1 \geq 0$? Sim. Entrar no <i>loop</i>.</p>
x	Soma	n																																							
lixo	lixo	lixo																																							
1	0	0																																							

<pre> 1. program Sequencia; 2. var x,soma,n : integer; 3. begin 4. write('Entre com um número: '); 5. readln(x); 6. soma := 0; 7. n := 0; 8. while (x>=0) do begin 9. soma := soma + x; → n := n + 1; 11. write('Entre com um número: '); 12. readln(x); 13. end; 14. writeln('Soma=',soma,' Quantidade=',n); 15. end.</pre>	<table border="1"> <thead> <tr><th>x</th><th>Soma</th><th>n</th></tr> </thead> <tbody> <tr><td>lixo</td><td>lixo</td><td>lixo</td></tr> <tr><td>1</td><td>0</td><td>0</td></tr> <tr><td></td><td>1</td><td>1</td></tr> <tr><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td></tr> </tbody> </table>	x	Soma	n	lixo	lixo	lixo	1	0	0		1	1																						<p>Execução dos passos 9 e 10. Acumular valor em Soma. Incrementar n.</p>
x	Soma	n																																	
lixo	lixo	lixo																																	
1	0	0																																	
	1	1																																	
<pre> 1. program Sequencia; 2. var x,soma,n : integer; 3. begin 4. write('Entre com um número: '); 5. readln(x); 6. soma := 0; 7. n := 0; 8. while (x>=0) do begin 9. soma := soma + x; 10. n := n + 1; 11. write('Entre com um número: '); → readln(x); 13. end; 14. writeln('Soma=',soma,' Quantidade=',n); 15. end.</pre>	<table border="1"> <thead> <tr><th>x</th><th>Soma</th><th>n</th></tr> </thead> <tbody> <tr><td>lixo</td><td>lixo</td><td>lixo</td></tr> <tr><td>1</td><td>0</td><td>0</td></tr> <tr><td>2</td><td>1</td><td>1</td></tr> <tr><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td></tr> </tbody> </table>	x	Soma	n	lixo	lixo	lixo	1	0	0	2	1	1																						<p>Execução dos passos 11 e 12. Leitura do valor 2.</p>
x	Soma	n																																	
lixo	lixo	lixo																																	
1	0	0																																	
2	1	1																																	
<pre> 1. program Sequencia; 2. var x,soma,n : integer; 3. begin 4. write('Entre com um número: '); 5. readln(x); 6. soma := 0; 7. n := 0; → while (x>=0) do begin 9. soma := soma + x; 10. n := n + 1; 11. write('Entre com um número: '); 12. readln(x); 13. end; 14. writeln('Soma=',soma,' Quantidade=',n); 15. end.</pre>	<table border="1"> <thead> <tr><th>x</th><th>Soma</th><th>n</th></tr> </thead> <tbody> <tr><td>lixo</td><td>lixo</td><td>lixo</td></tr> <tr><td>1</td><td>0</td><td>0</td></tr> <tr><td>2</td><td>1</td><td>1</td></tr> <tr><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td></tr> </tbody> </table>	x	Soma	n	lixo	lixo	lixo	1	0	0	2	1	1																						<p>Teste da condição do while. Elemento lido é ≥ 0? $2 \geq 0$? Sim. Entrar no <i>loop</i>.</p>
x	Soma	n																																	
lixo	lixo	lixo																																	
1	0	0																																	
2	1	1																																	
<pre> 1. program Sequencia; 2. var x,soma,n : integer; 3. begin 4. write('Entre com um número: '); 5. readln(x); 6. soma := 0; 7. n := 0; 8. while (x>=0) do begin 9. soma := soma + x; → n := n + 1; 11. write('Entre com um número: '); 12. readln(x); 13. end; 14. writeln('Soma=',soma,' Quantidade=',n); 15. end.</pre>	<table border="1"> <thead> <tr><th>x</th><th>Soma</th><th>n</th></tr> </thead> <tbody> <tr><td>lixo</td><td>lixo</td><td>lixo</td></tr> <tr><td>1</td><td>0</td><td>0</td></tr> <tr><td>2</td><td>1</td><td>1</td></tr> <tr><td></td><td>3</td><td>2</td></tr> <tr><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td></tr> </tbody> </table>	x	Soma	n	lixo	lixo	lixo	1	0	0	2	1	1		3	2																			<p>Execução dos passos 9 e 10. Acumular valor em Soma. Incrementar n.</p>
x	Soma	n																																	
lixo	lixo	lixo																																	
1	0	0																																	
2	1	1																																	
	3	2																																	
<pre> 1. program Sequencia; 2. var x,soma,n : integer; 3. begin 4. write('Entre com um número: '); 5. readln(x); 6. soma := 0; 7. n := 0; 8. while (x>=0) do begin 9. soma := soma + x; 10. n := n + 1; 11. write('Entre com um número: '); → readln(x); 13. end; 14. writeln('Soma=',soma,' Quantidade=',n); 15. end.</pre>	<table border="1"> <thead> <tr><th>x</th><th>Soma</th><th>n</th></tr> </thead> <tbody> <tr><td>lixo</td><td>lixo</td><td>lixo</td></tr> <tr><td>1</td><td>0</td><td>0</td></tr> <tr><td>2</td><td>1</td><td>1</td></tr> <tr><td>3</td><td>3</td><td>2</td></tr> <tr><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td></tr> </tbody> </table>	x	Soma	n	lixo	lixo	lixo	1	0	0	2	1	1	3	3	2																			<p>Execução dos passos 11 e 12. Leitura do valor 3.</p>
x	Soma	n																																	
lixo	lixo	lixo																																	
1	0	0																																	
2	1	1																																	
3	3	2																																	

<pre> 1. program Sequencia; 2. var x,soma,n : integer; 3. begin 4. write('Entre com um número: '); 5. readln(x); 6. soma := 0; 7. n := 0; → while (x>=0) do begin 9. soma := soma + x; 10. n := n + 1; 11. write('Entre com um número: '); 12. readln(x); 13. end; 14. writeln('Soma=',soma,' Quantidade=',n); 15. end.</pre>	<table border="1"> <thead> <tr><th>x</th><th>Soma</th><th>n</th></tr> </thead> <tbody> <tr><td>lixo</td><td>lixo</td><td>lixo</td></tr> <tr><td>1</td><td>0</td><td>0</td></tr> <tr><td>2</td><td>1</td><td>1</td></tr> <tr><td>3</td><td>3</td><td>2</td></tr> <tr><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td></tr> </tbody> </table>	x	Soma	n	lixo	lixo	lixo	1	0	0	2	1	1	3	3	2																			<p>Teste da condição do while. Elemento lido é ≥ 0 ? $3 \geq 0$? Sim. Entrar no <i>loop</i>.</p>
x	Soma	n																																	
lixo	lixo	lixo																																	
1	0	0																																	
2	1	1																																	
3	3	2																																	
<pre> 1. program Sequencia; 2. var x,soma,n : integer; 3. begin 4. write('Entre com um número: '); 5. readln(x); 6. soma := 0; 7. n := 0; 8. while (x>=0) do begin 9. soma := soma + x; → n := n + 1; 11. write('Entre com um número: '); 12. readln(x); 13. end; 14. writeln('Soma=',soma,' Quantidade=',n); 15. end.</pre>	<table border="1"> <thead> <tr><th>x</th><th>Soma</th><th>n</th></tr> </thead> <tbody> <tr><td>lixo</td><td>lixo</td><td>lixo</td></tr> <tr><td>1</td><td>0</td><td>0</td></tr> <tr><td>2</td><td>1</td><td>1</td></tr> <tr><td>3</td><td>3</td><td>2</td></tr> <tr><td></td><td>6</td><td>3</td></tr> <tr><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td></tr> </tbody> </table>	x	Soma	n	lixo	lixo	lixo	1	0	0	2	1	1	3	3	2		6	3																<p>Execução dos passos 9 e 10. Acumular valor em Soma. Incrementar <i>n</i>.</p>
x	Soma	n																																	
lixo	lixo	lixo																																	
1	0	0																																	
2	1	1																																	
3	3	2																																	
	6	3																																	
<pre> 1. program Sequencia; 2. var x,soma,n : integer; 3. begin 4. write('Entre com um número: '); 5. readln(x); 6. soma := 0; 7. n := 0; 8. while (x>=0) do begin 9. soma := soma + x; 10. n := n + 1; 11. write('Entre com um número: '); → readln(x); 13. end; 14. writeln('Soma=',soma,' Quantidade=',n); 15. end.</pre>	<table border="1"> <thead> <tr><th>x</th><th>Soma</th><th>n</th></tr> </thead> <tbody> <tr><td>lixo</td><td>lixo</td><td>lixo</td></tr> <tr><td>1</td><td>0</td><td>0</td></tr> <tr><td>2</td><td>1</td><td>1</td></tr> <tr><td>3</td><td>3</td><td>2</td></tr> <tr><td>-1</td><td>6</td><td>3</td></tr> <tr><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td></tr> </tbody> </table>	x	Soma	n	lixo	lixo	lixo	1	0	0	2	1	1	3	3	2	-1	6	3																<p>Execução dos passos 11 e 12. Leitura do valor -1.</p>
x	Soma	n																																	
lixo	lixo	lixo																																	
1	0	0																																	
2	1	1																																	
3	3	2																																	
-1	6	3																																	
<pre> 1. program Sequencia; 2. var x,soma,n : integer; 3. begin 4. write('Entre com um número: '); 5. readln(x); 6. soma := 0; 7. n := 0; → while (x>=0) do begin 9. soma := soma + x; 10. n := n + 1; 11. write('Entre com um número: '); 12. readln(x); 13. end; 14. writeln('Soma=',soma,' Quantidade=',n); 15. end.</pre>	<table border="1"> <thead> <tr><th>x</th><th>Soma</th><th>n</th></tr> </thead> <tbody> <tr><td>lixo</td><td>lixo</td><td>lixo</td></tr> <tr><td>1</td><td>0</td><td>0</td></tr> <tr><td>2</td><td>1</td><td>1</td></tr> <tr><td>3</td><td>3</td><td>2</td></tr> <tr><td>-1</td><td>6</td><td>3</td></tr> <tr><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td></tr> </tbody> </table>	x	Soma	n	lixo	lixo	lixo	1	0	0	2	1	1	3	3	2	-1	6	3																<p>Teste da condição do while. Elemento lido é ≥ 0 ? $-1 \geq 0$? Não. Ir para o próximo comando depois do <i>loop</i>.</p>
x	Soma	n																																	
lixo	lixo	lixo																																	
1	0	0																																	
2	1	1																																	
3	3	2																																	
-1	6	3																																	
<pre> 1. program Sequencia; 2. var x,soma,n : integer; 3. begin 4. write('Entre com um número: '); 5. readln(x); 6. soma := 0; 7. n := 0; 8. while (x>=0) do begin 9. soma := soma + x; 10. n := n + 1; 11. write('Entre com um número: '); 12. readln(x); 13. end; → writeln('Soma=',soma,' Quantidade=',n); 15. end.</pre>	<table border="1"> <thead> <tr><th>x</th><th>Soma</th><th>n</th></tr> </thead> <tbody> <tr><td>lixo</td><td>lixo</td><td>lixo</td></tr> <tr><td>1</td><td>0</td><td>0</td></tr> <tr><td>2</td><td>1</td><td>1</td></tr> <tr><td>3</td><td>3</td><td>2</td></tr> <tr><td>-1</td><td>6</td><td>3</td></tr> <tr><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td></tr> </tbody> </table>	x	Soma	n	lixo	lixo	lixo	1	0	0	2	1	1	3	3	2	-1	6	3																<p>Impressão de <i>Soma</i> e <i>n</i>. <i>Soma</i> = 6 <i>n</i> = 3</p>
x	Soma	n																																	
lixo	lixo	lixo																																	
1	0	0																																	
2	1	1																																	
3	3	2																																	
-1	6	3																																	

5.3 Alinhamento de Comandos

Nesta seção apresentamos algumas formas de alinhamento que visam a melhor visualização dos comandos (ou declarações) que estão sob atuação de outros. Estas permitem que programas fiquem melhor apresentados e mais fáceis de se entender. A idéia é fazer com que os comandos que estão sob atuação de outros estejam alinhados e deslocados alguns espaços a direita.

A seguir apresentamos algumas exemplos de alinhamento para alguns comandos. O leitor não necessariamente precisa seguir esta regra, mas é importante que se use alguma regra razoável de alinhamento dos comandos.

Obs.: As barras verticais são apenas para visualizar o alinhamento dos comandos.

- Declarações: const, type e var.

```
const
  | MAX = 101;
  | PI = 3.14158;

type
  | MeuInteiro = integer;
  | TipoNome = string[50];

var
  | a, b, c : integer;
  | x, y, z : real;
```

- Bloco de Comandos:

```
begin
  | Comando1;
  | Comando2;
  |
  | ComandoN;
end;
```

- Comando if-then, if-then-else.

```
if (Condição) then
  | Comando;
```

```
if (Condição) then begin
  | Comando1;
  |
  | ComandoN;
end;
```

if (Condição) then begin

```
  | Comando1;
  |
  | ComandoN;
end else begin
  | ComandoN+1;
  |
  | ComandoN+M;
end;
```

- Comando for

```
for i := (Expressão) to (Expressão) do
  | Comando;
```

for i := (Expressão) to (Expressão) do begin

```
  | Comando1;
  |
  | ComandoN;
end;
```

- Comando while

```
while (Condição) do
  | Comando;
```

while (Condição) do begin

```
  | Comando1;
  |
  | ComandoN;
end;
```

- Comando repeat

```
repeat
  | Comando1;
  |
  | ComandoN;
until (Condição);
```

Alinhamento de comandos encaixados

Quando um comando é colocado internamente a outro comando, o comando interno e todos sobre sua atuação são deslocados adicionalmente ao alinhamento do comando externo.

Exemplo 5.3 No exemplo a seguir apresentamos o esboço de um programa com alguns comandos encaixados em outros e o alinhamento em cada um deles.

```
program EsboçoAlinhamento;
var
  a, b, c : integer;
  x, y, z : real;
begin
  if (Condição) then begin
    Comando1;
    for i:=(Expressão) to (Expressão) do begin
      ComandoF1;
      ⋮
      ComandoFN;
    end;
    if(Condição) then begin
      ComandoI1;
      for i:=(Expressão) to (Expressão) do begin
        ComandoIF1;
        ⋮
        ComandoIFN;
      end;
      while (Expressão) do begin
        if (Condição) then begin
          ComandoIFT1;
          ⋮
          ComandoIFTN;
        end else begin
          ComandoIFEN+1;
          ⋮
          ComandoIFEN+M;
        end;
        ComandoIW1;
        ⋮
        ComandoIWN;
      end;
    end;
  end;
  repeat
    ComandoR1;
    ⋮
    ComandoRK;
  until (Condição);
end.
```

5.4 Desenvolvimento de Algoritmos Eficientes

Em geral teremos inúmeros algoritmos para se resolver o mesmo problema. Alguns deles podem ser rápidos, enquanto outros podem exigir mais processamento e conseqüentemente podem resultar em programas lentos ou até mesmo inviáveis de serem executados. Outros algoritmos podem ser rápidos mas podem consumir muita memória, deixando o programa igualmente inviável.

Ao desenvolver um algoritmo, precisamos considerar os propósitos do algoritmo juntamente com os recursos computacionais usados por ele. Dois dos principais recursos computacionais são o *tempo de processamento* e a *memória*.

Outra consideração importante é o grau de facilidade de desenvolvimento e manutenção do programa. Programas escritos de maneira simples são em geral mais fáceis de se dar manutenção do que aqueles que usam estruturas complexas e códigos otimizados.

No exemplo seguinte consideramos algumas versões de algoritmos para o problema de se verificar se um número é primo ou não.

Exemplo 5.4 *Faça um programa para verificar se um número positivo, lido pelo programa, é primo ou não. Obs.: Um número positivo maior que 1 é primo se é divisível somente por ele mesmo e pela unidade.*

```
Program Primo1;
var i,n      : integer;
    EhPrimo : Boolean;
begin
  writeln('Programa para verificar se um número positivo é primo ou não. ');
  write('Entre com o número a testar: ');
  readln(n);
  EhPrimo:= True;
  for i:=2 to n-1 do
    if (n mod i = 0) then EhPrimo := false;
  if (EhPrimo) and (n>1)
  then writeln('O número ',n,' é primo.')
  else writeln('O número ',n,' não é primo.')
end.
```

O programa acima é bem simples e compacto. É fácil ver que ele está correto, observando apenas a definição de número primo, não ser divisível por nenhum número entre $\{2, \dots, n - 1\}$, e do comando de repetição que percorre estes números testando se n é divisível por um deles. Além disso, este programa usa pouca memória, uma vez que usamos um pequeno número de variáveis. Por outro lado, se n é um número grande, a quantidade de interações é no pior casos igual a $n - 2$.

Um outro extremo é quando temos já armazenados todos os números representáveis de nosso interesse (por exemplo: os números que podem ser armazenados em um tipo inteiro de 2 bytes), dizendo se cada um é primo ou não. Note que isto já exigiu um grande pré-processamento para todos os números, mas uma vez feito isso, todas as consultas poderão ser feitas em tempo constante e portanto extremamente rápido. É fácil ver que este tipo de solução pode ser inviável devido a grande memória usada para armazenar todos estes números. Uma solução menos drástica seria armazenar todos os primos representáveis (em vez de todos os números). Mas mesmo com esta redução a quantidade de primos existentes é grande, além disso agora não podemos mais fazer o teste em tempo constante (é possível fazer em tempo proporcional a $\log(m)$, onde m é a quantidade de primos).

Outra estratégia interessante é testar primeiro para um subconjunto fixo de primos (quantidade constante). Por exemplo, vamos testar inicialmente se n é divisível por 2 (i.e., n é par); em caso positivo, n não é primo, caso contrário testamos apenas para os elementos ímpares dos números em $\{3, \dots, n - 1\}$. Isto já nos dá um algoritmo que faz, no pior caso, em torno da metade do processamento do primeiro algoritmo.

Uma estratégia interessante é estudarmos mais sobre as características do problema para se tentar otimizar o programa. Note que se um número n não é primo, então deve ser possível escrever n como $n = a \cdot b$. Sem perda de generalidade, considere $a \leq b$. Para verificar se n não é primo, basta achar um dos divisores. Então se encontramos a , não é mais necessário testar para os outros números maiores que a . Além disso, podemos usar o fato de a ser

limitado por \sqrt{n} . Este fato é importante uma vez que para valores grandes de n , \sqrt{n} é bem menor que n , o que reduz bastante a quantidade de testes feito pelo programa. O seguinte programa apresenta a implementação desta idéia.

```
Program Primo2;
var i,n      : integer;
    Max      : real;
    EhPrimo  : Boolean;
begin
  writeln('Programa para verificar se um número positivo é primo ou não. ');
  write('Entre com o número a testar: ');
  readln(n);
  if (n = 2) then EhPrimo := true
  else if (n mod 2 = 0) or (n <= 1) then EhPrimo := False
  else begin
    EhPrimo := True;
    i := 3;
    Max := sqrt(n);
    while (i <= Max) and (n mod i <> 0) do i := i + 2;
    if (i <= Max) and (n mod i = 0) then EhPrimo := false;
  end;
  if EhPrimo
  then writeln('O número ',n,' é primo.')
  else writeln('O número ',n,' não é primo. ');
end.
```

Note que se $n = 1031$ (n é primo), o primeiro programa faz 1029 interações, no comando de repetição, e o programa acima, que também faz uso de uma quantidade constante de memória, faz apenas 15 interações.

Exercício 5.3 Faça um programa que verifica se um número é primo, como no programa Primo2, mas além de pular os múltiplos de 2, também pula os múltiplos de 3.

5.5 Precisão Numérica e Erros de Precisão

A linguagem Pascal nos oferece o tipo real para trabalharmos com números. Em geral este tipo é implementado internamente com uma quantidade fixa de bytes, o que torna impossível a representação exata de todos os números. Isto é razoável uma vez que há infinitos números reais, mesmo em intervalos pequenos como $[0, 1]$.

Apesar disso, a maioria das implementações representa o número $\frac{1}{3}$ internamente com uma boa quantidade de casas decimais corretas. Por exemplo, se $\frac{1}{3}$ for representado como 0,333333333333333315 teremos o valor correto até a 16ª casa decimal. Na maioria das aplicações este erro na precisão pode ser desprezível. Um dos principais problemas com erros de precisão é o cálculo entre valores que já contêm erros de precisão, que podem gerar resultados com erros maiores. O seguinte programa mostra como o erro de precisão em apenas uma conta faz com que tenhamos um resultado totalmente inesperado.

```
program Terco;
var i      : integer;
    terco  : real;
begin
  terco := 1/3;
  for i:=1 to 30 do begin
    terco := 4 * terco - 1;
    writeln('Iteração: ',i:2,' , Terco = ',terco:20:18);
  end;
end.
```

Se não houvesse erros de precisão, a variável *Terco* começaria com o valor $\frac{1}{3}$ e em cada interação o valor de *Terco*

é atualizado como $\boxed{4 \cdot Terco - 1}$. Se *Terco* tivesse o valor $\frac{1}{3}$, o novo valor de *Terco* seria $4 \cdot \frac{1}{3} - 1$ que é igual a $\frac{1}{3}$. Isto é, em cada interação *Terco* recebe novamente o valor $\frac{1}{3}$.

No entanto, a maioria dos programas executáveis construídos a partir deste programa fonte deve imprimir, na última interação, um valor que é totalmente diferente de $\frac{1}{3}$.

Vamos analisar este programa com mais detalhes. Note que $\frac{1}{3}$ não deve ser armazenado de forma exata, sendo armazenado correto até certa casa decimal. Portanto, há um pequeno erro de precisão na primeira atribuição de *Terco*. Vamos supor que o valor realmente atribuído seja igual a $\frac{1}{3} - \epsilon$, onde $\epsilon > 0$ é este pequeno erro de precisão. A seguir, vamos computar os valores de *Terco* em cada interação:

Interação	<i>Terco</i>
Início	$= \frac{1}{3} - \epsilon$
1	$= 4 \cdot Terco - 1$ $= 4 \cdot \left(\frac{1}{3} - \epsilon\right) - 1$ $= \frac{1}{3} - 4\epsilon$
2	$= 4 \cdot Terco - 1$ $= 4 \cdot \left(\frac{1}{3} - 4 \cdot \epsilon\right) - 1$ $= \frac{1}{3} - 4^2\epsilon$
3	$= 4 \cdot Terco - 1$ $= 4 \cdot \left(\frac{1}{3} - 4^2 \cdot \epsilon\right) - 1$ $= \frac{1}{3} - 4^3\epsilon$
⋮	⋮
30	$= 4 \cdot Terco - 1$ $= 4 \cdot \left(\frac{1}{3} - 4^{29} \cdot \epsilon\right) - 1$ $= \frac{1}{3} - 4^{30}\epsilon$

Note que 4^{30} é um número muito grande, e mesmo que ϵ seja bem pequeno, o valor $4^{30}\epsilon$ chega a ser maior que $\frac{1}{3}$, resultando em um valor negativo para *Terco*. A função $f(n) = 4^n$ é uma função exponencial e tem um crescimento extremamente rápido. A medida que n cresce a função $g(n) := \frac{1}{3} - 4^n\epsilon$ é dominada pelo termo $4^n\epsilon$. Assim, o valor $g(n)$ rapidamente se torna negativo.

A seguinte tabela apresenta a impressão gerada pela execução deste programa (o programa executável foi gerado em um computador Sun-Sparc 4 com o compilador *gpc - Gnu Pascal Compiler*).

Interação:	1,	Terco = 0,3333333333333325931847
Interação:	2,	Terco = 0,33333333333333303727386
Interação:	3,	Terco = 0,33333333333333214909544
Interação:	4,	Terco = 0,33333333333332859638176
Interação:	5,	Terco = 0,33333333333331438552705
Interação:	6,	Terco = 0,33333333333325754210819
Interação:	7,	Terco = 0,33333333333303016843274
Interação:	8,	Terco = 0,33333333333212067373097
Interação:	9,	Terco = 0,33333333332848269492388
Interação:	10,	Terco = 0,33333333331393077969551
Interação:	11,	Terco = 0,33333333325572311878204
Interação:	12,	Terco = 0,33333333302289247512817
Interação:	13,	Terco = 0,33333333209156990051270
Interação:	14,	Terco = 0,33333332836627960205078
Interação:	15,	Terco = 0,33333331346511840820312
Interação:	16,	Terco = 0,33333325386047363281250
Interação:	17,	Terco = 0,33333301544189453125000
Interação:	18,	Terco = 0,33333206176757812500000
Interação:	19,	Terco = 0,33332824707031250000000
Interação:	20,	Terco = 0,33331298828125000000000
Interação:	21,	Terco = 0,33325195312500000000000
Interação:	22,	Terco = 0,33300781250000000000000
Interação:	23,	Terco = 0,33203125000000000000000
Interação:	24,	Terco = 0,32812500000000000000000
Interação:	25,	Terco = 0,31250000000000000000000
Interação:	26,	Terco = 0,25000000000000000000000
Interação:	27,	Terco = 0,00000000000000000000000
Interação:	28,	Terco = -1,00000000000000000000000
Interação:	29,	Terco = -5,00000000000000000000000
Interação:	30,	Terco = -21,00000000000000000000000

Observações:

1. Caso a primeira atribuição de *Terco* seja algo como $\frac{1}{3} + \epsilon$, $\epsilon > 0$, na última interação teríamos algo como $\frac{1}{3} + 4^{30}\epsilon$, que igualmente daria um valor inesperado para *Terco*, mas desta vez positivo.
2. Note que o único lugar onde consideramos um erro de precisão foi na primeira atribuição de *Terco* e todas as outras atribuições envolveram cálculos exatos.

5.6 Tipos definidos pelo programador

Além dos tipos já definidos na linguagem Pascal, é possível se construir novos tipos associados a apenas um identificador. Assim, poderemos declarar tipos complicados e associar este tipo com um identificador. A definição destes novos tipos deve ser feita na área de declarações, usando a palavra reservada **type** seguida das definições de cada tipo. Cada tipo novo é definido como:

$$\text{IdentificadorDeTipo} = \text{especificação_do_tipo};$$

Exemplo 5.5 Declare tipos chamados *MeuInteiro*, *TipoNome* e *TipoNumero*, onde *MeuInteiro* é igual ao tipo **integer**, *TipoNome* é igual ao tipo **string[50]** e *TipoNumero* é igual ao **real**.

```
type
  MeuInteiro = integer;
  TipoNome   = string[50];
  TipoNumero = real;
```

Exemplo 5.6 Os seguintes programas são equivalentes:

```
program ExemploTipos;
type
  TipoNome   = string[50];
  MeuInteiro = integer;
var
  Idade : MeuInteiro;
  Nome  : TipoNome;
begin
  write('Entre com seu nome: ');
  readln(Nome);
  write('Entre com sua idade: ');
  readln(Idade);
  writeln(Nome, ' tem ', idade, ' anos. ');
end.
```

```
program ExemploTipos;
var
  Idade : integer;
  Nome  : string[50];
begin
  write('Entre com seu nome: ');
  readln(Nome);
  write('Entre com sua idade: ');
  readln(Idade);
  writeln(Nome, ' tem ', idade, ' anos. ');
end.
```

Exemplo 5.7 Outros exemplos de declarações de tipos e variáveis.

```
program Exemplo;
type
  TipoNome   = string[50];
  TipoRG     = string[15];
  TipoIdade  = integer;
  TipoSalario = real;
var
  NomePedro, NomePaulo : TipoNome;
  IdadePedro, IdadePaulo : TipoIdade;
  RgPedro, RgPaulo      : TipoRG;
  SalarioPedro, SalarioPaulo : TipoSalario;
```

Algumas vantagens e necessidades de se usar tipos:

- Basta se lembrar do nome do tipo, sem precisarmos escrever toda a especificação de um novo objeto a ser declarado. Um exemplo disto é o caso do *TipoRG* usado no exemplo acima, a cada vez que formos declarar um RG, não precisaremos nos lembrar se um RG é declarado com 15 ou 16 ou mais caracteres; esta preocupação já foi considerada no momento da especificação do tipo *TipoRG*.

- As re-especificações de tipos ficam mais fáceis. Usando **string[15]** em todos os lugares onde declaramos um RG e se quisermos mudar para **string[20]**, devemos percorrer todo o programa procurando pelas declarações de RG's e fazendo as devidas modificações (note que isto não pode ser feito de qualquer forma, já que nem todo lugar onde aparece **string[15]** é uma declaração de RG. Usando tipos, basta mudar apenas uma vez, i.e., na especificação de *TipoRG*.
- Além disso há tipos de objetos da linguagem Pascal que não admitem declarações usando mais que uma palavra. Alguns tipos de objetos deste caso são os parâmetros e as funções, que veremos na próxima seção.