Capítulo 1 : Verificação Formal

❒ Introdução

{ Uma Introdução Informal à Especificação
e Verificação Formais de Algoritmos Iterativos }

Problema: Calcular e^x por desenvolvimento em Série de Taylor.

Como especificar?

Qual a expressão da Série de Taylor para a função exponencial?

Existe soma? Quando converge? Para que valores de x?

Quantos termos somar?

Qual o erro de aproximação permitido?

O que se pretende calcular?

Quais são os Dados e os Resultados?

Qual é o Problema?

Um pouco de Análise Infinitesimal:

O desenvolvimento em série de Taylor da função exponencial

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots + \frac{x^n}{n!} + \cdots$$

é convergente $\forall x \in \mathbb{R}$.

Portanto, a sucessão das Somas Parciais tende para o valor de e^x

$$\lim_{n\to\infty}\{S_n\} = \lim_{n\to\infty}\{1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots + \frac{x^n}{n!}\} = e^x$$

e o Termo Geral tende para zero

$$\lim_{n\to\infty}\{t_n\} = \lim_{n\to\infty}\{\frac{x^n}{n!}\} = 0$$

Problema: Calcular o valor de uma Soma Parcial do desenvolvimento em Série de Taylor da Função Exponencial para um dado $x \in \mathbb{R}$, de modo a que o valor absoluto do último termo somado seja inferior a um dado $\varepsilon \in \mathbb{R}^+$.

Especificação:

Entrada: $\{x : x \in \mathbb{R},\ \varepsilon : \varepsilon \in \mathbb{R}^+\}$

Saída: $\{soma : soma = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots + \frac{x^n}{n!} \wedge\ |\frac{x^n}{n!}| < \varepsilon\}$

{ uma solução do Problema }

Algoritmo:

```
n:= 0; termo:= 1; soma:= 1;

while   abs(termo) >= erro do
        begin    n:= n + 1;
                 termo:= termo * x / n;
                 soma:=  soma + termo
        end;
```

{ mas estará "certo"? }

- **O Algoritmo está "certo" se e só se obedecer às Especificações**

$\{ x \in \mathbb{R} \wedge erro \in \mathbb{R}^+ \}$

```
n:= 0; termo:= 1; soma:= 1;

while   abs(termo) >= erro do
        begin    n:= n + 1;
                 termo:= termo * x / n;
                 soma:=  soma + termo
        end;
```

$\{ soma = 1 + x + x^2/2! + \cdots + x^n/n! \wedge termo = x^n/n! \wedge |termo| < erro \}$

{ mas como provar? }

É condição necessária para que o Algoritmo esteja "certo" que:

$\{ soma = 1 + x + x^2/2! + \cdots + x^n/n! \wedge termo = x^n/n! \}$

(Proposição Lógica envolvendo as **Variáveis Livres** n, termo e soma) permaneça Verdadeira **antes**, **durante** e **depois** do Ciclo.

$\{ x \in \mathbb{R} \wedge erro \in \mathbb{R}^+ \}$

n:= 0; termo:= 1; soma:= 1;

→ $\{ soma = 1 + x + x^2/2! + \cdots + x^n/n! \wedge termo = x^n/n! \}$

while abs(termo) >= erro **do**
→ **begin** $\{ soma = 1 + x + x^2/2! + \cdots + x^n/n! \wedge termo = x^n/n! \}$
n:= n + 1;
termo:= termo * x / n;
soma:= soma + termo
→ $\{ soma = 1 + x + x^2/2! + \cdots + x^n/n! \wedge termo = x^n/n! \}$
end;

→ $\{ soma = 1 + x + x^2/2! + \cdots + x^n/n! \wedge termo = x^n/n! \}$

A Proposição Lógica (Atributo, Asserção, Predicado, ...):

$\{ soma = 1 + x + x^2/2! + \cdots + x^n/n! \wedge termo = x^n/n! \}$

chama-se **Invariante do Ciclo**.

- **Utilização da noção de Invariante na verificação da Correcção de um Ciclo:**

1ª Regra:	O Invariante deve ser Verdadeiro à entrada do Ciclo.
2ª Regra:	A execução do Ciclo deve conservar o Invariante.
3ª Regra:	O Invariante deve ser definido de forma a garantir a Correcção do Algoritmo.
4ª Regra:	O Ciclo deve terminar após um número Finito de iterações.

{ para o exemplo anterior }

1ª Regra: Invariante Verdadeiro à entrada (antes) do Ciclo.

n:= 0; termo:= 1; soma:= 1;

{ n = 0 $\wedge$ termo = 1 $\wedge$ soma = 1 }

{ soma = 1 + x + $x^2/2!$ +⋯+ $x^n/n!$ $\wedge$ termo = $x^n/n!$ } ✓

while abs(termo) >= erro **do**

...

2ª Regra: O Ciclo preserva o Invariante.
(Verdadeiro no Início ⇒ Verdadeiro no Fim)

begin { soma = $1 + x + x^2/2! + \cdots + x^n/n!$ ∧ termo = $x^n/n!$ }

n:= n + 1;

{ soma = $1 + x + x^2/2! + \cdots + x^{(n-1)}/(n-1)!$ ∧ termo = $x^{(n-1)}/(n-1)!$ }

termo:= termo ∗ x / n;

{ termo = $x^{(n-1)}/(n-1)!$ ∗ x / n = $x^n/n!$ }

soma:= soma + termo

{ soma = $1 + x + x^2/2! + \cdots + x^{(n-1)}/(n-1)! + x^n/n!$ }

{ soma = $1 + x + x^2/2! + \cdots + x^n/n!$ ∧ termo = $x^n/n!$ } ✓

end;

3ª Regra: Invariante garante a Correcção do Algoritmo.
(Verdadeiro à Saída do Ciclo ⇒ Algoritmo Correcto)

...
while abs(termo) >= erro **do**
begin …
end;

{ soma = $1 + x + x^2/2! + \cdots + x^n/n!$ ∧ termo = $x^n/n!$ } ∧ { |termo| < erro }

{ soma = $1 + x + x^2/2! + \cdots + x^n/n!$ ∧ termo = $x^n/n!$ ∧ |termo| < erro } ✓

4ª Regra: O Ciclo **pára** após um número finito de iterações.
(A Condição do Ciclo deverá atingir o valor **Falso**)

while abs(termo) >= erro **do**

Que garantia temos de que, em tempo finito,
será atingido um termo, tal que |termo| < erro ?

$$\lim_{n\to\infty}\{t_n\} = 0 \iff \forall\varepsilon \in \mathbb{R}^+,\ \exists N : \forall n \geq N \Rightarrow |t_n| < \varepsilon$$ ✓

- **Os quatro posicionamentos do Invariante para a verificação das quatro Regras:**

[inicializações]
{ Invariante }
while [condição] **do**
begin **{ Invariante }** ∧ { condição }
[instruções]
{ Invariante }
end;
{ Invariante } ∧ { ¬ condição }

- **A Verificação das 4 Regras permite, de modo semi-formal, garantir a Correcção do Ciclo.**

- **A noção de Invariante permite também a Construção do Ciclo.**

Problema:

Calcular o valor de uma Soma Parcial do desenvolvimento em Série de Taylor da Função Exponencial para um dado $x \in \mathbb{R}$, de modo a que o valor absoluto do **primeiro termo desprezado** seja inferior a um dado $\varepsilon \in \mathbb{R}^+$.

Especificação:

Entrada: $\{x : x \in \mathbb{R},\ \varepsilon : \varepsilon \in \mathbb{R}^+\}$

Saída:

$$\{soma : soma = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots + \frac{x^{n-1}}{(n-1)!} \wedge |\frac{x^{n-1}}{(n-1)!}| \geq \varepsilon \wedge |\frac{x^n}{n!}| < \varepsilon\}$$

Construção do Algoritmo:

- **Analisar as Especificações:**

{ $x \in \mathbb{R} \wedge erro \in \mathbb{R}^+$ }

[inicializações]

{ Invariante }

while [condição] **do**

begin { Invariante } $\wedge$ { condição }

[instruções]

{ Invariante }

end;

{ Invariante } $\wedge$ { $\neg$ condição }

{ $soma = 1 + x + x^2/2! + \cdots + x^{(n-1)}/(n-1)! \wedge |x^{(n-1)}/(n-1)!| \geq erro$

$\wedge termo = x^n/n! \wedge |termo| < erro$ }

- **Qual é o Invariante? Qual é a Condição do Ciclo?**

Invariante: { $soma = 1 + x + x^2/2! + \cdots + x^{(n-1)}/(n-1)! \wedge termo = x^n/n!$ }

Condição do Ciclo: { $|termo| \geq erro$ }

- **Posicionar o Invariante e a Condição do Ciclo:**

$\{ x \in \mathbb{R} \wedge erro \in \mathbb{R}^+ \}$

[inicializações]

$\{ soma = 1 + x + x^2/2! + \cdots + x^{(n-1)}/(n-1)! \wedge termo = x^n/n! \}$

while abs(termo) >= erro **do**

begin $\{ soma = 1 + x + x^2/2! + \cdots + x^{(n-1)}/(n-1)! \wedge termo = x^n/n! \}$

$\wedge \{ |termo| \geq erro \}$

[instruções]

$\{ soma = 1 + x + x^2/2! + \cdots + x^{(n-1)}/(n-1)! \wedge termo = x^n/n! \}$

end;

$\{ soma = 1 + x + x^2/2! + \cdots + x^{(n-1)}/(n-1)! \wedge termo = x^n/n! \}$

$\wedge \{ |termo| < erro \}$

$\{ soma = 1 + x + x^2/2! + \cdots + x^{(n-1)}/(n-1)! \wedge \ | x^{(n-1)}/(n-1)! | \geq erro$

$\wedge termo = x^n/n! \wedge |termo| < erro \}$

- **Construir o Ciclo:**

begin $\{ soma = 1 + x + x^2/2! + \cdots + x^{(n-1)}/(n-1)! \wedge termo = x^n/n! \}$

$\wedge \{ |termo| \geq erro \}$

[instruções]

$\{ soma = 1 + x + x^2/2! + \cdots + x^{(n-1)}/(n-1)! \wedge termo = x^n/n! \}$

end;

{ Temos uma soma parcial (correcta) e um novo termo, que ainda pode ser (correctamente) somado. }

begin $\{ soma = 1 + x + x^2/2! + \cdots + x^{(n-1)}/(n-1)! \wedge termo = x^n/n! \} \wedge \{ |termo| \geq erro \}$

soma:= soma + termo;

$\{ soma = 1 + x + x^2/2! + \cdots + x^n/n! \wedge termo = x^n/n! \} \wedge \{ |termo| \geq erro \}$

[instruções]

$\{ soma = 1 + x + x^2/2! + \cdots + x^{(n-1)}/(n-1)! \wedge termo = x^n/n! \}$

end;

{ Precisamos de um novo termo. }

begin $\{ soma = 1 + x + x^2/2! + \cdots + x^{(n-1)}/(n-1)! \wedge termo = x^n/n! \} \wedge \{ |termo| \geq erro \}$

soma:= soma + termo;

$\{ soma = 1 + x + x^2/2! + \cdots + x^n/n! \wedge termo = x^n/n! \} \wedge \{ |termo| \geq erro \}$

n:= n + 1;

$\{ soma = 1 + x + x^2/2! + \cdots + x^{(n-1)}/(n-1)! \} \wedge \{ termo = x^{(n-1)}/(n-1)! \}$

$\wedge \{ |termo| \geq erro \}$

→ $\{ |x^{(n-1)}/(n-1)!| \geq erro \}$

termo:= termo $*$ x / n;

$\{ termo = x^{(n-1)}/(n-1)! * x / n = x^n/n! \}$

$\{ termo = x^n/n! \} \wedge \{ |x^{(n-1)}/(n-1)!| \geq erro \}$

$\{ soma = 1 + x + x^2/2! + \cdots + x^{(n-1)}/(n-1)! \wedge termo = x^n/n! \}$

$\wedge \{ |x^{(n-1)}/(n-1)!| \geq erro \}$

end;

*{ O Ciclo está completo e **correcto**... }*

{ … e à Saída do Ciclo … }

while abs(termo) >= erro **do**

begin soma:= soma + termo;

n:= n + 1;

termo:= termo ∗ x / n;

{ soma = $1 + x + x^2/2! + \cdots + x^{(n-1)}/(n-1)!$ ∧ termo = $x^n/n!$ }

∧ { $|x^{(n-1)}/(n-1)!| \geq$ erro }

end;

{ soma = $1 + x + x^2/2! + \cdots + x^{(n-1)}/(n-1)!$ ∧ termo = $x^n/n!$ } ∧ { $|x^{(n-1)}/(n-1)!| \geq$ erro }

∧ { |termo| < erro }

*{ … temos a **Especificação de Saída**! }*

- **Estabelecer as Inicializações:**

{ $x \in \mathbb{R} \wedge$ erro $\in \mathbb{R}^+$ }

[inicializações]

{ soma = $1 + x + x^2/2! + \cdots + x^{(n-1)}/(n-1)!$ ∧ termo = $x^n/n!$ }

{ Basta escolher valores para as 3 Variáveis Livres, que obedeçam ao Invariante. }

n:= 0; soma:= 0; termo:= 1;

{ ou }

n:= 1; soma:= 1; termo:= x;

{ ou …}

- **Como temos a garantia de que este ciclo pára, o Algoritmo está completo e Correcto:**

$\{ x \in \mathbb{R} \wedge erro \in \mathbb{R}^+ \}$
n:= 0; soma:= 0; termo:= 1;
while abs(termo) >= erro **do**
begin soma:= soma + termo;
n:= n + 1;
termo:= termo $*$ x / n
end;
$\{ soma = 1 + x + x^2/2! + \cdots + x^{(n-1)}/(n-1)! \wedge \ | x^{(n-1)}/(n-1)! | \geq erro$
$\wedge\ termo = x^n/n! \wedge |termo| < erro \}$

⇨ **Uma questão de Análise Numérica:**

Problema: Cálculo de Raízes de Equações
pelo **Método das Bissecções Sucessivas**.

Especificação:

Entrada: { Uma Equação na forma f(x) = 0
e um intervalo [a, b] : $\exists^1$ raiz $\in$ [a ,b]
e um erro $\in \mathbb{R}^+$ }

Saída: { Um intervalo [a, b] : raiz $\in$ [a, b] $\wedge$ |a - b| < erro }

{ Qual é o Invariante?
Qual é a Condição do Ciclo? }

Invariante: { raiz ∈ [a, b] }

Condição do Ciclo: { |a - b| ≥ erro }

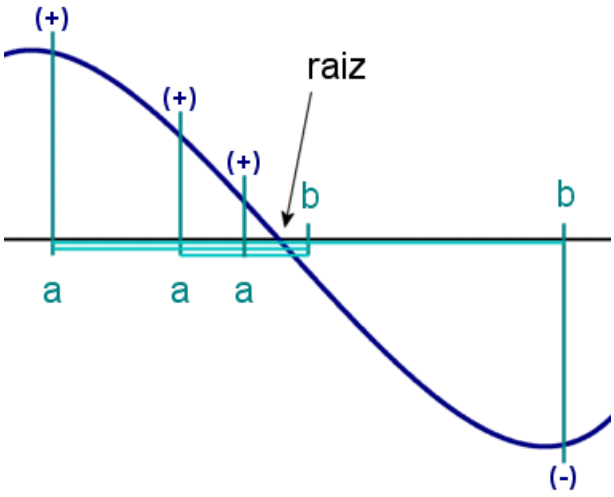
- **Construção do Algoritmo:**

{ f(x) = 0 ∧ raiz ∈ [a ,b] ∧ erro ∈ $\mathbb{R}^+$ }

[inicializações]
{ raiz ∈ [a, b] }
while abs(a – b) >= erro **do**
begin { raiz ∈ [a ,b] ∧ |a - b| ≥ erro }
[instruções]
{ raiz ∈ [a,b] }
end;

{ raiz ∈ [a, b] ∧ |a-b| < erro }

{ Aqui convém recordar o Método das Bissecções Sucessivas … }



meio ← ponto médio de [a, b];
calcular f(meio);
se raiz ∈ [meio, b]
então { [a,b] ← [meio, b] }
a ← meio
senão { [a,b] ← [a, meio] }
b ← meio;

- **Algoritmo e Prova:**

$\{ f(x) = 0 \wedge raiz \in [a, b] \wedge erro \in \mathbb{R}^+ \}$

fa:= f(a);
$\{ raiz \in [a, b] \}$
while abs(a – b) >= erro **do**
begin $\{ raiz \in [a, b] \wedge |a - b| \geq erro \}$
meio:= (a + b) / 2;
fm:= f(meio);
→ **if** (fa > 0) = (fm > 0)
then $\{ raiz \notin [a, meio] \} \Rightarrow \{ raiz \in [meio, b] \}$
a := meio
$\{ raiz \in [a, b] \}$
else $\{ raiz \in [a, meio] \}$
b := meio
$\{ raiz \in [a, b] \}$
$\{ raiz \in [a, b] \}$
end;

$\{ raiz \in [a, b] \wedge |a - b| < erro \}$

- **Observação:**

if (fa > 0) = (fm > 0) ←

then { raiz ∉ [a, meio] }

else { raiz ∈ [a, meio] }

f(a)	f(meio)	raiz ∉ [a,meio]	fa > 0	fm > 0	(fa > 0) = (fm > 0)
+	+	V	V	V	V
+	-	F	V	F	F
-	+	F	F	V	F
-	-	V	F	F	V

- **Paragem do Ciclo:**

while abs(a – b) >= erro **do**

Que garantia temos de que, em tempo finito,
será atingido um intervalo [a, b], tal que |a - b| < erro ?

O Algoritmo gera uma sucessão de intervalos, obtidos por Bissecções Sucessivas do intervalo [a, b] inicial, portanto:

$$\lim_{n\to\infty}\{\frac{|a-b|}{2^n}\} = 0 \iff \forall\varepsilon \in \mathbb{R}^+, \exists N : \forall n \geq N \Rightarrow \frac{|a-b|}{2^n} < \varepsilon$$

⇨ Uma questão de Teoria dos Números:

Problema: Cálculo do Máximo Divisor Comum pelo **Algoritmo de Euclides**.

Especificação:

Entrada: $\{ a, b : a, b \in \mathbb{N}_0, \neg (a = 0 \wedge b = 0) \}$

Saída: $\{ x : x = mdc(a, b) \}$

mdc(380, 55) = ?

a	b	resto
380	**55**	50
55	50	5
50	5	0
5	**0**	

mdc(380, 55) = 5

mdc(55, 380) = ?

a	b	resto
55	**380**	55
380	55	50
55	50	5
50	5	0
5	**0**	

mdc(55, 380) = 5

- **O Algoritmo resulta da aplicação directa das 3 Propriedades:**

mdc(a, 0) = a

mdc(a, b) = mdc(a mod b, b)

mdc(a, b) = mdc(b, a)

- **Algoritmo de Euclides:**

```
x:= a; y:= b;
while   y <> 0  do
        begin   resto:= x mod y;
                x:= y;
                y:= resto
        end;
```

- **Algoritmo de Euclides e sua Demonstração:**

{ a, b : a, b $\in \mathbb{N}_0$, (a $\neq$ 0 $\vee$ b $\neq$ 0) }

x:= a; y:= b;

{ mdc(a, b) = mdc(x, y) }

while y <> 0 **do**

begin { mdc(a, b) = mdc(x, y) } $\wedge$ { y $\neq$ 0 }

{ mdc(x, y) = mdc(x mod y, y) = mdc(y, x mod y) }

resto:= x **mod** y;

{ mdc(a, b) = mdc(y, resto) }

x:= y;

{ mdc(a, b) = mdc(x, resto) }

y:= resto

{ mdc(a, b) = mdc(x, y) }

end;

{ mdc(a, b) = mdc(x, y) } $\wedge$ { y = 0 }

{ x = mdc(a, b) }

{ Verifique os casos em que a *ou* b *são nulos. }*

- **A noção de Invariante permite ainda a construção de Algoritmos Iterativos destinados a Demonstrar Teoremas.**

Problema: Somar os n primeiros números naturais, **demonstrando** também que:

$$1 + 2 + 3 + \cdots + n = n\,(n+1) / 2$$

Especificação:

Entrada: $\{ n : n \in \mathbb{N} \}$

Saída: $\{ soma : \; soma = 1 + 2 + 3 + \cdots + n \; \wedge \; soma = n\,(n+1) / 2 \}$

- **Construção do Algoritmo:**
  A Condição de Paragem do Ciclo:

$\{ n \in \mathbb{N} \}$

k:= 1;
$\{ k = 1 \}$
while k < n **do**
begin $\{ k < n \} \equiv \{ k+1 < n+1 \}$
k:= k+1;
$\{ k < n+1 \}$

...

$\{ k < n+1 \}$
end;

$\{ k \geq n \; \wedge \; k < n+1 \} \Rightarrow$ **$\{ k = n \}$**

- **Construção do Algoritmo: O Somatório:**

{ n ∈ ℕ }
k:= 1; soma:= 1;
{ k = 1 } ∧ { soma = 1 + 2 + … + k }
while k < n **do**
begin { soma = 1 + 2 + … + k } ∧ { k < n }
k:= k+1;
{ soma = 1 + 2 + … + (k-1) } ∧ { k < n+1 }
soma:= soma + k
{ soma = 1 + 2 + … + (k-1) + k } ∧ { k < n+1 }
end;
{ soma = 1 + 2 + … + k } ∧ { k < n+1 } ∧ { k ≥ n }
{ soma = 1 + 2 + … + n }

- **O Algoritmo e a Demonstração pretendida:**

{ n ∈ ℕ }
k:= 1; soma:= 1;
{ k = 1 } ∧ { soma = 1 + 2 + … + k } ∧ { soma = k (k+1) / 2 }

while k < n **do**
begin { soma = 1 + 2 + … + k } ∧ {k < n} ∧ { soma = k (k+1) / 2 }
k:= k+1;
{ soma = 1 + 2 + … + (k-1) } ∧ { k < n+1 }
∧ { soma = (k-1) k / 2 }
soma:= soma + k
{ soma = 1 + 2 + … + k } ∧ { k < n+1 }
∧ { soma = (k-1) k / 2 + k = k (k+1) / 2 }
end;

{ soma = 1 + 2 + … + k } ∧ { k < n+1 } ∧ { k ≥ n }
∧ { soma = k (k+1) / 2 }

{ soma = 1 + 2 + … + n } ∧ { soma = n (n+1) / 2 }

- **Acabámos de Demonstrar, pelo Princípio da Indução que:**

$$\sum_{k=1}^{n} k = \frac{n(n+1)}{2}$$

A noção de Invariante representa o Princípio da Indução Matemática.

- **Com a noção de Invariante podemos ainda Deduzir Algoritmos.**

Problema: Calcular os n primeiros **quadrados perfeitos**, utilizando apenas somas.

Especificação:

Entrada: { n : n ∈ ℕ }

Saída: { soma : soma = 1 , 4, 9, 16, 25, …, n^2 }

- **Dedução do Algoritmo: O Ciclo:**

$\{ n \in \mathbb{N} \}$
k:= 1;
$\{ k \in \mathbb{N} \}$
while k < n **do**
begin $\{ k \in \mathbb{N} \} \wedge \{ k < n \}$
k:= k+1;
$\{ k \in \mathbb{N} \} \wedge \{ k < n+1 \}$
...
$\{ k \in \mathbb{N} \} \wedge \{ k < n+1 \}$
end;
$\{ k \in \mathbb{N} \} \wedge \{ k \geq n \ \wedge \ k < n+1 \}$
$\{ k = n \}$

- **Dedução do Algoritmo: O Somatório:**

$\{ n \in \mathbb{N} \}$
k:= 1; soma:= 1;
$\{ k \in \mathbb{N} \} \wedge \{ soma = k^2 \}$
while k < n **do**
begin $\{ k \in \mathbb{N} \} \wedge \{ soma = k^2 \} \wedge \{ k < n \}$
k:= k+1;
$\{ k \in \mathbb{N} \} \wedge \{ soma = (k-1)^2 \} \wedge \{ k < n+1 \}$
$\{ soma = k^2 - 2k + 1 \}$
...
soma:= soma + termo
$\{ soma = k^2 \}$
$\{ k \in \mathbb{N} \} \wedge \{ soma = k^2 \} \wedge \{ k < n+1 \}$
end;
$\{ k \in \mathbb{N} \} \wedge \{ soma = k^2 \} \wedge \{ k \geq n \ \wedge \ k < n+1 \}$
$\{ k = n \}$
$\{ soma = n^2 \}$

- **Dedução do Algoritmo: O Termo do Somatório:**

{ Qual o valor do termo? }

{ soma = $k^2 - 2k + 1$ }

...

soma:= soma + termo

{ soma = k^2 }

{ termo = 2k - 1 }

{ $n \in \mathbb{N}$ }

k:= 1; termo:= 1; soma:= 1;

{ $k \in \mathbb{N}$ } $\wedge$ { termo = $2k - 1$ } $\wedge$ { soma = k^2 }

while k < n **do**

begin { $k \in \mathbb{N}$ } $\wedge$ { termo = $2k - 1$ } $\wedge$ { soma = k^2 } $\wedge$ { k < n }

k:= k+1;

{ $k \in \mathbb{N}$ } $\wedge$ { termo = $2 (k-1) - 1$ } $\wedge$ { soma = $(k-1)^2$ } $\wedge$ { k < n+1 }

{ termo = $2k - 3$ } $\wedge$ { soma = $k^2 - 2k + 1$ }

...

{ termo = $2k - 1$ }

soma:= soma + termo

{ soma = $k^2 - 2k + 1 + 2k - 1 = k^2$ }

{ soma = k^2 }

{ $k \in \mathbb{N}$ } $\wedge$ { soma = k^2 } $\wedge$ { k < n+1 }

end;

{ $k \in \mathbb{N}$ } $\wedge$ { soma = k^2 } $\wedge$ { $k \geq n \wedge k < n+1$ }

{ k = n }

{ soma = n^2 }

{ Qual o incremento do termo? }

- **O Algoritmo e sua Demonstração:**

$\{ n \in \mathbb{N} \}$
k:= 1; termo:= 1; soma:= 1;
$\{ k \in \mathbb{N} \} \wedge \{ termo = 2k - 1 \} \wedge \{ soma = k^2 \}$

while k < n **do**
begin $\{ k \in \mathbb{N} \} \wedge \{ termo = 2k - 1 \} \wedge \{ soma = k^2 \} \wedge \{ k < n \}$
k:= k+1;
$\{ k \in \mathbb{N} \} \wedge \{ termo = 2 (k-1) - 1 \} \wedge \{ soma = (k-1)^2 \} \wedge \{ k < n+1 \}$
$\{ termo = 2k - 3 \} \wedge \{ soma = k^2 - 2k + 1 \}$
termo:= termo + 2;
$\{ termo = 2k - 3 + 2 = 2k - 1 \}$
soma:= soma + termo
$\{ soma = k^2 - 2k + 1 + 2k - 1 = k^2 \}$
$\{ soma = k^2 \}$
$\{ k \in \mathbb{N} \} \wedge \{ soma = k^2 \} \wedge \{ k < n+1 \}$
end;

$\{ k \in \mathbb{N} \} \wedge \{ soma = k^2 \} \wedge \{ k \geq n \ \wedge \ k < n+1 \}$
$\{ soma = n^2 \}$

- **Só o Algoritmo:**

k:= 1; termo:= 1; soma:= 1;

while k < n **do**
begin k:= k+1;
termo:= termo + 2;
soma:= soma + termo
end;

{ Gera quadrados perfeitos … }

k	termo	soma
1	1	1
2	3	1 + 3 = 4
3	5	1 + 3 + 5 = 9
4	7	1 + 3 + 5 + 7 = 16
5	9	1 + 3 + 5 + 7 + 9 = 25
…		
n	2n – 1	n^2

- **A dedução do Algoritmo gerou, por Indução, a propriedade:**

$$\sum_{k=1}^{n} (2k - 1) = n^2$$

- **De forma (ainda) mais simples, podemos escrever:**

```
k:= 1; termo:= 1; soma:= 1;

for  k:= 2 to n do
     begin  termo:= termo + 2;
            soma:= soma + termo
     end;
```

- **Exercício:** Calcular os n primeiros **cubos perfeitos**, utilizando apenas somas.

- **Exercício:** Construa um Algoritmo para Demonstrar que:

$$1^3 + 2^3 + 3^3 + \cdots + n^3 = n^2 (n+1)^2 / 4 = (1 + 2 + 3 + \cdots + n)^2$$

- **Podemos ainda Reutilizar Algoritmos.**

Problema: Se temos um Algoritmo para calcular n^2, poderemos **alterá-lo** de forma a calcular $\sqrt{n}$?

Claro que, estando em Aritmética de Inteiros, só poderemos calcular uma **aproximação inteira** de $\sqrt{n}$.

Especificação: Entrada: $\{ n : n \in \mathbb{N} \}$

Saída: $\{ k : k = \lceil \sqrt{n} \rceil \}$

Nota: $k = \lceil x \rceil \Leftrightarrow (k-1) < x \leq k$

$k = \lfloor x \rfloor \Leftrightarrow k \leq x < (k+1)$

- **O que temos do Algoritmo anterior:**

$\{ n \in \mathbb{N} \}$

```
k:= 1; termo:= 1; soma:= 1;
while  k < n do
       begin  k:= k+1;
              termo:= termo + 2;
              soma:= soma + termo
       end;
```

$\{ k \in \mathbb{N} \} \wedge \{ soma = k^2 \} \wedge \{ k \geq n \} \wedge \{ k < n+1 \}$

$\{ k \in \mathbb{N} \} \wedge \{ soma = k^2 \} \wedge \{ k \geq n \} \wedge \{ k < n+1 \}$

- **O que pretendemos do novo Algoritmo:**

$$(k = \lceil \sqrt{n} \rceil \Leftrightarrow (k-1) < \sqrt{n} \leq k)$$

$\{ k \in \mathbb{N} \} \wedge \{ (k-1) < \sqrt{n} \} \wedge \{ k \geq \sqrt{n} \} \equiv$

$\{ k \in \mathbb{N} \} \wedge \{ (k-1)^2 < n \} \wedge \{ k^2 \geq n \}$

- **Basta portanto alterar a Condição do Ciclo:**

$\{ n \in \mathbb{N} \}$
k:= 1; termo:= 1; soma:= 1;
$\{ k \in \mathbb{N} \} \wedge \{ soma = k^2 \}$

while soma < n **do**

begin $\{ k \in \mathbb{N} \} \wedge \{ soma = k^2 \} \wedge \{ soma < n \}$
$\{ k^2 < n \}$
k:= k+1;
$\{ (k-1)^2 < n \}$
termo:= termo + 2;
soma:= soma + termo
$\{ soma = k^2 \}$
$\{ k \in \mathbb{N} \} \wedge \{ soma = k^2 \} \wedge \{ (k-1)^2 < n \}$

end;

$\{ k \in \mathbb{N} \} \wedge \{ soma = k^2 \} \wedge \{ (k-1)^2 < n \} \wedge \{ soma \geq n \}$

$\{ k \in \mathbb{N} \} \wedge \{ (k-1)^2 < n \} \wedge \{ k^2 \geq n \}$

$\{ k \in \mathbb{N} : k = \lceil \sqrt{n} \rceil \}$

❒ A Axiomática de Hoare

{ Sistema Formal para a Verificação da Correcção Parcial de Algoritmos. }

C.A.R. Hoare: *"An Axiomatic Basis for Computer Programming"*, Comm. ACM 12(10), pp. 576-583, 1969.

C. A. R. Hoare and N. Wirth: *"An axiomatic definition of the programming language PASCAL"*, *Acta Informatica*, 2(4):335-355, 1973.

- **Uma Lógica Predicativa para a Especificação e Verificação de Programas (Algoritmos) numa Linguagem Imperativa.**
- **Conjunto de Axiomas e de Regras de Inferência.**
- **Especificação de um Algoritmo:**

Antecedente
Pré – Condição
Condição de Entrada

Consequente
Pós – Condição
Condição de Saída

$$\{P\}\ S\ \{Q\}$$

{ Se { P } for verdadeiro antes da execução de S, então { Q } é verdadeiro após a execução de S.}

- **O Axioma da Atribuição:**

{ Para que {P(x)} seja verificado após a execução de x:= e,
é suficiente que {P(e)} seja verdadeiro antes.}

$$\{P(e)\}\ x := e\ \{P(x)\}$$

Predicado obtido a partir de *{ P(x) }*,
substituindo todas as Ocorrências Livres de *x* por *e*.

Ex. :

{ soma = 1 + 2 + 3 + … + n }

{ soma = 1 + 2 + 3 + … + (n + 1 – 1) }

n:= n + 1;

{ soma = 1 + 2 + 3 + … + (n – 1) }

Ex. :

{ soma = k^2 – 2k + 1 }

{ soma = k^2 – (**2k – 1**) }

{ termo = 2k – 1 }

soma:= soma + termo

{ soma = k^2 }

- **As Regras de Inferência:**

$$\frac{\{P\}\ S\ \{Q\},\ \ Q \Rightarrow R}{\{P\}\ S\ \{R\}}$$

{ Se { P } garante { Q }, após a execução de S, então também garante todo o Predicado implicado por { Q }.}

Ex.:
{ x > 0 }
x:= 2 ∗ x;
{ x par }
{ x par } ⇒ { x+1 ímpar }
{ x+1 ímpar }

$$\frac{P \Rightarrow Q,\ \ \{Q\}\ S\ \{R\}}{\{P\}\ S\ \{R\}}$$

{ Se { Q } for antecedente para S produzir { R }, então também é antecedente todo o Predicado que implique { Q }.}

Ex.:
{ x+1 ímpar }
{ x+1 ímpar } ⇒ { x par }
{ x par }
r:= x **mod** 2;
{ r = 0 }

- **A Regra da Composição Sequencial:**

$$\frac{\{P\}\ S_1\ \{Q\} \quad \{Q\}\ S_2\ \{R\}}{\{P\}\ S_1\ ;\ S_2\ \{R\}}$$

{ Se a execução de S_1 com antecedente { P } garante { Q },
e se a execução de S_2 com antecedente { Q } garante { R },
então a execução de S_1 ; S_2 com antecedente { P } garante { R }.}

Ex.: { x = a ∧ y = b }

aux:= x;
{ aux = a ∧ x = a ∧ y = b }
x:= y;
{ x = b ∧ aux = a ∧ y = b }
y:= aux;
{ y = aux ∧ x = b ∧ aux = a }

{ x = b ∧ y = a }

- **Generalização:**

$$\frac{\{P_{i-1}\}\ S_i\ \{P_i\}\ \forall\ i = 1, 2, \ldots, n}{\{P_0\}\ S_1\ ;\ S_2\ ;\ \ldots\ ;\ S_n\ \{P_n\}}$$

{ ... }

- **As Regras da Alternativa**:

$$\frac{\{P \text{ and } B\}\ S_1\ \{Q\} \qquad \{P \text{ and not } B\}\ S_2\ \{Q\}}{\{P\} \text{ if } B \text{ then } S_1 \text{ else } S_2\ \{Q\}}$$

Ex.: { raiz ∈ [a, b] }

if (fa > 0) = (fm > 0)
then { raiz ∉ [a, meio] ∧ raiz ∈ [a, b] } ⇒ { raiz ∈ [meio, b] }
a := meio
{ raiz ∈ [a, b] }
else { raiz ∈ [a, meio] }
b := meio
{ raiz ∈ [a, b] }

{ raiz ∈ [a, b] }

$$\frac{\{P \text{ and } B\}\ S\ \{Q\} \qquad \{P \text{ and not } B\} \Rightarrow \{Q\}}{\{P\} \text{ if } B \text{ then } S\ \{Q\}}$$

Ex.: { $x = x_0$ }

if x < 0
then { $x = x_0 \wedge x < 0$ }
x := - x
{ $x = -x_0 \wedge x > 0$ }
{ $x = |x_0|$ }
{ $x = x_0 \wedge x \geq 0$ } ⇒ { $x = |x_0|$ }

{ $x = |x_0|$ }

Regras da Repetição:

- **A Regra do Ciclo <u>while</u>:**

$$\frac{\{I \text{ and } B\}\ S\ \{I\}}{\{I\}\ \text{while } B \text{ do } S\ \{I \text{ and not } B\}}$$

{ Relacionar com as Quatro Regras anteriores e com o Princípio da Indução. }

```
{ I }
while  B  do
       begin  { I ∧ B }
              S
              { I }
       end;
{ I ∧ ¬ B }
```

{ Caso Base }

{ Passagem Indutiva }

Problema: Calcular o **valor máximo** de um conjunto de inteiros.

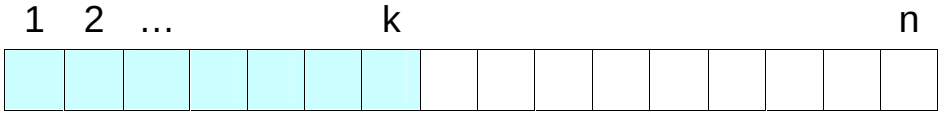
Especificação: Entrada: { $x[1..n] \in \mathbb{Z}^n$ }

Saída: { maximo = max {x[1..n]} }

Formulação: { maximo = max {x[1..n]} } ≡

{ ∃ maximo ∈ x[1..n] : ∀ k ∈ [1..n] ⇒ x[k] ≤ maximo}

Estratégia: Conservar Invariante: { maximo = max { x[1..k] } }



Algoritmo e Prova:

{ x[1..n] ∈ $\mathbb{Z}^n$ }

k:= 1; maximo:= x[1];
{ maximo = max {x[1..k]} }

while k < n **do**
begin { maximo = max {x[1..k]} ∧ (k+1-1 < n) }
k := k + 1;
{ maximo = max {x[1..k-1]} ∧ (k-1 < n) }
if x[k] > maximo
then { maximo = max {x[1..k-1]} ∧ x[k] > maximo }
máximo:= x[k]
{ maximo = max {x[1..k]} }
{ maximo = max {x[1..k-1]} ∧ x[k] ≤ maximo } ⇒
{ maximo = max {x[1..k]} }
{ maximo = max {x[1..k]} ∧ (k-1 < n) }
end;

{ maximo = max {x[1..k]} ∧ (k-1 < n) ∧ (k ≥ n) }
{ maximo = max {x[1..n]} }

Exercício: Calcular o **índice** do valor máximo.

Problema: **Pesquisa Sequencial**.

Entrada: $\{ x \in \mathbb{Z} , a[1..n] \in \mathbb{Z}^n \}$

Saída: $\{$ **?** $x \in a[1..n] \}$

Formulação: $\{ x \in a[1..n] \} \equiv \{ \exists\, k \in [1..n] : x = a[k] \}$

$\{ x \notin a[1..n] \} \equiv \{ \forall\, k \in [1..n] \Rightarrow x \neq a[k] \}$

{ Um Ciclo ... }

Especificação:

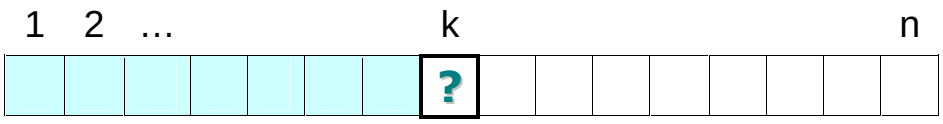
Entrada: $\{ x \in \mathbb{Z} , a[1..n] \in \mathbb{Z}^n \}$

Saída: $\{$ achou $\in \{ V, F \}$:

achou = ($\exists\, k \in [1..n]$: $x = a[k]$) $\}$

Estratégia: Conservar Invariante:

$\{ x \notin a[1..k-1] \wedge$ achou = ($x = a[k]$) $\}$



Algoritmo e Prova:

$\{ x \in \mathbb{Z} , a[1..n] \in \mathbb{Z}^n \}$

k:= 1;
achou:= x = a[1];
$\{ a[1..k-1] = \emptyset \}$
$\{ x \notin a[1..k-1] \wedge achou = (x = a[k]) \}$

while **not** achou **and** (k < n) **do**
begin $\{ x \notin a[1..k-1] \wedge achou = (x = a[k]) \} \wedge \{ \neg achou \wedge (k < n) \}$
$\{ x \notin a[1..k] \wedge (k < n) \}$

k := k + 1;
$\{ x \notin a[1..k-1] \wedge (k-1 < n) \}$

achou:= x = a[k]

$\{ x \notin a[1..k-1] \wedge achou = (x = a[k]) \wedge (k-1 < n) \}$

end;

$\{ x \notin a[1..k-1] \wedge achou = (x = a[k]) \wedge (k-1 < n) \} \wedge \{ achou \vee (k \geq n) \}$

$\{ x \notin a[1..k-1] \wedge achou = (x = a[k]) \wedge (k-1 < n) \wedge achou \}$ V
$\{ x \notin a[1..k-1] \wedge achou = (x = a[k]) \wedge (k-1 < n) \wedge (k \geq n) \}$

$\{ achou \wedge (x = a[k]) \wedge k \in [1..n] \}$ V $\{ achou = (x = a[n]) \}$

$\{ achou = (\exists k \in [1..n] : x = a[k]) \}$

$\{ achou = x \in a[1..n] \}$

{ Se achou*, o valor de* k *à saída do Ciclo é o da primeira ocorrência de* x = a[k]*. }*

Nota: Como $\{ \neg A \vee B \} \equiv \{ A \Rightarrow B \}$

$\{ achou \vee (k \geq n) \} \equiv \{ \neg achou \Rightarrow (k \geq n) \}$

portanto, no caso de **{ ¬ achou }** , temos:

$\{ x \notin a[1..k-1] \wedge (x \neq a[k]) \wedge (k-1 < n) \wedge (k \geq n) \}$

$\{ x \notin a[1..k-1] \quad (x \neq a[k]) \wedge (k = n) \}$

$\{ x \notin a[1..n] \}$

Assim, como alternativa aos cálculos à saída do Ciclo:

a) **{ achou }**

$\{ achou = (x = a[k]) \wedge (k-1 < n) \}$

$\{ (x = a[k]) \wedge k \in [1..n] \}$

b) **{ ¬ achou }**

$\{ x \notin a[1..k-1] \quad (x \neq a[k]) \wedge (k = n) \}$

$\{ x \notin a[1..n] \}$

Nota: Como alternativa para as inicializações,

```
k:= 0;
achou:= false;
```

{ Teste de igualdade entre um Valor Definido e um Valor Indefinido. }

$\{ a[1..k-1] = \emptyset \} \wedge \{ (x = a[k]) = false \}$

$\{ x \notin a[1..k-1] \wedge achou = (x = a[k]) \}$

Problema: **Inclusão de Conjuntos**.

Entrada: { a[1..m] $\in \mathbb{Z}^m$, b[1..n] $\in \mathbb{Z}^n$ }

Saída: { **?** a[1..m] $\subseteq$ b[1..n] }

Formulação:

{a[1..m] $\subseteq$ b[1..n]} $\equiv$ { $\forall$ i $\in$ [1..m], $\exists$ j $\in$ [1..n] : a[i] = b[j] }

{ Dois Ciclos ... }

Especificação:

Entrada: { a[1..m] $\in \mathbb{Z}^m$, b[1..n] $\in \mathbb{Z}^n$ }

Saída: { contido $\in$ { V, F } :

contido = ($\forall$ i $\in$ [1..m], $\exists$ j $\in$ [1..n] : a[i] = b[j]) }

Observação: Já temos um módulo para a Pesquisa,

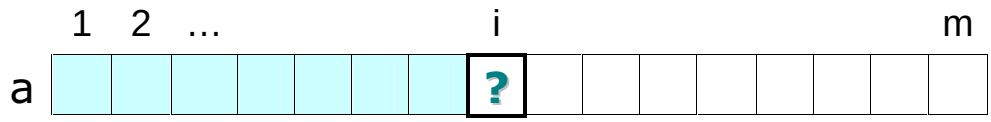
```
{ Pesquisar a[i] em b[1..n] }

j:= 0;
achou:= false;
while   not achou and (j < n) do
        begin   j := j + 1;
                achou:= a[i] = b[j]
        end;

{ achou =  a[i] ∈ b[1..n]  }
```

Estratégia: Conservar Invariante:

$\{ a[1..i-1] \subseteq b[1..n] \wedge contido = (a[i] \in b[1..n]) \}$



Algoritmo:

```
i:= 0; contido:= true;
{ ∅ ⊆ b[1..n] }

while  contido and (i < m) do
       begin  { a[1..i-1] ⊆ b[1..n] } ∧ { a[i] ∈ b[1..n]  }
              { a[1..i] ⊆ b[1..n] } }
              i := i + 1;
              { a[1..i-1] ⊆ b[1..n] }

              { Pesquisar a[i] em b[1..n] }
              j:= 0;  achou:= false;
              while   not achou and (j < n) do
                      begin  j := j + 1;
                             achou:= a[i] = b[j]
                      end;
              { achou =  a[i] ∈ b[1..n]  }

              contido:= achou
       end;

{ contido = ( a[1..m] ⊆ b[1..n] ) }
```

{ Se ¬ contido*, o valor de* i *à saída do Ciclo é o da primeira ocorrência de* a[i] ∉ b[1..n]*. }*

Algoritmo e Prova:

```
i:= 0;
contido:= true;
{ a[1..i-1] = ∅ } ∧ { ∅ ⊆ b[1..n] }

{ a[1..i-1] ⊆ b[1..n] ∧ contido = ( a[i] ∈ b[1..n] ) }

while contido and (i < m) do
        begin  { a[1..i-1] ⊆ b[1..n] ∧ contido = ( a[i] ∈ b[1..n] ) } ∧
                                                    { contido ∧ ( i < m) }
               { a[1..i] ⊆ b[1..n] } ∧ ( i < m) }
               i := i + 1;
               { a[1..i-1] ⊆ b[1..n] ∧ ( i-1 < m) }

               { Pesquisar a[i] em b[1..n] }
               j:= 0;
               achou:= false;
               while   not achou and (j < n) do
                       begin   j := j + 1;
                               achou:= a[i] = b[j]
                       end;
               { achou =  a[i] ∈ b[1..n]  }

               contido:= achou

               { a[1..i-1] ⊆ b[1..n] ∧ contido = ( a[i] ∈ b[1..n] ) ∧ ( i-1 < m) }
        end;


{ a[1..i-1] ⊆ b[1..n] ∧ contido = ( a[i] ∈ b[1..n] ) ∧ ( i-1 < m) } ∧
                                                    { ¬ contido ∨ (i ≥ m) }
```

$\{ a[1..i-1] \subseteq b[1..n] \wedge contido = (a[i] \in b[1..n]) \wedge (i-1 < m) \} \wedge$
$\{ \neg contido \vee (i \geq m) \}$

a) **{ contido }**

$\{ \neg contido \vee (i \geq m) \} \equiv \{ contido \Rightarrow (i \geq m) \}$

$\{ a[1..i-1] \subseteq b[1..n] \wedge a[i] \in b[1..n] \wedge (i-1 < m) \wedge (i \geq m) \}$

$\{ a[1..i-1] \subseteq b[1..n] \wedge a[i] \in b[1..n] \wedge (i = m) \}$

$\{ a[1..m-1] \subseteq b[1..n] \wedge a[m] \in b[1..n] \}$

$\{ \forall i \in [1..m], \exists j \in [1..n] : a[i] = b[j] \}$

$\{ a[1..m] \subseteq b[1..n] \}$

b) **{ ¬ contido }**

$\{ a[1..i-1] \subseteq b[1..n] \wedge contido = (a[i] \in b[1..n]) \wedge (i-1 < m) \}$

$\{ a[1..i-1] \subseteq b[1..n] \quad a[i] \not\in b[1..n] \wedge (i-1 < m) \}$

$\{ \exists i \in [1..m] : a[i] \notin b[1..n] \}$

{ ... e conhecemos esse i*. }*

$\{ a[1..m] \not\subseteq b[1..n] \}$

$\{ contido = (a[1..m] \subseteq b[1..n]) \}$

Problema: **Pesquisa Binária** num vector ordenado.

Entrada: $\{ x \in \mathbb{Z} , a[1..n] \in \mathbb{Z}^n : \forall i, j \in [1..n], \ i < j \ \Rightarrow \ a[i] \leq a[j] \}$

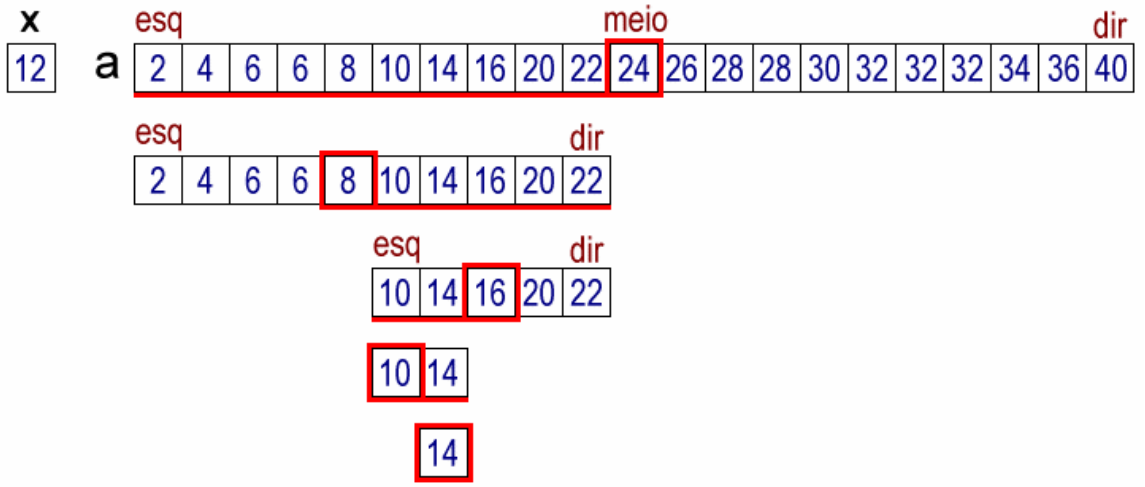
Saída: $\{$ **?** $x \in a[1..n] \}$

Especificação:

Entrada: $\{ x \in \mathbb{Z} , a[1..n] \in \mathbb{Z}^n : \forall i, j \in [1..n], \ i < j \ \Rightarrow \ a[i] \leq a[j] \}$

Saída: $\{$ achou $\in \{ V, F \} :$ achou $= (\exists k \in [1..n] \ : x = a[k]) \}$

Estratégia: Partições Binárias …



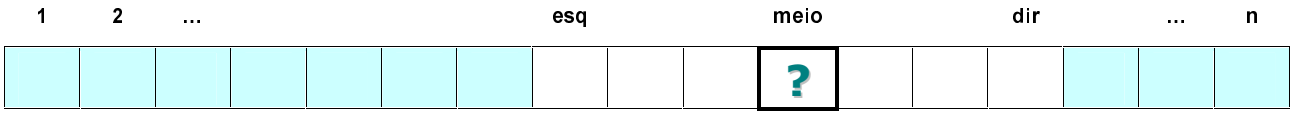
Condição do Ciclo: **not** achou **and** (esq <= dir)

Invariante: ?

Algoritmo:

```
esq:= 1; dir:= n;
achou:= false;
while  not achou and (esq <= dir) do
       begin  meio:= (esq+dir) div 2;
              if x = a[meio]
              then achou:= true
              else  if x < a[meio]
                    then  dir:= meio-1
                    else  esq:= meio+1
       end;
```

{ achou = $x \in a[1..n]$ }



Invariante (1ª versão):

{ $x \notin a[1..esq-1] \wedge x \notin a[dir+1..n] \wedge achou = (x = a[meio])$ }

{ Fazer ... }

Invariante (2ª versão):

{ $(x \in a[1..n] \Rightarrow x \in a[esq..dir]) \wedge achou = (x = a[meio])$ }

Algoritmo e Prova:

esq:= 1; dir:= n; achou:= false;

{ (x ∈ a[1..n] ⇒ x ∈ a[esq..dir]) ∧ achou = (x = a[meio]) } ∧
{ esq, dir ∈ [1..n] }

while **not** achou **and** (esq <= dir) **do**
begin { (x ∈ a[1..n] ⇒ x ∈ a[esq..dir]) ∧
achou = (x = a[meio]) } ∧ { esq, dir ∈ [1..n] } ∧
{ ¬ achou ∧ (esq ≤ dir) }
{ x ≠ a[meio] }
meio:= (esq+dir) **div** 2;
{ meio ∈ [esq..dir] ⊆ [1..n] }

if x = a[meio]
then { x = a[meio] }
achou:= true
{ (x = a[meio]) ∧ achou ∧ meio ∈ [1..n] }
else { x ≠ a[meio] ∧ ¬ achou }
if x < a[meio]
then { x ∉ a[meio..n] } { ordenado … }
{ (x ∈ a[1..n] ⇒ x ∈ a[esq..meio-1]) }
dir:= meio-1
{ dir ∈ [1..n] }
{ (x ∈ a[1..n] ⇒ x ∈ a[esq..dir]) }
else { x ≠ a[meio] ∧ x ≥ a[meio] }
{ x > a[meio] }
{ x ∉ a[1..meio] } { ordenado … }
{ (x ∈ a[1..n] ⇒ x ∈ a[meio+1..dir]) }
esq:= meio+1
{ esq ∈ [1..n] }
{ (x ∈ a[1..n] ⇒ x ∈ a[esq..dir]) }
{ (x ∈ a[1..n] ⇒ x ∈ a[esq..dir]) ∧
achou = (x = a[meio]) } ∧ { esq, dir, meio ∈ [1..n] }
end;

$\{ (x \in a[1..n] \Rightarrow x \in a[esq..dir]) \wedge achou = (x = a[meio]) \} \wedge$
$\{ esq, dir, meio \in [1..n] \} \wedge \{ achou \vee (esq > dir) \}$

a) **$\{ achou \}$**

$\{ achou = (x = a[meio]) \wedge (meio \in [1..n]) \}$

$\{ x = a[meio] \wedge (meio \in [1..n]) \}$

b) **$\{ \neg achou \}$**

$\{ achou \vee (esq > dir) \} \equiv \{ \neg achou \Rightarrow (esq > dir) \}$

$\{ (x \in a[1..n] \Rightarrow x \in a[esq..dir]) \wedge (esq > dir) \}$

$\{ x \in a[1..n] \Rightarrow x \in \emptyset \}$

$\{ x \in a[1..n] \Rightarrow Falso \}$

$\{ \neg x \in a[1..n] \}$

$\{ x \notin a[1..n] \}$

$\{ achou = x \in a[1..n] \}$

Problema: Verificar **se** um dado vector está **Ordenado**.

Entrada: $\{ a[1..n] \in \mathbb{Z}^n \}$

Saída: $\{ \mathbf{?}\ \forall\ i, j \in [1..n],\ i < j \Rightarrow a[i] \leq a[j] \}$

Especificação de Saída:

$\{$ ordenado $\in \{ V, F \}$:
ordenado = $(\forall\ i, j \in [1..n],\ i < j \Rightarrow a[i] \leq a[j]) \}$

Estratégia: $(\forall\ i, j \in [1..n],\ i < j \Rightarrow a[i] \leq a[j]) \equiv$

{ Dois ciclos ... }

$(\forall\ k \in [1..n-1] \Rightarrow a[k] \leq a[k+1])$

{ ...um ciclo. }

Algoritmo:

```
k:= 1;
ordenado:= true;

while   ordenado and (k < n) do
        begin   ordenado:= a[k] <= a[k+1];
                k:= k+1
        end;
```

{ ordenado = $(\forall\ k \in [1..n-1] \Rightarrow a[k] \leq a[k+1])$ }

{ Estará certo? Provar ... }

Mais Regras da Repetição:

- **A Regra do Ciclo repeat:**

$$\frac{\{P\}\ S\ \{I\} \qquad \{I \text{ and not } B\}\ S\ \{I\}}{\{P\} \text{ repeat } S \text{ until } B\ \{I \text{ and } B\}}$$

{ Relacionar com a Regra do while *…. }*

```
{ P }
repeat   S
         { I }
until B;
{ I ∧ B }
```

```
{ P }
S;
{ I }
while   not  B  do
        { I ∧ ¬ B }
        S
        { I }
{ I ∧ B }
```

- **Teorema:**

  [**repeat** S **until** B] ≡ [S; **while not** B **do** S]

- **As Regras do Ciclo for**:

$$\frac{\{x \in [a\,..\,b] \textbf{ and } P([a\,..\,x))\}\ S\ \{P([a\,..\,x])\}}{\{P([\,])\}\ \textbf{for}\ x := a\ \textbf{to}\ b\ \textbf{do}\ S\ \{P([a\,..\,b])\}}$$

$$\frac{\{x \in [a\,..\,b] \textbf{ and } P((x\,..\,b])\}\ S\ \{P([x\,..\,b])\}}{\{P([\,])\}\ \textbf{for}\ x := b\ \textbf{downto}\ a\ \textbf{do}\ S\ \{P([a\,..\,b])\}}$$

{ Também equivalentes ao while *.... }*

{ P([ ]) }

for x:= a **to** b **do**
{ x ∈ [a..b] ∧ P([a..x-1]) }
S
{ P([a..x]) }

{ P([a..b]) }

```
if  a <= b
then  begin  x:= a;
             S;
             while x < b do
                   begin x:=succ(x);
                         S
                   end
      end;
```

{ P([ ]) }

for x:= b **downto** a **do**
{ x ∈ [a..b] ∧ P([x+1..b]) }
S
{ P([x..b]) }

{ P([a..b]) }

```
if  a >= b
then  begin  x:= b;
             S;
             while x > a do
                   begin x:=pred(x);
                         S
                   end
      end;
```

Problema: **Ordenar** um dado vector.

Especificação:

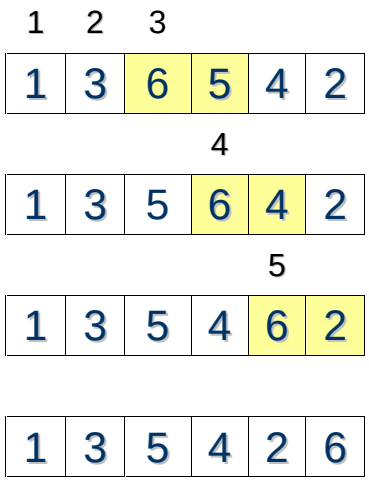
Entrada: $\{ a[1..n] \in \mathbb{Z}^n \}$

Saída: $\{ a[1..n] \in \mathbb{Z}^n : \forall\ k \in [1..n-1] \Rightarrow a[k] \leq a[k+1] \}$

Estratégia: Para $k \in [1..n-1]$
identificar a[k] > a[k+1] e trocar.

```
for  k:= 1 to n-1  do
     if a[k] > a[k+1]
     then  begin  aux:= a[k];
                  a[k]:= a[k+1];
                  a[k+1]:= aux
           end;
```

{ ordenado? }



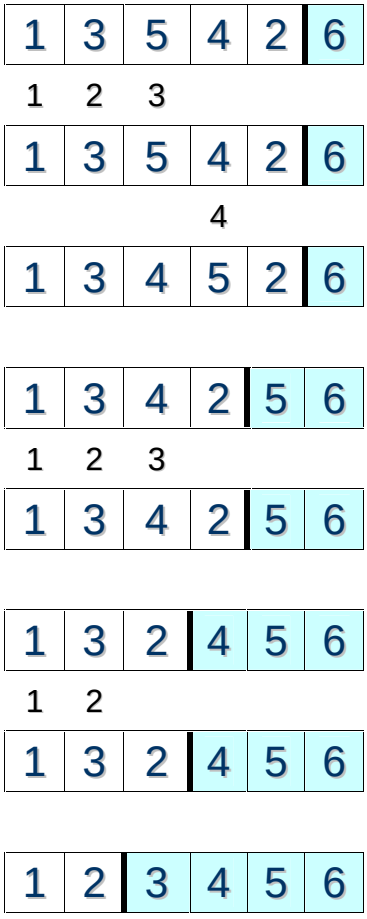
{ Só conseguimos pôr o máximo na última posição. }

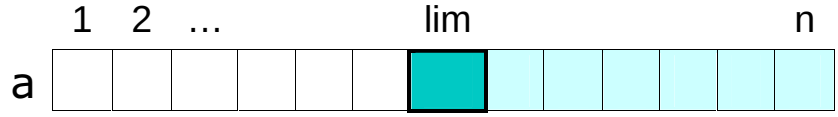
```
for   k:= 1 to n-1   do
      { x[k] = max {x[1..k]} }
      if a[k] > a[k+1]
      then  begin  aux:= a[k];
                   a[k]:= a[k+1];
                   a[k+1]:= aux
            end;
      { x[k+1] = max {x[1..k+1]} }

{ x[n] = max {x[1..n]} }
```

{ … mas o último elemento ficou ordenado! }

É necessário repetir o processo,
para o vector a[1..n-1],
a[1..n-2],
…,
a[1..2]





Algoritmo - BubbleSort:

```
for   lim:= n downto 2 do
      { a[lim+1..n] ordenado }

      for   k:= 1 to lim-1 do
            if a[k] > a[k+1]
            then  begin  aux:= a[k];
                         a[k]:= a[k+1];
                         a[k+1]:= aux
                  end;
      { x[lim] = max {x[1..lim]} }

      { a[lim..n] ordenado }

{ a[2..n] ordenado ∧ x[2] = max {x[1..2]} }

{ a[1..n] ordenado }
```

❒ O Problema da Paragem

{ Existem Sistemas Formais que permitem a Verificação da Correcção Parcial de Algoritmos. }

Especificação de um Algoritmo:

$$\{P\}\ S\ \{Q\}$$

- **Correcção Parcial:** Se S terminar, {P} garante {Q} após a execução de S.

- **Correcção Total:** {P} garante que S termina e também garante {Q} após a execução de S.

{ Mas nenhum Sistema Formal pode permitir a Verificação da Correcção Total de Algoritmos. }

O Problema da Paragem é Indecidível

{ Que podemos fazer? }

- Podemos sempre tentar verificar se **Algoritmos particulares** param ou não.