Erros na programação

A construção de programas traz sempre a produção de erros. Estes erros são de dois tipos: erros de natureza sintáctica e erros de natureza lógica.

Erros sintácticos

Estes resultam do facto de o programador não ter escrito as instruções de acordo com as regras da gramática da linguagem de programação utilizada.

Erros lógicos (erros semânticos)

São erros em geral mais difíceis de detectar do que os erros sintácticos. Estes resultam do facto de o programador não ter expressado correctamente, através da linguagem de programação, a sequência de acções a ser executada (o programador queria dizer uma coisa mas disse outra).

Ao processo de detecção e correcção, tanto de erros sintácticos como dos erros lógicos, dá-se o nome de depuração. Em inglês, este processo é denominado debugging e aos erros que existem num programa chama-se *bugs*.

Na produção do algoritmo em pseudocódigo o erro sintáctico tem pouca importância, sendo essencial a detecção de erros lógicos.

Para o teste dos algoritmos usa-se elaborar uma tabela onde se vão marcando os valores das variáveis e condições do algoritmo a testar. Esta tabela é conhecida por quadro de traçagem (*trace table*).

Estruturas de controlo

Para o controlo do fluxo dos dados num programa, existem estruturas base. Estas condicionam o fluxo dos dados conforme as necessidades do programador, permitindo tomar decisões no decorrer do programa.

A estrutura mais elementar de um programa é o cumprimento sequencial de instruções até ao final do programa. A necessidade de obter outros tipos de controlo levou à criação de mais dois tipos de estruturas as quais vamos designar por estruturas de decisão e estruturas de repetição.

Estruturas de controlo do fluxo de dados:
- sequêncial;
- decisão;
- repetição.

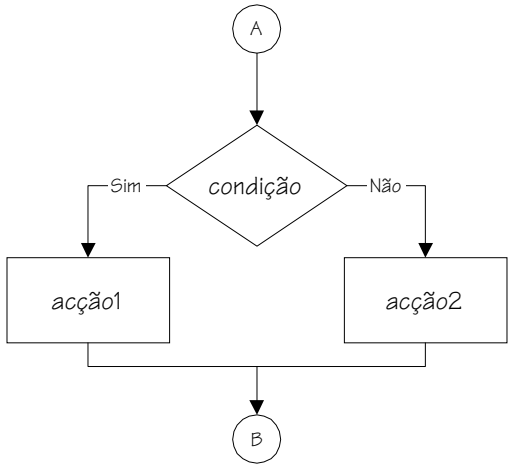
Estruturas de decisão

Estrutura Se

Até agora temos estado a lidar com algoritmos puramente sequênciais, isto é, as instruções são cumpridas uma após outra até chegar ao fim do algoritmo. No entanto existem estruturas que permitem a tomada de decisões em função de condições estabelecidas e assim, o algoritmo apresenta acções alternativas. Estas condições podem ser verdadeiras ou falsas; conforme esse valor o algoritmo segue uma acção ou segue outra em alternativa.

Estas estruturas não são mais que instruções complexas, geralmente, compostas por várias palavras, mas formando um todo.

A primeira que vamos usar é de **decisão simples** e tem o formato




```
Se condição então
   acção1
senão
   acção2
fim se
```

Esta será a sua forma em pseudolinguagem.

Exercícios

1. Faça a discussão da equação do 1º grau

A equação do primeiro grau é do tipo Ax+B=0. Sabe-se, também, que a sua solução é dada pela expressão

$$x = -\frac{B}{A}$$

Temos de analisar esta expressão em função de A *ser igual a zero ou não. Caso seja não se pode determinar. Então, temos:*

```
1. Entrar pelo teclado o valor A e B
2. Verificar para A=0 ou para A<>0
3. Sair para o ecrã o resultado
4. Fim do algoritmo
```

Desenvolvendo o ponto 2, temos:

```
2. Se A=0
   Se B=0 - Sair no ecrã: "Equação indeterminada" (x = -0/0)
   Se B<>0 - Sair no ecrã: "Equação impossível" (x = -B/0)
   Se A<>0
   Calcular o valor de x (x = -B/A)
   Sair no ecrã: x
```

2. Calcule as ajudas de custo a um funcionário de uma empresa

Pretende-se calcular as ajudas de custo a pagar aos funcionários de uma empresa que tenham de se deslocar em viatura própria para serviço da empresa. O valor é em função da distância percorrida e do valor por quilómetro que, por sua vez, é função da motorização do veículo (cilindrada do motor), como se indica a seguir:

- motor de cilindrada inferior a 2000cm^3 será financiado a 3$00 por quilómetro

- motor de cilindrada superior ou igual a 2000cm^3 será financiado a 3$50 por quilómetro.

A fórmula para o cálculo do valor será:

*ajuda de custo = valor por quilómetro * distância percorrida*

3. Calcule o salário semanal de um operário que trabalha por turnos (I)

Pretende-se calcular o salário semanal de um operário em função do turno realizado e do número de horas de trabalho nessa semana.
Sabemos que no turno de dia será pago a 750$00 por hora. No turno da noite será pago a 1000$00 por hora.

4. Calcule o salário semanal de um operário que trabalha por turnos (II)

Pegando no problema anterior, vamos considerar a possibilidade do operário poder fazer horas extraordinárias. Se realizar acima de 35 horas semanais, as restantes horas são consideradas trabalho extraordinário e será pago pelos seguintes valores:

- turno de dia - cada hora extraordinária vale 1000$00;

- turno de noite - cada hora extraordinária vale 1250$00.

5. De entre dois valores dados, verificar qual deles é o maior

O que se pretende é a construção de um algoritmo que permita verificar qual o maior de dois números.
O algoritmo será:

```
1. Entrar pelo teclado os dois valores
2. Compara os dois valores
3. Sair no ecrã o maior
```

O ponto 2 pode ser desenvolvido do seguinte modo:

```
2. Se primeiro valor for maior que o segundo
     Guardar o primeiro valor
   Se segundo valor for maior que o primeiro
     Guardar o segundo valor
```

O algoritmo completo, em pseudolinguagem, é

```
Algoritmo O valor mais elevado I
[Vamos trabalhar só com números inteiros]
Inteiro val1, val2, maior

Início
Entrar val1
Entrar val2
Se val1>val2 então
   maior ← val1
senão
   maior ← val2
fim se
Sair maior
Fim
```

Partindo do algoritmo anterior, propomos ao leitor a construção de um algoritmo para fazer a verificação para três valores.

6. De entre três valores dados, verificar qual deles é o maior

7. De que formas podemos representar um algoritmo?

8. Construa os fluxogramas dos algoritmos dos exercícios anteriores.

9. Implemente os algoritmos anteriores em QuickBasic.

10. Verifique, usando tabela de traçagem, a correcção do algoritmo que construiu para a questão dois.

Para compreender o funcionamento da estrutura SE, vamos usar uma tabela de traçagem para analisar os algoritmos que poderão ser a resposta à questão seis da lista de exercícios anterior.

```
[Verificação do valor mais elevado (versão 2)]
1. Entrar val1, val2, val3
2. Se val1>val2 então
      maior ← val1
3. senão
      maior ← val2
   fim se
4. Se val3>maior então
      maior ← val3
   fim se
```

```
[Verificação do valor mais elevado (versão 1)]
1. Entrar val1, val2, val3
2. Se val1>val2 então
   2.1 Se val1>val3 então
            maior ← val1
   2.2 senão
            maior ← val3
      fim se
3. senão
   3.1 Se val2>val3 então
            maior ← val2
   3.2 senão
            maior ← val3
      fim se
   fim se
```

Para a construção duma trace table começamos por numerar os passos do algoritmo como nos exemplos anteriores. Seguidamente, construímos uma tabela na qual vamos marcando os valores, das variáveis e condições, passo a passo como se o computador estivesse a seguir o algoritmo.
Nos exemplos anteriores, foram realizados os testes para um valor de cada variável, mas devem-se realizar testes utilizando várias situações possíveis que o algoritmo tenha que *enfrentar*.

	val1	val2	val3	val1>val2	senão	val3>maior	maior
1	2	4	5	-	-	-	-
2	2	4	5	F	-	-	-
3	2	4	5	F	V	-	4
4	2	4	5	F	V	V	5

	val1	val2	val3	val1>val2			senão			maior
					val1>val3	senão		val2>val3	senão	
1	2	4	5	-	-	-	-	-	-	-
2	2	4	5	F	-	-	-	-	-	-
3	2	4	5	F	-	-	V	-	-	-
3.1	2	4	5	F	-	-	V	F	-	-
3.2	2	4	5	F	-	-	V	F	V	5

Vejamos o seguinte exemplo:

```
Algoritmo Selecção
[caracterização de algumas letras do alfabeto]
Carácter letra, frase

Início
Entrar letra
Caso letra
   "a","b":
     frase ← "Primeiras letras"
   "k","w","y":
     frase ← "Alfabeto inglês"
   "z":
     frase ← "Última letra"
   Senão
     frase ← "Outras letras do alfabeto"
Fim caso
Sair frase
Fim
```

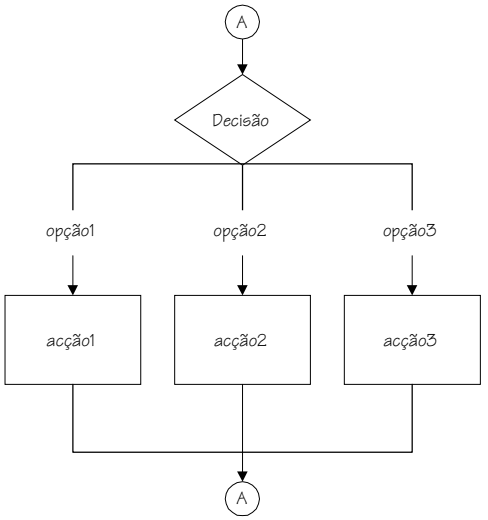
Estrutura Caso

Há casos em que se tem de optar entre várias opções que uma variável pode dispor, pelo que se terá de recorrer a várias estruturas SE encadeadas. O encadeamento das estruturas SE traz dificuldade na interpretação do algoritmo. Neste caso, temos uma estrutura de **decisão composta** ou **decisão múltipla** que vai tornar a leitura do algoritmo mais fácil.

Essa estrutura é conhecida pela estrutura CASO.

Em pseudocódigo, apresentará a seguinte forma:

```
Caso variável
   opção1:
     acção1
   opção2:
     acção2
   ...
Fim caso
```



Exercício

1. Descreva o que faz o seguinte algoritmo.

```
Algoritmo números
Inteiro número
Caracter frase
Início
Entrar número
Se número > 10 então
   frase ← "É maior de 10"
Senão
   Se número < 10 então
      frase ← "É menor de 10"
   Senão
      frase ← "É igual a 10"
   Fim se
Fim se
Sair frase
Fim
```

2. Construa o algoritmo da discussão da equação do segundo grau usando a estrutura CASO.

A equação do segundo grau apresenta-se na forma $ax^2 + bx + c = 0$
A resolução desta equação é baseada na utilização do binómio discriminante $b^2 - 4ac$.
Assim, temos três situações:

- binómio discriminante > 0
Tem solução dada por $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$

- binómio discriminante < 0
Não tem solução nos reais.

-binómio discriminante = 0
Tem solução dada por $x = \frac{-b}{2a}$

3. Construa um algoritmo que disponha do cálculo do valor da área de três figuras geométricas. O utilizador escolherá uma das três figuras disponíveis a tratar e o algoritmo produzirá o valor da área.

A partir de um menu em que são apresentadas as figuras geométricas disponíveis, o utilizador escolherá a pretendida. De seguida, ser-lhe-ão pedidos os dados referentes à figura e produzir-se-á o cálculo do valor da área que será apresentado no ecrã.

4. Construa os fluxogramas dos algoritmos dos exercícios anteriores.

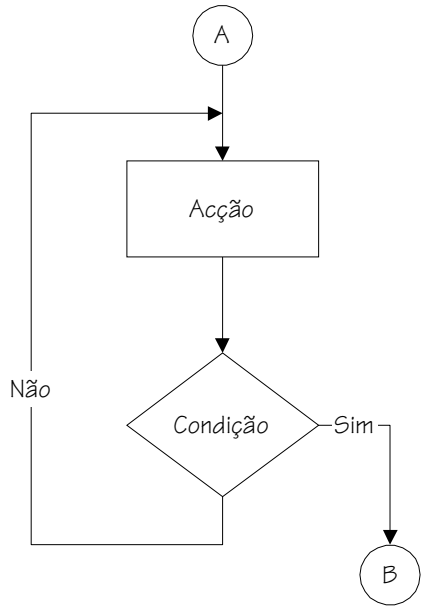
Estruturas de Repetição

Os computadores são particularmente bem adaptados a aplicações nas quais uma dada operação - ou uma série de operações - é repetida muitas vezes, ou seja, a produção de ciclos. As estruturas de repetição são construções fundamentais e podem ocorrer numa variedade de formas.

REPETIR ... ATÉ

Apresentará o seguinte formato:

```
Repetir
   acção
até condição
```



A acção *será repetida até que a* condição *seja verdadeira.*
***A* condição *será testada depois de realizada a* acção**. *Portanto, a* acção *realizar-se-á pelo menos uma vez.*

Apresentamos um exemplo. Trata-se de um algoritmo para o cálculo dos múltiplos de um número até um determinado valor definido por Limite

```
Algoritmo múltiplos
inteiro Número, Limite, Múltiplo

Início
Entrar Número, Limite
Múltiplo ← Número
Repetir

   SAIR Múltiplo
   Múltiplo ← Múltiplo + Número

até Múltiplo >= Limite
Fim
```

ENQUANTO ... REPETIR

Esta apresenta a seguinte formulação em pseudocódigo:

```
Enquanto condição fazer
   acção
repetir
```

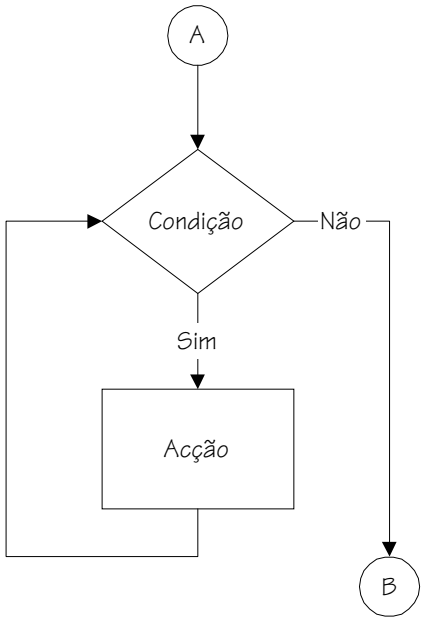
Neste caso, a acção *repete-se enquanto a* condição *for verdadeira.*
***A* condição *é testada antes de se iniciar a* acção**. *Se a* condição *for verdadeira produz-se a* acção*, se for falsa já não se produz a* acção.

A estrutura REPETIR ... ATÉ pode ser realizada utilizando a estrutura ENQUANTO ... REPETIR. Basta mudar de **REPETIR ... ATÉ *condição***, para **ENQUANTO ~*condição* ... REPETIR**.

Assim, temos:

```
Múltiplo ← Valor
Repetir

   Sair Múltiplo
   Múltiplo ← Múltiplo + Valor

até Múltiplo >= Número
```

```
Múltiplo ← Valor
Enquanto ~(Múltiplo >= Número) fazer

   Sair Múltiplo
   Múltiplo ← Múltiplo + Valor

repetir
```



```
Algoritmo Números Primos
[Vamos verificar se é primo]
inteiro número, contagem
carácter frase
booleano primo

Início
primo ← verdadeiro
Entrar numero

[Para não ser primo tem de ser divisível
por qualquer entre 1 e ele próprio, isto é,
de 2 até número-1]
Para contagem ← 2 até número-1 fazer

   [Se for divisível não é primo]
   Se (numero mod contagem = 0) então
      primo ← falso
   fim se

Seguinte
Se primo então
   frase ← "É número primo"
senão
   frase ← "Não é primo"
fim se
Sair frase
Fim
```

PARA... SEGUINTE

Por último, temos uma estrutura de repetição por contagem:

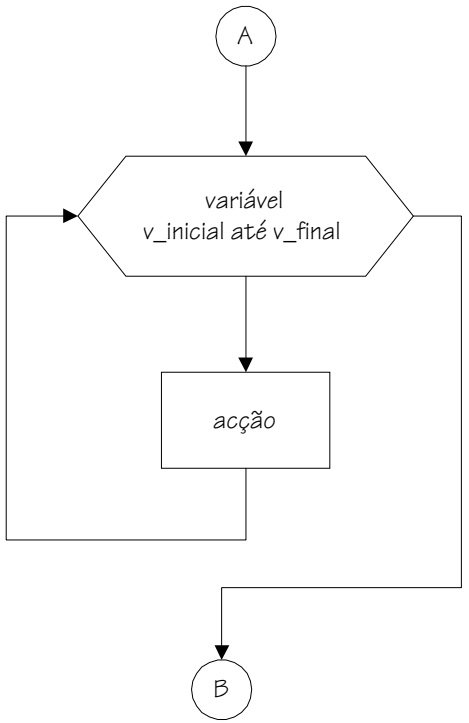
Para variável ← *valor inicial* **até** *valor final* **passo** *n* **fazer**
acção
seguinte

A variável toma um valor inicial que vai incrementando (ou decrementando) até um valor final. O valor do incremento (ou decremento) é dado pelo valor positivo (ou negativo) de *n*.

O passo é de utilização opcional. Se não for declarado é, por defeito, de uma unidade.

Vamos ver o algoritmo para verificar se um número é ou não primo.

Não está estabelecida qualquer representação gráfica para a estrutura PARA, mas podemos adoptar a seguinte.

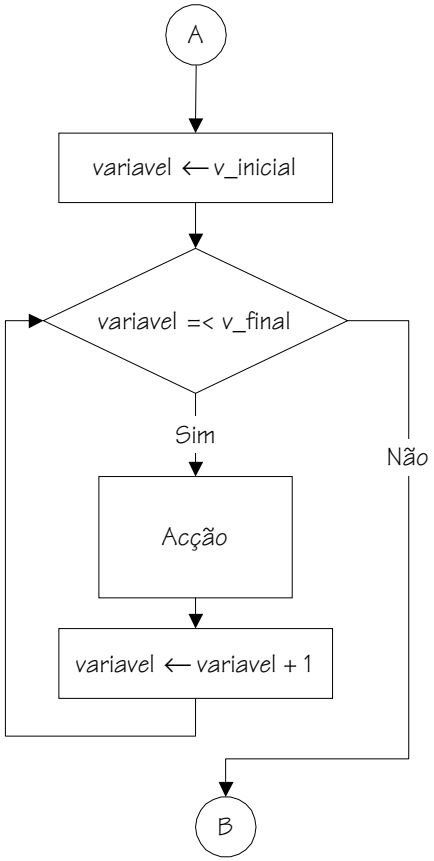


Aqui é apresentada uma aplicação desta estrutura. A variável *contagem* é utilizada como contador. A não indicação do passo leva a que o valor seguinte de *contagem* será o valor anterior mais um.

Esta estrutura PARA ... pode-se considerar uma variante da estrutura ENQUANTO... Vejamos:

Neste caso é necessária a inicialização da variável contador antes da estrutura ENQUANTO.

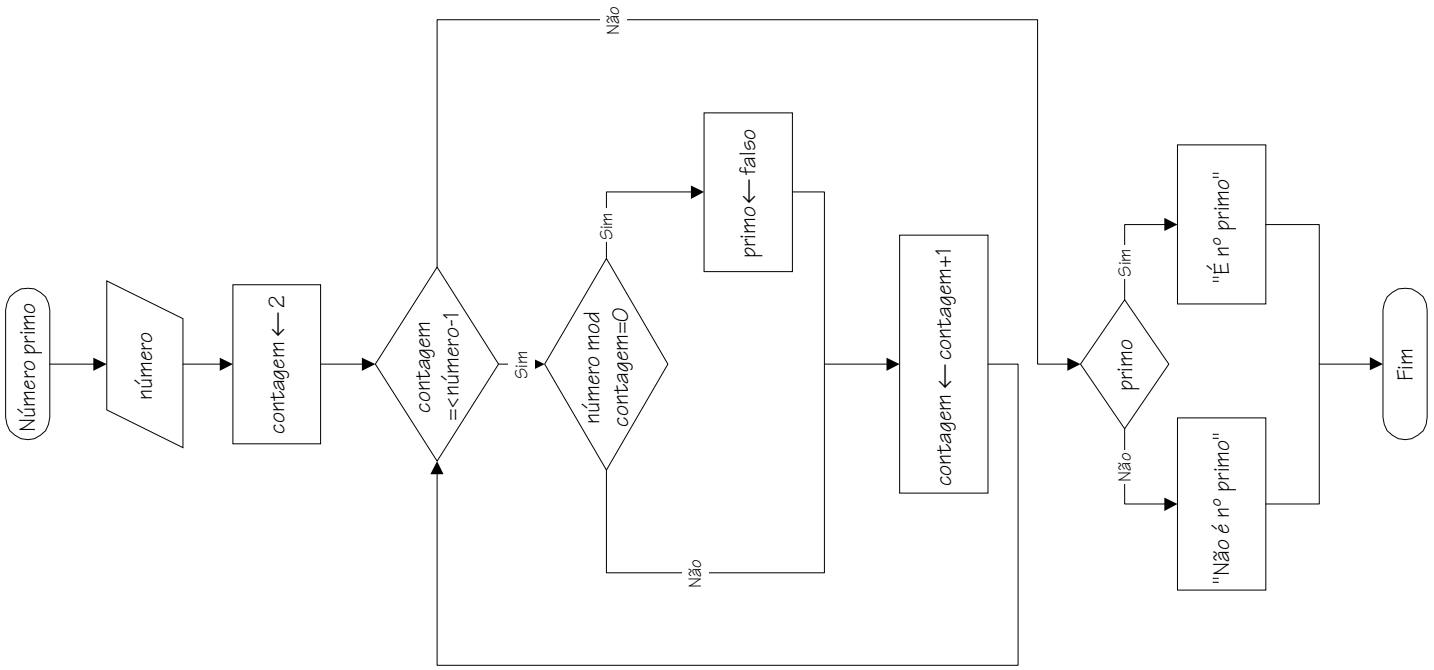
Dentro do ciclo, é necessário proceder ao incremento da variável contador para que alcance o valor final pretendido.




```
Algoritmo Números Primos
[Vamos verificar se é primo]
inteiro número, contagem
booleano primo

Início
primo ← verdadeiro
Entrar numero

[Para não ser primo tem de ser divisível
por qualquer entre 1 e ele próprio, isto é,
de 2 até número-1]
contagem ← 2 [Para iniciar a contagem]
Enquanto contagem =< número-1 fazer

   [Se for divisível não é primo]
   Se (numero mod contagem = 0) então
     primo ← falso
   Fim se
   contagem ← contagem + 1 [Incrementa de uma unidade]

Repetir
Se primo então
   Sair "É número primo"
senão
   Sair "Não é primo"
Fim se
Fim
```

Usando a estrutura ENQUANTO, o fluxograma fica como o apresentado ao lado.

Fluxograma do algoritmo *Número primo* usando a estrutura ENQUATO.



Exercícios

1. Análise do algoritmo do número primo.

Um número primo é aquele que é divisível, só e somente, pela unidade e por ele próprio. Sendo divisível por qualquer outro, não é número primo.
Como todos os números são divisíveis por um e por si próprios não necessitamos de verificar a divisibilidade. Assim, temos de verificar todos os outros números desde 2 até ao número anterior ao que queremos verificar (número - 1). Por isso aparece a estrutura

```
Para contagem ← 2 até número-1 fazer

   acção

Seguinte
```

A acção é a verificação da divisibilidade.
Se um número a *for divisível por outro* b *então o resto da divisão será zero. Temos uma operação que nos dá o resto da divisão de inteiros. É o* ***mod****.*
Neste caso temos que se número ***mod*** contagem = *0 então o* número *não será primo. Daí que surge*

```
Para contagem ← 2 até número-1 fazer

   Se (numero mod contagem = 0) então
     primo ← falso
   Fim se

Seguinte
```

Na realidade estamos a verificar se o número não é primo. Para indicar se o número é ou não primo usamos uma variável booleana que toma um de dois valores: ou verdadeiro ou falso. A variável está identificada com o nome primo. Por isso usamos

```
   Se (numero mod contagem = 0) então
     primo ← falso
   Fim se.
```

Para mandar a indicação para o ecrã, a frase será a conveniente, isto é, se o número é primo ou não. Usamos o valor da variável booleana para fazer a diferenciação.

```
Se primo então
   frase ← "É número primo"
senão
   frase ← "Não é primo"
Fim se
```

*Se for primo (*primo *terá o valor verdadeiro) é apresentado no ecrã "É número primo"; se não for será apresentada a frase "Não é primo".*

2. Somar os N primeiros números inteiros

Somar todos os números desde 1 até ao número N indicado pelo utilizador.
Faça o algoritmo com a estrutura PARA e outro com a estrutura ENQUANTO.

3. Construa o algoritmo do factorial de um número, isto é, o produto de todos os inteiros desde 1 até ao número escolhido

Por exemplo, o factorial de 3 é 1x2x3 = 6, mas o factorial de zero é 1.
Use a estrutura PARA, mas não se esqueça de ter em atenção do pormenor de o factorial de zero ser 1.

4. Implemente o algoritmo do número primo para apresentar todos os números primos até um número N indicado pelo utilizador.

Terá de usar mais uma estrutura PARA que a usada no algoritmo apresentado.

5. Verifique o algoritmo do número primo usando uma tabela de traçagem

6. Implemente os algoritmos dos exercícios usando o QuickBasic.

7. O algoritmo apresentado seguidamente produz um ciclo infinito. Corrija-o.

```
contagem ← 1
Enquanto contagem <= 10 fazer

   Sair "Não acabou"

Repetir
```

Use a técnica de trace table para analisar o funcionamento do algoritmo

8. Construa um algoritmo para, de entre dois números inteiros, verificar qual deles é o maior

Para a construção do algoritmo pedido basta usar uma estrutura SE.

```
Algoritmo valor mais elevado
inteiro val1, val2, maior
Início
Entrar val1, val2
Se val1>val2 então
   maior ← val1
senão
   maior ← val2
Fim se
Sair maior
Fim
```

9. Construa um algoritmo para verificar de entre três números inteiros, qual o maior

Será aconselhável basear-se no algoritmo do exercício anterior.

10. Melhoria do algoritmo "Números primos"

O algoritmo apresentado antes pode ser melhorado em dois aspectos:
- Não tem interesse a verificação a números negativos, nem o zero nem o número um.
Solução: impedir ao utilizador a introdução dos referidos valores.
- Quando se encontra um valor em que o número é divisível não há vantagem em continuar a verificação, pois o número é primo.
Solução: quando se encontra o valor que divide o número pára-se a estrutura PARA.
Vamos começar pelo primeiro aspecto

```
Algoritmo Números Primos
inteiro número, contagem
carácter frase
booleano primo

Início
primo ← verdadeiro
número ← 0
Enquanto número < 2 fazer
   Entrar número
Repetir
.......
```

Acrescentamos a estrutura ENQUANTO para verificar o valor que o utilizador introduz na variável número*, obrigando o utilizador a repetir novo valor caso tenha introduzido um valor não apropriado.*
Há a inicialização da variável número (número ← 0) para obrigar ao pedido da introdução do número a verificar. Caso não se faça a inicialização referida, não sabemos o que poderá acontecer uma vez que não sabemos que valor a variável número contém no momento do teste da condição da estrutura ENQUANTO.
Para o segundo aspecto referido, temos

```
Para (contagem ← 2 até número-1) ∧ ~(numero mod contagem=0) fazer

   Se (numero mod contagem = 0) então
     primo ← falso
   Fim se

Seguinte

.......
```

Vimos nos operadores lógicos que na conjunção *basta uma proposição ser falsa para que a condição seja falsa. Assim, basta encontrar um valor que divida o número para acabar a estrutura PARA. Esta, também acaba se a contagem chegar ao fim.*
Vejamos a tabela de verdade para as condições postas.
Tomemos ***a*** *= contagem ← 2* ***até*** *número-1*
b *= numero* ***mod*** *contagem=0*

a	b	~b	a ∧ ~b
1	1	0	0
1	0	1	1
0	1	0	0
0	0	1	0

A estrutura PARA continua

11. Como exercício, transforme a estrutura PARA, do algoritmo anterior, numa estrutura ENQUANTO. Valide a sua escolha com a verificação numa *trace table*.

Linguagens de programação

Os computadores comunicam em código binário, isto é, os programas e os dados têm de ser em código binário para que a máquina os possa entender. A esta linguagem dá-se o nome de *linguagem máquina*.

Esta foi a linguagem usada nos primeiros computadores e só especialistas poderiam programar essas máquinas.

Para facilitar o trabalho de programação, foram desenvolvidos programas tradutores que transformavam um código feito numa linguagem mais compreensível por humanos, em linguagem máquina. Esses programas eram conhecidos como assembladores (do inglês assembler) que traduziam uma linguagem mais próxima da humana para a linguagem máquina.

Posteriormente, surgiram programas que usavam linguagens mais desenvolvidas e mais próximas da linguagem falada (em geral inglês) e independentes da estrutura da máquina. Estas linguagens são conhecidas como *linguagens de alto nível*, pois permitem que o programador não necessite de ter algum conhecimento sobre o funcinamento interno do computador.

De uma forma genérica, podemos dizer que há duas maneiras destes programas fazerem atradução para a linguagem máquina: temos os compiladores que produzem um programa em linguagem máquina e que, depois, pode ser usado; por outro lado, temos os interpretadores que vão traduzindo o texto escrito na linguagem de alto nível enquanto este é executado.

A construção de um programa compilado implica a criação de um *programa fonte* que é o texto na linguagem de alto nível. Depois, este é compilado produzindo um programa em linguagem máquina chamado *programa objecto*.

Este programa objecto não é legível para o programador, pelo que terá de recorrer sempre ao programa fonte, este sim, texto legível pelo programador.

Para exemplo, temos o QuickBASIC, ao qual o leitor terá fácil acesso através do conhecido MS-DOS, trata-se de uma linguagem de alto nível interpretada. No entanto, o TurboPascal é compilado.

No início da unidade vimos que a criação de um programa passa pela construção do algoritmo do problema em estudo e, depois, tendo em conta o tipo de problema tratado, escolhe-se a linguagem conveniente e procede-se à fase de implementação para que possa realmenter ser usado num computador.