

Visual Basic

I – Introdução à Programação e ao Visual Basic



Universidade
do Porto

Faculdade de
Engenharia

FEUP

CFAC – Conceção e Fabrico Assistidos
por Computador

João Manuel R. S. Tavares

DEMEC
DEPARTAMENTO DE ENGENHARIA MECÂNICA

Sumário

1. Ciclo de desenvolvimento de um programa;
2. Descrição de algoritmos;
3. Desenvolvimento modular de programas;
4. Estruturas de controlo de um programa;
5. Introdução ao Visual Basic: Controlos, Métodos, Eventos, Ambiente de Programação.

Execução de uma tarefa no computador

Passos até escrever as instruções (codificação) para executar uma determinada tarefa:

- 1 - Determinar qual deve ser a saída;
- 2 - Identificar os dados, ou entrada, necessária para obter a saída;
- 3 - Determinar como processar a entrada para obter a saída desejada.



Execução de uma tarefa no computador

Exemplos de execução de tarefas:

1 - Um exemplo do dia a dia: fazer um bolo de maçã

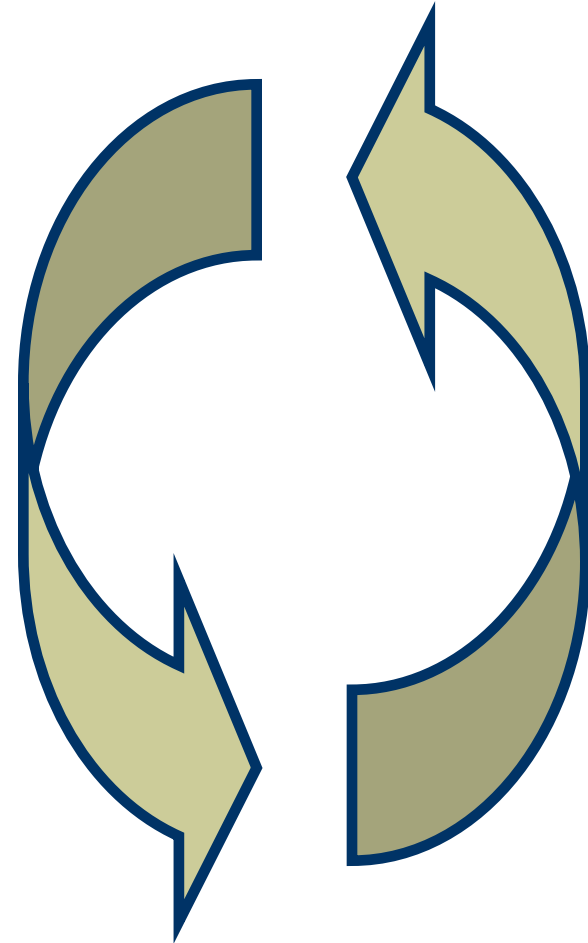
- *Saída*: bolo de maçã;
- *Entrada*: ingredientes e respectivas quantidades;
 - Os ingredientes e quantidades são determinados por aquilo que se quer fazer;
- *Processamento*: a receita indica como proceder.

2 - Um problema de cálculo: determinar o valor do selo de uma carta

- *Saída*: valor do selo;
- *Entrada*: peso da carta, escalões de peso, custo/escalão;
- *Processamento*: o algoritmo indica como proceder.

Ciclo de desenvolvimento de um programa

1. Analisar o problema;
2. Planear a solução;
3. Escolher a interface;
4. Codificar;
5. Testar e corrigir erros;
6. Completar a documentação.



1 - Analisar o problema

- ◆ Compreender o que o programa deve fazer, qual deve ser a saída;
- ◆ Ter uma ideia clara de que dados (entrada) são fornecidos;
- ◆ Perceber muito bem qual a relação entre a entrada e a saída desejada.

2 - Planejar a solução

- ◆ Encontrar uma sequência lógica e precisa de passos para resolver o problema.
 - Tal sequência de passos é chamada um algoritmo;
 - O algoritmo deve incluir todos os passos, mesmo aqueles que parecem óbvios;
 - Existem vários métodos de especificar o algoritmo:
 - ◆ diagramas de fluxo ou fluxogramas;
 - ◆ pseudocódigo;
 - ◆ diagramas *top-down*.
- ◆ O planejamento também envolve um teste “manual” do algoritmo, usando dados representativos.

3 - Escolher a interface

- ◆ Determinar como é que a entrada será obtida (dados de entrada) e como é que a saída será apresentada (resultados).
- ◆ Por exemplo, em *Visual Basic*:
 - Criar objectos para receber a entrada e apresentar a saída;
 - Criar botões de comando apropriados para que o utilizador possa controlar o programa (eventos).

4 - Codificar

- ◆ Traduzir o algoritmo para uma linguagem de programação (ex.: *Visual Basic*) obtendo-se assim o programa pretendido;
- ◆ Introduzir o programa no computador.

5 - Testar o programa e corrigir erros (*debugging* / depuração)

◆ Localizar e remover eventuais erros do programa:

- Os erros sintácticos resultam do facto de o utilizador não ter escrito o programa de acordo com as regras da gramática da linguagem de programação utilizada; são detectados pelo compilador/interpretador da linguagem.

O computador não executará nenhuma instrução sintacticamente incorrecta.

- Os erros semânticos resultam do facto de o programador não ter expressado correctamente, através da linguagem de programação, a sequência de acções a ser executada.

Estes erros têm de ser detectados pelo programador através de testes exaustivos com dados variados para os quais a saída é perfeitamente conhecida.

6 - Completar a documentação

- ◆ A documentação serve para que outra pessoa ou o próprio programador, mais tarde, entenda o programa.
- ◆ A documentação consiste em incluir comentários no programa que especificam:
 - o objectivo do programa;
 - como usar o programa;
 - a função das variáveis;
 - a natureza dos dados guardados nos ficheiros;
 - as tarefas executadas em certas partes do programa;
 - ...
- ◆ Em programas comerciais, a documentação inclui, normalmente, um manual de instruções.
- ◆ Outros tipos de documentação: fluxograma, pseudocódigo, diagrama *top-down*.

Descrição de algoritmos

◆ Duas formas utilizadas:

Pseudocódigo

Descreve a sequência de passos usando uma linguagem parecida com a linguagem comum (Inglês, Português, ...) mas usando frases com construções próximas das que são aceites por muitas linguagens de programação.





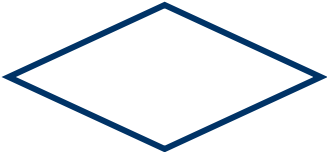
Exemplos de construções:

- 1 - Se condição então fazer acções senão fazer acções
- 2 - Repetir acções até que condição

Fluxograma ou diagrama de fluxo

Descreve graficamente a sequência de passos a executar para resolver um determinado problema e como os passos estão interligados. É constituído por um conjunto de símbolos geométricos ligados por setas.

Símbolos ANSI usados em fluxogramas

	Linha de fluxo	- usado para ligar os outros símbolos indicando a sequência de operações
	Terminal	- usado para representar o início ou o fim de uma tarefa
	Entrada/Saída	- usado para operações de entrada/saída tais como ler ou imprimir (os dados a ler/escrever são indicados no interior)
	Processamento	- usado para operações de manipulação dos dados ou operações aritméticas
	Decisão	- usado para indicar operações de teste; tem uma entrada e duas saídas correspondentes ao resultado do teste ser verdadeiro ou falso

Símbolos ANSI usados em fluxogramas



**Processo
pré-definido**

- usado para representar um grupo de operações que constituem uma tarefa



Conector

- usado para ligar diferentes linhas de fluxo



**Conector para
fora da página**

- usado para indicar que o fluxograma continua noutra página



Comentário

- usado para fornecer informação adicional acerca de outro símbolo do fluxograma

Exemplo - Pseudocódigo

PROBLEMA:

Calcular as raízes reais de uma equação do 2º grau.

equação : $Ax^2+Bx+C = 0$

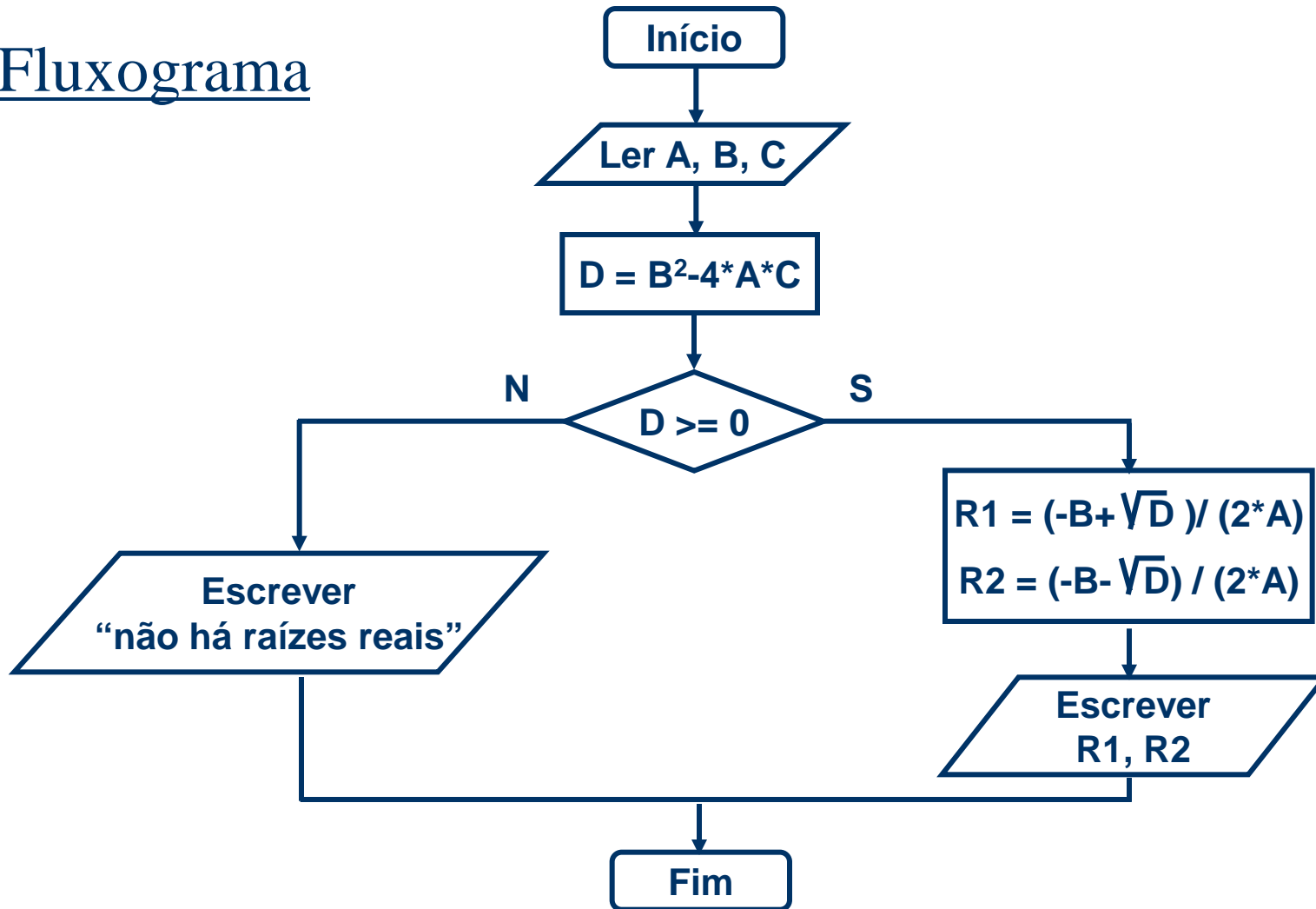
raízes : $x = (-B \pm \sqrt{B^2 - 4AC}) / (2A)$

Pseudocódigo

- Ler (A, B, C) ;
- Calcular $D = B^2 - 4 * A * C$;
- Se $D \geq 0$ então
 - { Calcular $R1 = (-B + \sqrt{D}) / (2 * A)$;
 - Calcular $R2 = (-B - \sqrt{D}) / (2 * A)$;
 - Escrever (R1, R2) ; }
- Senão
 - Escrever (“não tem raízes reais”) ;
- Fim

Exemplo - Fluxograma

Fluxograma



Descrição da estrutura de um programa

- ◆ A estrutura de um programa pode ser descrita através de um diagrama de estrutura, diagrama hierárquico ou diagrama *top-down* que descreve a organização do programa, mas omite os pormenores das operações;
- ◆ Ele descreve o que cada parte ou módulo do programa faz e mostra como os diferentes módulos estão relacionados entre si;
- ◆ O diagrama lê-se do topo para baixo (*top-down*) e da esquerda para a direita;
- ◆ Cada módulo pode estar dividido em submódulos e assim sucessivamente;
- ◆ Estes diagramas são úteis no planeamento inicial do programa e ajudam a escrever programas bem estruturados.

Desenvolvimento modular de programas

- ◆ Método usado para lidar com problemas de programação complexos;
- ◆ Começa-se por dividir a tarefa inicial em sub-tarefas algumas das quais poderão ser de grande complexidade;
- ◆ Cada uma destas sub-tarefas é, por sua vez, dividida em sub-tarefas mais simples e assim sucessivamente, até que todas as tarefas estejam descritas de forma suficientemente elementar para poderem ser facilmente codificadas na linguagem de programação escolhida;
- ◆ Vantagens do desenvolvimento modular:
 - um módulo pode ser facilmente reutilizado;
 - facilita a detecção e correção de erros (analisando os sintomas de um erro é mais fácil reduzir a causa desse erro a um determinado módulo).

Exemplo - Diagrama top-down

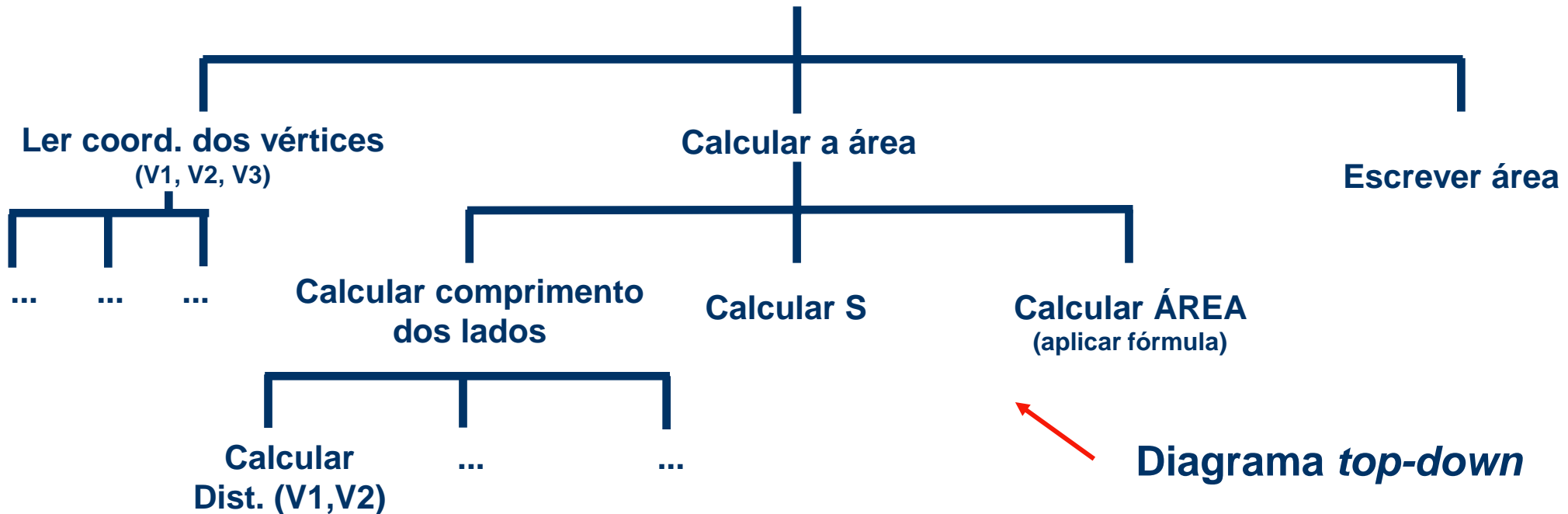
PROBLEMA:

Calcular a área de um triângulo, dadas as coordenadas dos vértices, usando a fórmula de Heron :

$$\text{AREA} = S * (S-A) * (S-B) * (S-C)$$

em que $S = \text{semiperímetro} = (A + B + C) / 2$

A, B e C = comprimentos dos lados



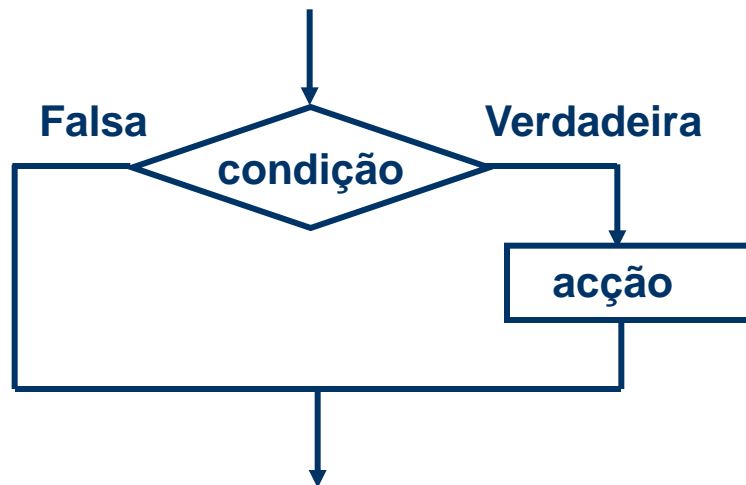
Estruturas de controlo de um programa

- ◆ Em geral, um programa não é constituído por uma sequência rígida, linear, de instruções que são executadas sempre do mesmo modo.
- ◆ Muitas problemas requerem que seja tomada uma decisão para seleccionar entre duas sequências de instruções qual a que vai ser executada.
- ◆ Por vezes, é necessário repetir um determinado conjunto de instruções enquanto se verificar uma determinada condição, até que se verifique uma determinada condição, ou um determinado número de vezes.
- ◆ A generalidade das linguagens de programação possui além de instruções simples de leitura, escrita e atribuição de valores instruções de controlo que envolvem acções de selecção ou de repetição de sequências de instruções, permitindo “fugir” a uma sequência rígida, linear, de execução de um programa.

Instruções condicionais

Permitem uma selecção de sequências alternativas de instruções.

Fluxograma



Pseudocódigo

Se *condição* então *acção*

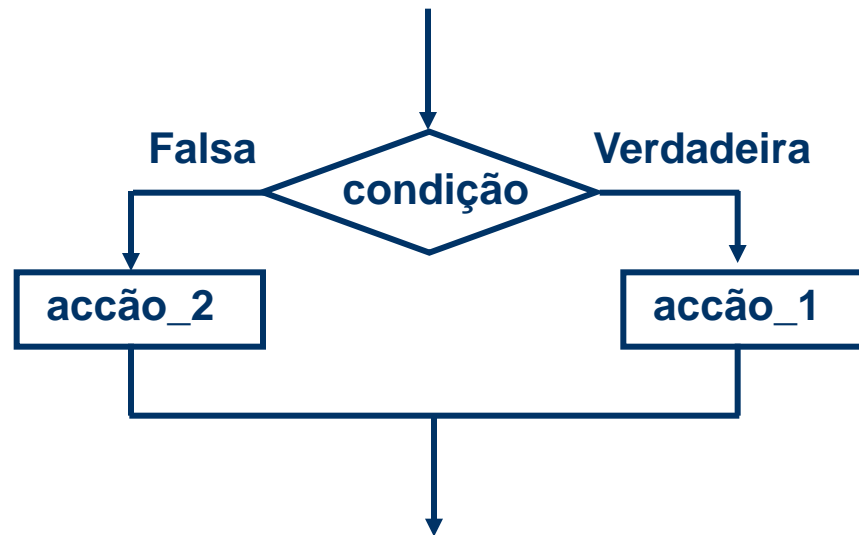
Se a condição for verdadeira a acção é executada.

Se a condição for falsa não é executada nenhuma acção, passando a ser executada a instrução seguinte.

Uma acção pode ser constituída por uma ou mais instruções.

Instruções condicionais

Fluxograma



Pseudocódigo

Se *condição* então
acção_1
senão
acção_2

Se a condição for verdadeira é executada a acção_1.

Se a condição for falsa é executada a acção_2.

Uma acção pode ser constituída por uma ou mais instruções.

Instruções de repetição

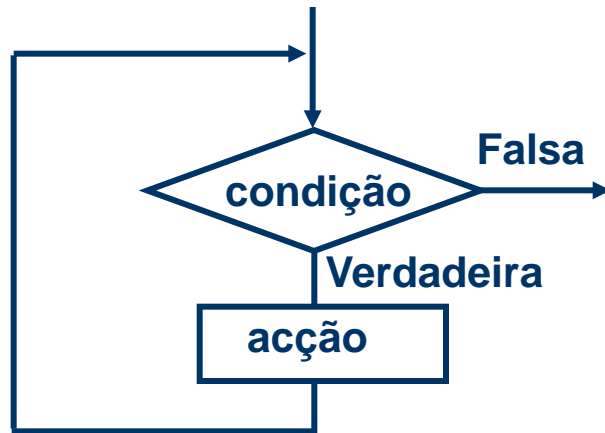
Usadas quando se pretende executar uma sequência de instruções zero ou mais vezes.

Há 3 variantes de instruções repetitivas:

- 1) Enquanto se verificar uma condição executar uma acção;
- 2) Repetir uma acção até que se verifique uma dada condição;
- 3) Executar uma acção um certo número de vezes.

Instruções de repetição

Fluxograma



Pseudocódigo

Enquanto *condição* executar
acção

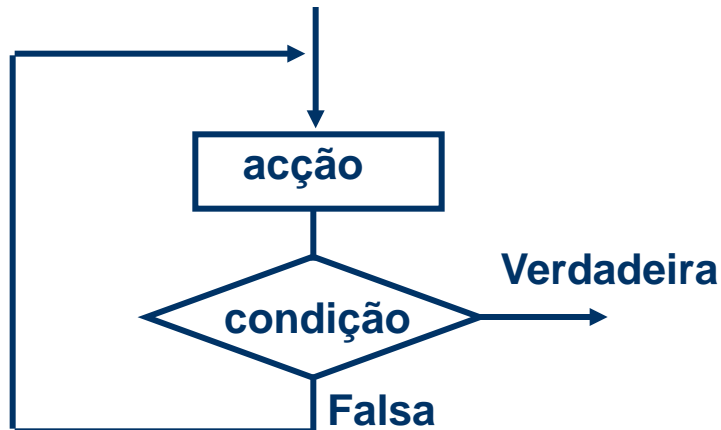
Se a condição for verdadeira é executada a acção e volta-se a testar a condição.

Se a condição for falsa passa-se à execução da instrução seguinte.

A acção pode ser executada zero (se na 1ª vez o teste de condição resultar logo em falso) ou mais vezes.

Instruções de repetição

Fluxograma



Pseudocódigo

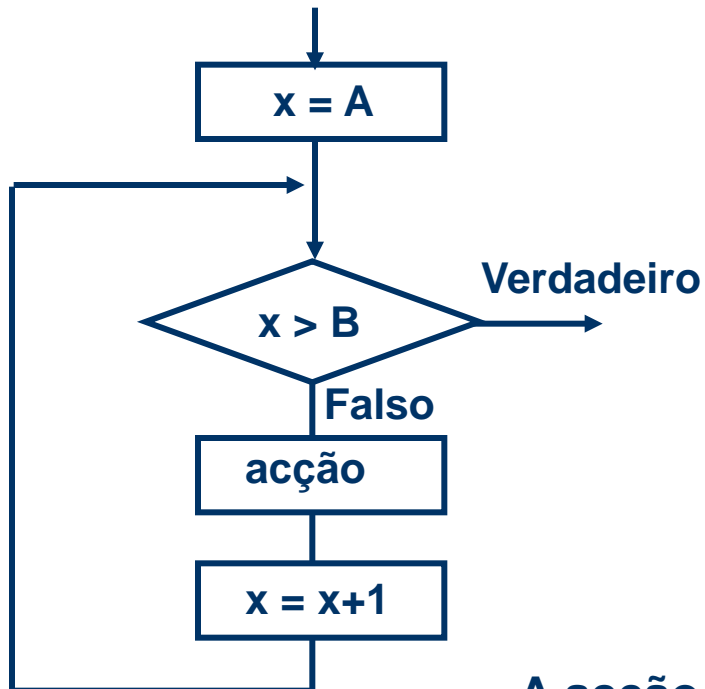
Repetir
acção
até que *condição*

**A acção é executada e, a seguir, testa-se a condição.
Se a condição for falsa a acção é repetida e volta-se a testar a condição.
Se a condição for verdadeira passa-se à execução da instrução seguinte.**

A acção pode ser executada uma (se o teste de condição resultar verdadeiro logo na 1ª vez) ou mais vezes.

Instruções de repetição

Fluxograma



Pseudocódigo

Para x de A até B executar
acção

A acção é executada um certo número de vezes, desde um valor inicial (*A*) até um valor final (*B*) de uma variável (*x*), designada *contador*, que controla o ciclo.

Se $A > B$ o ciclo não é executado nenhuma vez.

Introdução ao *Visual Basic*

- ◆ BASIC significa “*Beginner’s All-purpose Symbolic Instruction Code*”;
- ◆ Desenvolvida por John Kemeny e Thomas Kurtz nos anos 1960;
- ◆ *Visual Basic* é desenvolvida pela *Microsoft Corporation* desde 1991 (versão 1.0);
- ◆ Linguagem utilizada para criar aplicações para ambientes *Microsoft Windows*;
- ◆ Disponibiliza a criação de uma interface gráfica entre a aplicação computacional (programa) e o seu utilizador (*GUI - Graphical User Interface*).

Introdução ao *Visual Basic*

- ◆ O *Visual Basic* é uma linguagem “guiada por eventos” usada para desenvolver aplicações que correm em ambientes *Microsoft Windows*.
- ◆ *Linguagens de programação tradicionais (procedural - “procedimental”)*:
 - É o programa que especifica a sequência de todas as operações a executar.
Exemplos: Pascal, C, Fortran
- ◆ *Linguagens “guiadas por eventos” (event-driven)*:
 - Não existe uma sequência pré-determinada de execução do código do programa;
 - É o utilizador que, pressionando teclas ou clicando em botões e janelas desencadeia (eventos) a execução de procedimentos (conjuntos de instruções) que executam certas tarefas.

Exemplo

Programa para ler dois números e apresentar a sua soma no écran

Em *TURBO PASCAL* (ling. "procedimental")

```
Program Somador;  
Var  
  Op1, Op2, Soma: Integer;  
Begin  
  Write('1º operando ? '); Readln(Op1);  
  Write('2º operando ? '); Readln(Op2);  
  Soma := Op1 + Op2;  
  Write('Resultado da soma = ', Soma);  
End.
```

Exemplo de execução:

```
1º operando ? 4  
2º operando ? 7  
Resultado da soma = 11
```

- ◆ A ordem de introdução dos dados é fixada durante a escrita do programa;
- ◆ O utilizador do programa não tem controle sobre isso;
- ◆ Por cada soma a efectuar é necessário introduzir os dois operandos.

Exemplo

Em *VISUAL BASIC* (ling. “guiada por eventos”)

- ◆ A ordem de introdução dos dados é escolhida pelo utilizador;
- ◆ A soma é efectuada quando o utilizador clicar no botão “Somar”;
- ◆ Depois de efectuar uma soma, o utilizador pode alterar apenas um dos operandos e clicar novamente em “Somar”.



Introdução ao *Visual Basic*

- ◆ Em *Visual Basic* trabalha-se com objectos;
- ◆ Os objectos têm propriedades e métodos associados e podem reconhecer determinados eventos;
- ◆ Objecto:
 - Características:
 - São os blocos constituintes do *Visual Basic*;
 - Os tipos de objectos que é possível utilizar já estão definidos. O programador cria instâncias desses objectos;
 - É possível interactuar com os objectos.

Introdução ao *Visual Basic*

■ Um objecto tem:

● Propriedades

- ◆ as características individuais do objecto.

● Métodos

- ◆ comandos que o objecto pode executar.

● Eventos associados

- ◆ estímulos do utilizador, do ambiente ou de outros objectos a que o objecto pode responder.

Propriedades

Características

- Propriedades são variáveis associadas a um objecto;
- Cada objecto tem uma lista diferente de propriedades;
- Algumas propriedades são comuns a muitos objectos;
 - ex: Visible
- Outras são específicas de um certo tipo de objecto;
 - ex: Interval, num temporizador (*Timer*).
- Podem ter vários tipos de valores diferentes:
 - Texto (*Caption*)
 - Numérico (*Height*)
 - Booleano - True/False (*Visible*)
- As propriedades são usadas para modificar o aspecto dos objectos (ex: *BackColor*) ou para modificar o seu comportamento.
 - ex: *Enabled*, determina se um objecto pode responder a eventos.

Propriedades

- ◆ As propriedades podem ser alteradas
 - Durante o desenho da interface
 - ◆ usando a caixa de propriedades
 - Durante a execução do programa
 - ◆ incluindo instruções adequadas no código, do tipo:

Let objectName.property = setting

Propriedades

Exemplos:

```
Let Text1.Text = ""
```

```
Let Text1.Font1.Size = 12
```

```
Let Text1.Font.Bold = True
```

```
Let Text1.ForeColor = &HFF&
```

```
Let Label1.Caption = "Hello"
```

```
Let Label2.BorderStyle = 2
```

Propriedades

Notas:

- O formulário é o objecto por defeito:

```
Let Form1.property = setting
```

é equivalente a

```
Let property = setting
```

- Com as propriedades Caption, Text e Font.Name o valor atribuído (*setting*) deve estar entre aspas:

Exemplo:

```
Let Form1.Caption = "Somador"  
Let Text1.Font.Name = "Courier"
```

Métodos

◆ *Características:*

- Um tipo especial de comando associado a um objecto;
- Só funciona com o objecto a que estiver associado;
- Alguns objectos têm poucos métodos associados:
 - ex: *Check boxes*
- Outros têm muitos métodos associados:
 - ex: *Text boxes*
- Só são activados durante a execução do programa.

Métodos

◆ *Sintaxe:*

objectName.method (executa *method* sobre *object*)

Exemplo:

object.SetFocus (“foca” o objecto)

object.Print (escreve no objecto)

object.Cls (limpa o conteúdo do objecto, por exemplo texto ou gráficos)

Eventos

◆ *Características:*

- Os estímulos ou acções que um objecto reconhece;
 - ex: um clique no rato ou o carregamento de uma tecla.
- Listados no menu *Proc* da janela de código;
- Alguns eventos são reconhecidos por quase todos os objectos;
 - ex: *Click*.
- Outros são específicos de certos objectos;
 - ex: do *Timer*.
- Podem ser activados pelo utilizador;
 - ex: *Click*.
- Ou pelo ambiente (por código);
 - ex: *Load, Timer*.

Concepção de programas “guiados por eventos” (*event-driven*)

- ◆ A maior parte do código em *Visual Basic* está associado a relações “objecto-evento”.

- ex: *Picture1_Click*

- Leia-se “quando o utilizador clicar no controle *Picture1*, acontecerá o seguinte ...”

- ◆ O utilizador ou o ambiente geram eventos os quais, por sua vez, desencadeiam a execução de código associado ao objecto que “recebeu” o evento.

Compete ao programador escrever este código.

- ◆ O *Visual Basic* ignora os eventos que não tenham nenhum procedimento (código) associado.

Etapas de desenvolvimento de um programa

◆ Planeamento:

- Desenhar um esquema da interface:
seleccionar os objectos, a sua posição e tamanho.
- Fixar as propriedades relevantes dos objectos.
Para cada objecto, elaborar uma lista das propriedades a alterar e dos valores dessas propriedades.

● *Exemplo:*

<u>Objecto</u>	<u>Propriedade</u>	<u>Valor</u>
Form1	Caption	Somador
Text1	Text	(blank)
Command1	Caption	Terminar
Command1	Font	Arial

Etapas de desenvolvimento de um programa

◆ ...Planeamento

- Escrever o código a executar quando ocorrerem os eventos:
 - Determinar que eventos requerem que sejam executadas acções;
 - Planear, passo a passo, as acções a executar;
 - A descrição das acções a executar pode ser feita, numa 1ª fase, recorrendo a pseudocódigo ou a diagramas de fluxo;
 - *Exemplo:*

Procedimento
cmdClear_Click

Acção (pseudocódigo)
Limpar as caixas de texto Text1 e Text2 e colocar o "foco" na caixa de texto Text1

cmdExit_Click

Terminar o programa

Etapas de desenvolvimento de um programa

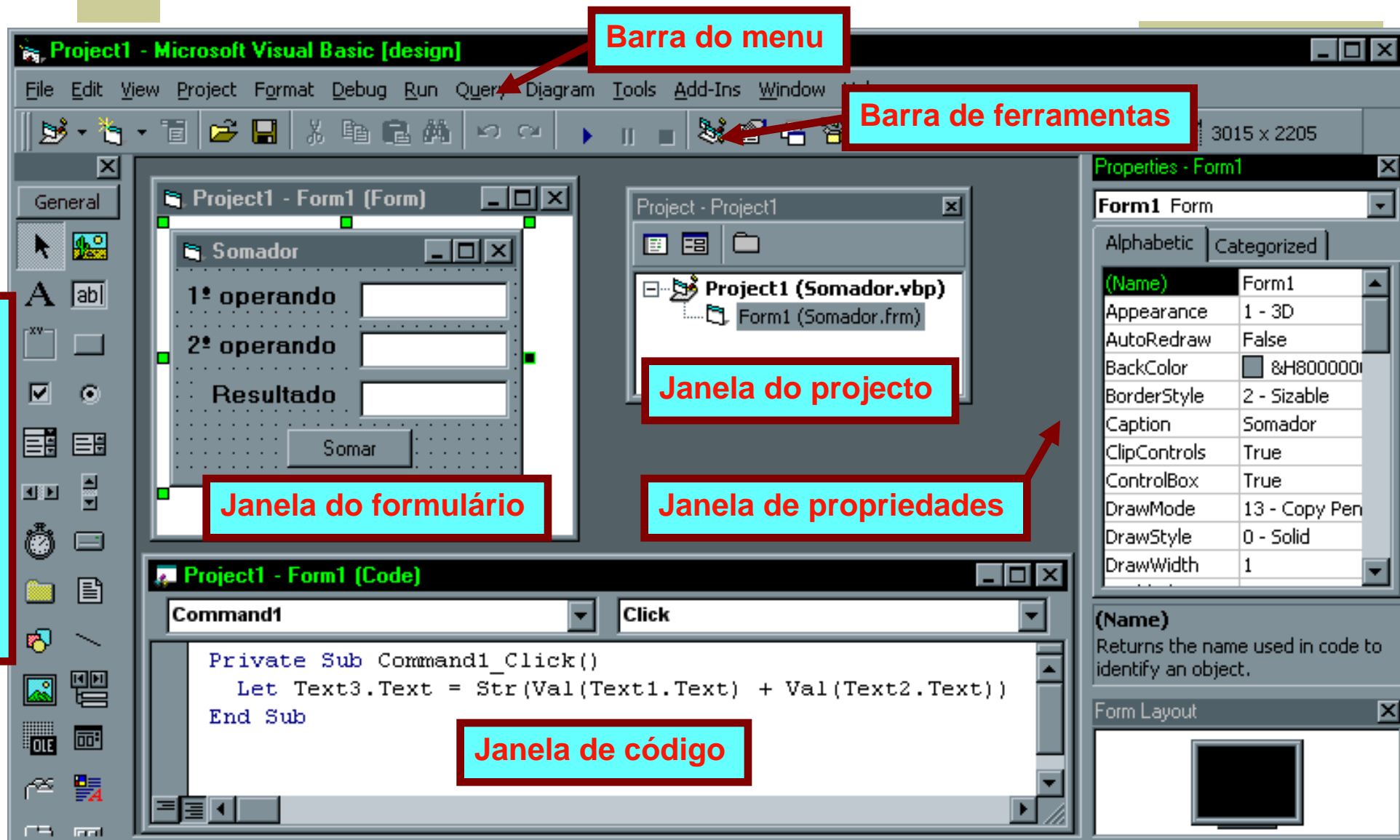
♦ *Programação*

- Criar os formulários (*forms*) e controlos concebidos na fase de planeamento;
- Fixar as propriedades dos objectos: nome, tamanho, título, ...;
- Escrever o código em *Visual Basic*;
 - O código é escrito em procedimentos (*procedures*) ou subprogramas;
 - Cada procedimento começa com as palavras `Private Sub` e termina com `End Sub`;
 - O *Visual Basic* nomeia automaticamente os procedimentos; O nome é composto pelo nome do objecto, seguido de “_” e do nome do evento.

♦ *Exemplo:*

```
Private Sub Command1_Click()  
    Let Text3.Text = Str(Val(Text1.Text)+Val(Text2.Text))  
End Sub
```

Ambiente de desenvolvimento do *Visual Basic*



Ambiente de desenvolvimento do *Visual Basic*

Janela do formulário (*form*)

- É a janela de interacção com o utilizador, onde se colocam objectos: caixas em que o utilizador pode introduzir e editar informação e botões sobre os quais o utilizador pode actuar para iniciar certas acções, etc. As caixas, botões e outros objectos são designados controles.

Janela do projecto (*project*)

- Mostra os nomes dos ficheiros que fazem parte da aplicação.
- Os ficheiros podem ser de vários tipos:
 - .FRM - formulários da aplicação;
 - .VBX - controles adicionais;
 - .BAS - blocos de código.

Ambiente de desenvolvimento do *Visual Basic*

Janela das propriedades (*properties*)

- As propriedades definem as características de cada objecto/controle da aplicação.
- O conjunto de propriedades depende do objecto/controle seleccionado.
Ex: Uma caixa de texto tem propriedades diferentes de uma figura.
- As propriedades podem ser alteradas durante a fase de construção do programa (da interface) ou durante a execução do programa (algumas delas só durante a execução).

Janela de código

- É onde se escreve o código (instruções) que o computador deve executar para responder às acções do utilizador.
- Para abrir uma janela de código basta dar um duplo clique sobre o objecto cujos eventos associados se quer tratar e no menu *Proc* seleccionar o evento a tratar (ex: *Click*, *KeyPress*, *GotFocus*, ...).

O ambiente de desenvolvimento do *Visual Basic*

Caixa de ferramentas (*toolbox*)

- Contém os diferentes tipos de objectos que podem ser colocados num formulário.
 - Caixa de texto (*text box*), etiqueta (*label*), botão de comando (*command button*), figura (*picture box*), temporizador (*timer*), ...
- Os tipos de objectos dependem da versão e da edição do *Visual Basic*.

Barra de ferramentas (*toolbar*)

- É uma colecção de ícones que executam comandos básicos, quando pressionados.

Ex: o ícone que representa uma disquete serve para gravar o programa.
- Também é possível executar os comandos associados às barras de ferramentas a partir dos menus.

Atribuição de nomes a objectos, procedimentos e variáveis

A utilização de nomes adequados para os objectos pode tornar mais fácil a compreensão do conteúdo e dos objectivos de um projecto, facilitando também a detecção e correcção de erros.

Regras de atribuição de nomes a objectos:

- o nome tem de começar por uma letra;
- pode ser seguido por outras letras, dígitos ou símbolo “_”;
- não pode conter espaços nem símbolos de pontuação;
- pode ter até 40 caracteres de comprimento.

Atribuição de nomes a objectos, procedimentos e variáveis

Algumas convenções frequentemente utilizadas:

- Começar o nome por 3 letras minúsculas, indicando o tipo de objecto

frm - *Form*

lbl - *Label*

cmd - *Command button*

opt - *Option button*

txt - *Text box*

pic - *Picture box*

...

seguido do verdadeiro nome do objecto, começado por uma maiúscula.

- Em nomes com várias palavras, escrever a primeira letra de cada palavra com maiúsculas.
- Usar nomes sugestivos da função do objecto.
 - *Exemplo:* cmdExit
lblDiscountRate

Instruções do *Visual Basic*

Instrução de atribuição

- Permite atribuir um valor a uma propriedade ou a uma variável.

- A forma geral é

```
Let objectName.property = value
```

ou

```
Let variable = value
```

significando que o valor indicado do lado direito é atribuído à propriedade ou à variável do lado esquerdo.

- A palavra reservada `Let` é dispensável, embora seja usualmente considerado que torna os programas mais legíveis.

`variable = value` é o mesmo que `Let variable = value`

Instruções do *Visual Basic*

Instrução de fim:

- A instrução End termina a execução de um programa.
- Em geral será incluída no fim do procedimento associado a um botão “Terminar” ou à opção “Terminar” de um menu.

Comentários:

- São usados para documentar o programa, tornando-o mais legível.
- Podem assumir uma das duas formas seguintes:

Rem *Comentário*

ou

' *Comentário*

Exemplo:

Rem Este programa calcula as raízes de uma equação do 2.º grau
' A, B e C são os coeficientes da equação

Detecção e correcção de erros

Podemos encontrar três tipos de erros num programa:

Erros sintácticos (de compilação):

- Erros que violam a sintaxe da linguagem.
- *Exemplos:*
 - ◆ Escrever `Ennd` em vez de `End`
 - ◆ Escrever `Label1.Capion="Teste"` em vez de `Label1.Caption="Teste"`
- Se o compilador encontrar um erro deste tipo, mostra a janela de código, assinala a linha que contém o erro e entra no modo designado por *break time*.
Os outros modos existentes são *design time* (desenho da interface e escrita do código) e *run time* (execução do programa).

Detecção e correcção de erros

Erros de execução (run-time errors):

- Erros que resultam da execução de operações incorrectas, apesar de estarem sintacticamente correctas.

Exemplo: tentativa de executar uma divisão por zero ou de calcular a raiz quadrada de um número negativo.

- Se o compilador encontrar um erro deste tipo entra em *break time* e assinala a instrução que causou o problema.

Detecção e correcção de erros

Erros semânticos (de lógica do programa):

- Erros na concepção do programa porque os algoritmos desenvolvidos não estão correctos ou a transcrição do algoritmo para o código da linguagem não foi feita correctamente.
- O programa executa mas não produz os resultados desejados.

◆ *Depuração (debugging):*

- Os erros dos programas são designados em inglês por *bugs*.
- *Debugging* é a designação da operação de detecção e correcção de erros.

Os erros semânticos são os mais difíceis de detectar.

A ajuda do *Visual Basic*

O *Visual Basic* possui um sistema de ajuda (*help*) bastante útil, que permite entre outras coisas:

- aceder a manuais da linguagem;
- ver exemplos de código e copiá-los para janelas de código do utilizador, modificando-os, se necessário;
- obter ajuda por tópicos ou por palavra-chave;
- obter ajuda sensível ao contexto, seleccionando um objecto ou uma palavra-reservada no écran e clicando em *F1*.