# Introdução
# Ao
# Desenho de Programas



## Desenvolvimento de Programas

- Programar é um processo criativo.
- A parte mais difícil de desenvolver um programa em qualquer linguagem (por exemplo Java) não é saber como expressar a nossa solução na mesma, mas sim, desenvolver um método (algoritmo) para resolver o problema.

> "An algorithm is a set of rules for getting a specific output from a specific input. Each step must be so precisely defined that it can be translated into a computer language and executed by machine."
>
> Donald Knuth

- O conceito de algoritmo é independente da linguagem de programação em que ele é programado, aliás um algoritmo até pode ser executado "à mão" por uma pessoa (ex. receitas de cozinha).



## Exemplo informal de "algoritmo"

**Rebuçados de ovos**

500 g de açúcar;
2 colheres de sopa de amêndoas peladas e raladas;
5 gemas de ovos;
250 g de açúcar para a cobertura;
farinha q.b.

Leva-se ao lume com um pouco de água e deixa-se ferver até fazer ponto de pérola. Junta-se a amêndoa e deixa-se ferver um pouco. Retira-se do calor e adicionam-se as gemas. Leva-se o preparado novamente ao lume e deixa-se ferver até se ver o fundo do tacho. Deixa-se arrefecer completamente. Em seguida, com a ajuda de um pouco de farinha, molda-se a massa de ovos em bolas. Leva-se o restante açúcar ao lume com 1dl de água e deixa-se ferver até fazer ponto de rebuçado. Passam-se as bolas de ovos por este açúcar e põem-se a secar sobre uma pedra untada, após o que se embrulham em papel celofane de várias cores.



## Desenvolvimento de Programas

- O **desenvolvimento de um programa** compreende 6 fases distintas:
  1. análise do problema
  2. desenvolvimento da solução
  3. programação da solução
  4. depuração
  5. finalização da documentação
  6. manutenção

# Análise do problema

- O **analista** estuda o problema, juntamente com os futuros utilizadores, para determinar exactamente **o que tem de ser feito**.
- Esta fase é iniciada antes de se começar a pensar na solução do problema (**especificações do problema**; **objectivos a atingir**).
- O resultado desta fase é a elaboração de documentos para especificar objectivamente, **o que faz o programa**, **estudos das possibilidades de desenvolvimento do programa**, **custos estimados**, etc.
- Os documentos servem de garantia escrita do que vai ser feito para o futuro utilizador e servem como definição dos objectivos a atingir.



# Desenvolvimento da solução

- Determinado **o que deve ser feito**, durante o desenvolvimento da solução é determinado **como deve ser feito** (desenvolvimento de um algoritmo que constitui a solução do problema a resolver).

Um **algoritmo** é uma sequência finita de instruções **bem definidas** e **não ambíguas**, cada uma das quais pode ser **executada mecanicamente** num período de **tempo finito** com uma quantidade de **esforço finita**.



# Desenvolvimento da solução

- O desenvolvimento do algoritmo deve ser feito sem ligação com uma linguagem de programação particular, pensando apenas em termos das estruturas de controlo e das estruturas de informação que vão ser necessárias.
- Os algoritmos são geralmente escritos em **pseudocódigo** (uma mistura da linguagem natural - no nosso caso português - e de uma linguagem de programação) ou apresentados sob a forma de um diagrama (**fluxograma**) e, posteriormente, traduzidos para uma linguagem de programação.
- As metodologias a seguir durante esta fase são denominadas **desenvolvimento do topo para a base** (do inglês "top down design") e refinação por passos (do inglês "stepwise refinement")



# Escrita do Algoritmo

"The sooner you start coding your program the longer it is going to take"
Henry Ledgard

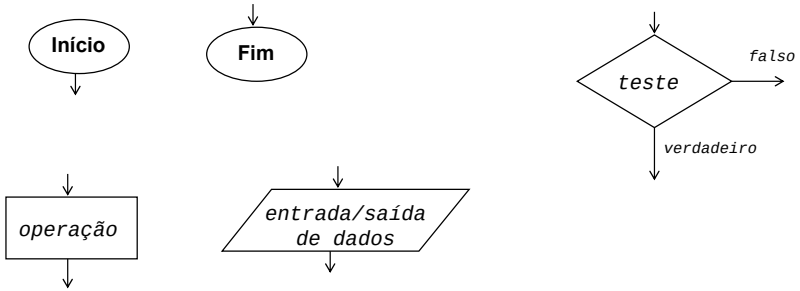
Exemplo de pseudocódigo

```
se condição então acção [senão acção] fimse
enquanto condição faz acção fimfaz
repete acção até condição fimrepete
ler variável
escrever variável(ou valor)
ir para instrução número da instrução
[calcular] operação_matemática
...
```

(as instruções poderão ser numeradas)

## Escrita do Algoritmo

Simbologia para os fluxogramas






## Programação da solução

- escrever o algoritmo desenvolvido recorrendo a uma linguagem de programação.
- escolha da linguagem de programação (2 critérios)
  - as linguagens existentes no computador que vai ser utilizado
  - a natureza do problema a resolver
- Uma vez decidida qual a linguagem de programação a utilizar, e tendo já uma descrição do algoritmo, a geração das instruções do programa é relativamente fácil. O programador terá então de decidir como representar as estruturas de informação necessárias e escrever as respectivas operações.
- Traduzir as instruções do seu algoritmo para instruções escritas na linguagem de programação a utilizar.



## Correcção de programas

- É importante escrever algoritmos (programas) correctos (**correcção de programas**), *i.e.* desenvolver algoritmos que satisfaçam determinadas propriedades (*e.g.* **terminação**, **correcção parcial e total**).
  - **Terminação**
    Um programa deverá inevitavelmente terminar a sua execução.
  - **Correcção Parcial e Total**
    (**Parcial**) Um programa nunca termina com uma resposta incorrecta, *i.e.* não é garantido que o programa termina e, consequentemente, produza alguma resposta; mas se terminar a resposta é correcta.
    (**Total**) Um programa termina inevitavelmente produzindo uma resposta correcta.



## Depuração

- Detectar, localizar e corrigir os erros contidos no programa desenvolvido (*debugging*)
  - **erros sintácticos** - resulta da não conformidade de uma instrução com as regras sintácticas de uma linguagem de programação. São detectados pelo compilador e, normalmente, são fáceis de corrigir.
  - **erros semânticos (ou lógicos)** - o programa não executa a acção pretendida. Resulta de erros (humanos) no desenho do algoritmo ou na sua implementação. Geralmente são os mais difíceis de detectar e corrigir
  - **Erros de *run-time*** - Erros produzidos durante a execução do programa (e.g. divisão por zero, *overflow*, violar a dimensão limite de um vector).

# Depuração

- **a fase de teste**
  - verificar se o programa resolve o problema para que foi proposto para todos os valores possíveis dos dados
  - os dados de teste deverão ser escolhidos criteriosamente de modo a testarem todos os caminhos, ou rastos, possíveis através do algoritmo

"testar programas pode ser usado para mostrar a presença de erros, mas nunca para mostrar a sua ausência"
E.W.Dijkstra



# Finalização da documentação

- A documentação de um programa é de dois tipos:
  - a documentação destinada aos utilizadores do programa (a documentação de utilização)
  - documentação destinada aos técnicos que irão fazer a manutenção e possíveis alterações no programa (a documentação técnica)



# Manutenção

- Esta fase decorre depois do programa ter sido considerado terminado, e tem duas facetas distintas:
  - verificação constante da possibilidade de alterações nas especificações do problema, e no caso de alteração de especificações, na alteração correspondente do programa
  - correcção de eventuais erros descobertos durante o funcionamento do programa



# Tipos de dados

- Um tipo de dados identifica um conjunto de valores (o conjunto suporte do tipo) e um conjunto de operações que estão disponíveis sobre esses valores. Estes valores podem ser assumidos por uma constante, variável ou expressão ou, ainda, gerados por uma função
- Exemplos de tipos de dados:
  - Inteiros (admitem valores do conjunto dos inteiros)
  - Reais (admitem valores do conjunto dos reais)
  - Booleanos (admitem apenas os valores True e False)
- Cada operador ou função tem argumentos de um dado tipo e produz resultados de um determinado tipo

# Operadores numéricos

- Os operadores numéricos são aqueles que usados em expressões devolvem um valor do tipo com que foi declarada a variável que o recebe.

| DESIGNAÇÃO | SIGNIFICADO |
|---|---|
| **Adição** | + |
| **Subtracção** | - |
| **Multiplicação** | * |
| **Divisão real** | / |
| **Divisão inteira** | / |
| **Resto da divisão inteira** | % |



# Operadores numéricos

- **EXEMPLOS:**
  - **x ← x + 1**
    adiciona uma unidade ao valor da variável x
  - **num ← a * b / 2**
    atribui à variável *num* o quociente do produto das variáveis *a* e *b* por 2
  - **10 / 3**
    apresenta o resultado de 3 – o quociente da divisão inteira de 10 por 3
  - **10 % 3**
    apresenta o resultado de 1 – o resto da divisão inteira de 10 por 3



# Operadores relacionais ou de comparação

- Os operadores relacionais ou de comparação são aqueles que permitem estabelecer comparações, donde resultará um valor lógico (verdadeiro ou falso). Assim, as expressões que os utilizam passam a ser expressões do tipo booleano.

| DESIGNAÇÃO | NOTAÇÃO |
|---|---|
| **Igual a** | = |
| **Diferente de** | < > |
| **Maior que** | > |
| **Menor que** | < |
| **Maior ou igual a** | > = |
| **Menor ou igual a** | < = |



# Operadores relacionais ou de comparação

- **EXEMPLOS:**
  - **num > 10**
    dará como resultado Verdadeiro ou Falso consoante o valor da variável seja maior ou menor ou igual do que 10
  - **20 / 4 = 6**
    tem o valor Falso porque o quociente de 20 por 4 é 5
  - **'A' < > 'a'**
    apresenta o resultado Verdadeiro porque os caracteres 'A' e 'a´ na realidade são diferentes

# Operadores lógicos ou booleanos

- Os operadores lógicos ou booleanos são aqueles que correspondem às funções lógicas, como por exemplo, a conjunção (E lógico) ou a disjunção (OU lógico).

| DESIGNAÇÃO | NOTAÇÃO |
|---|---|
| **Negação** | ~ , NOT |
| **Conjunção ou E lógico** | ∧ , AND |
| **Disjunção ou OU lógico** | ∨ , OR |
| **Disjunção exclusiva ou XOR lógico** | ∨ , XOR |



# Tabelas de verdade da lógica bivalente

**AND**

| p | q | p ∧ q |
|---|---|---|
| V | V | **V** |
| V | F | **F** |
| F | V | **F** |
| F | F | **F** |

**OR**

| p | q | p ∨ q |
|---|---|---|
| V | V | **V** |
| V | F | **V** |
| F | V | **V** |
| F | F | **F** |

**NOT**

| p | ~ p |
|---|---|
| V | **F** |
| F | **V** |



# Operadores lógicos ou booleanos

- **EXEMPLOS:**
  - **(x>1) AND (x<=10)**
    dará como resultado Verdadeiro se o valor da variável x for maior do que 1 e menor ou igual a 10. Caso contrário, será Falso.
  - **(y =5) OR (10 =10)**
    terá o valor Verdadeiro porque mesmo não conhecendo o valor de y=5 sabe-se que 10=10 é sempre verdadeiro.
  - **NOT (x=10)**
    é o mesmo que x <>10 e será Verdadeiro se o valor da variável x for diferente de 10 e Falso se for igual a 10.