Apontamentos de Programação

2004/2005

José Manuel Torres

ALGORITMOS E COMPUTAÇÃO

Algoritmos

Noção de Algoritmo

Noção Informal: Sequência finita e não ambígua de instruções elementares bem definidas, conducente à solução de um determinado problema, cada uma das quais pode ser executada mecanicamente, numa quantidade finita de tempo e com uma quantidade finita de esforço.

Objectivo $\Rightarrow$ Sub-Objectivos $\Rightarrow$ Passos de Resolução

Algoritmo Mudança de Lâmpada

Exemplo 1: Substituir uma lâmpada fundida de um candeeiro.

Passo	Descrição
1	Seleccione uma nova lâmpada
2	Remova a Lâmpada fundida
3	Insira uma nova lâmpada

1.1	Seleccione uma lâmpada da mesma potência da fundida
2.1	Posicione a escada em baixo do candeeiro
2.2	Suba a escada até que possa atingir a lâmpada
2.3	Rode a lâmpada no sentido contrário ao dos ponteiros do relógio até que esta se solte
3.1	Coloque a nova lâmpada no orifício correspondente
3.2	Rode a lâmpada no sentido dos ponteiros do relógio até que fique presa
3.3	Desça da escada

Definição mais precisa para o passo 1.1:

Dada uma caixa com lâmpadas, seleccione uma lâmpada candidata à substituição
Se a lâmpada não é da mesma potência da antiga, **então**,
  repita os passos seguintes **até** encontrar uma lâmpada da potência desejada:
        Pouse a lâmpada seleccionada
        Seleccione uma nova lâmpada

- Por exemplo, para os passos 2.2, 2.3, 3.2 e 3.3 poderiam também derivadas descrições mais precisas e detalhadas, do tipo: Repita ... até ...
- Aumento do detalhe do algoritmo pode continuar quase indefinidamente.
- Grau de detalhe depende das necessidades do agente que vai executar o algoritmo.

Algoritmo Lista-Telefónica

Exemplo 2: Encontrar o número de telefone que corresponde a um dado nome numa lista telefónica.

Passo	Descrição
1	Encontre a página da lista que contém o último apelido do nome
2	Encontre na página determinada no passo 1, o nome procurado

Aumentando o detalhe, obtemos instruções elementares não ambíguas:

1.1 Coloque o marcador D (dedo) ao acaso na lista
1.2 Abra a lista
1.3 Último apelido está contido numa das páginas abertas (esquerda ou direita)? Se sim, siga para o passo 2
1.4 Último apelido precede a página esquerda? Se sim, coloque o marcador atrás da página esquerda; se não, coloque o marcador à frente da página direita.
1.5 Vá para 1.2 (retome a sequência de instruções no passo 1.2)

Eliminando formulações mal definidas (ex.: coloque o marcador atrás):

1.1.1 Torne **A** igual ao apelido do nome a seleccionar (atribuição à variável **A)**
1.1.2 Escolha uma posição **n** ao acaso no intervalo [1,N]
(**n** representa o número de páginas útil da lista)
1.1.3 Torne **D** igual a **n** (atribua à variável **D** o valor **n**)
1.2 Abra a lista no local seleccionado pelo marcador **D**
1.3 **A** está contido numa das páginas abertas (esquerda ou direita)? Se sim, siga para o passo 2.
1.4 **A** precede o primeiro apelido da página esquerda? Se sim, faça **n** igual a **(n+1)/2** (actualização do valor de **n**); se não, faça **n** igual a **(N+n)/2**.
1.5 Vá para 1.2 (retome a sequência de instruções no passo 1.2)

Algoritmo

Incluí:

- **Acções** (escolha ... , coloque ... , etc.)
- **Testes de Condições** (apelido precede a página esquerda?, etc.)
- **Saltos** para passos fora da sequência normal (siga, vá para ...)

Formulações Rejeitáveis:

- "Se n for muito grande..." (especificação qualitativa)
- "Escreva qualquer coisa..." (domínio infinito)
- "Para o maior inteiro n tal que $x^n+y^n=z^n$, com x, y, z inteiros..." (solução matemática desconhecida)

Noção Formal de Algoritmo:

Um algoritmo é um processo discreto (sequência de acções indivisíveis), determinístico (para cada passo da sequência e para cada conjunto válido de dados, corresponde uma e uma só acção), que termina quaisquer que sejam os dados iniciais (pertencentes a conjuntos pré-definidos).

Características Importantes de um Algoritmo

- **Entradas:** Quantidades inicialmente especificadas (por exemplo através de instruções de leitura.
- **Saídas**: Uma ou mais saídas (habitualmente por instruções de escrita)
- **Finitude:** A execução deve terminar sempre num número finito de passos.
- **Precisão:** Todos os passos do algoritmo devem ter um significado preciso não ambíguo, especificando exactamente o que deve ser feito. Para evitar a ambiguidade das linguagens humanas (linguagens naturais), linguagens especiais (denominadas linguagens de programação) foram criadas para exprimir algoritmos.
- **Eficácia:** Os passos devem conduzir à resolução do problema proposto. Devem ainda ser executáveis numa quantidade finita de tempo e com uma quantidade finita de esforço.
- **Eficiência:** Em muitos casos colocam-se questões de eficiência a um algoritmo.

Algoritmo MDC

Exemplo 3: Dados dois números inteiros **m** e **n**, calcular o seu máximo divisor comum **mdc(m, n)**

1. **Leia** m, n (conjunto de valores válidos: inteiros, $\neq 0$)
2. **Se** m<n **Então** torne min = m **Senão** torne min = n
3. Torne mdc = min
4. Torne r1 = mod(m, mdc)
5. Torne r2 = mod(n, mdc)
6. **Se** r1 = 0 e r2 = 0 **Então Escreva** o valor de mdc; Stop
   Senão Torne mdc = mdc - 1 e **Vá Para 4**

No algoritmo MDC usaram-se as seguintes instruções:

Leitura e escrita de dados: Entradas e saídas de valores
Utiliza-se em notação formal leia(...) e escreva(...)

Atribuição de valor: Corresponde a acções do tipo "torne", "tome", "calcule", "seja"
Utiliza-se em notação formal $\leftarrow$: por exemplo m $\leftarrow$ m+1, mdc $\leftarrow$ min

Teste de condição: "Se ... então ... senão ...". Podem-se utilizar condições lógicas do tipo "Se x>0 E y>0 OU z<x então ...senão ... "
Utiliza-se em notação formal a sintaxe "Se ... então ... senão ..."

Salto: Redirecciona a sequência de execução "vá para"
Utiliza-se em notação formal a sintaxe "Vá para"

Paragem: Instrução "Stop"
Utiliza-se em notação formal "Stop"

Funções predefinidas: A função matemática mod(x, y) que permite calcular o resto da divisão inteira entre o número inteiro x e o número y

Conjunto mínimo de instruções de um computador corresponde às apresentadas anteriormente mais instruções de gestão do espaço de variáveis.

Representação de Algoritmos

Na fase de Desenvolvimento podemos utilizar:

- **Linguagem auxiliar (pseudo-código).**
- **Diagrama de Fluxo (fluxograma).**

Meio mais rigoroso e eficaz:

- **Linguagens de Programação.**

Linguagem auxiliar (pseudo-código)

Notação Utilizada:	Descrição
Leia(...) ou LeiaLinha(...)	Leitura de dados, atribuindo o valor lido a uma variável
Escreva(...) ou EscrevaLinha(...)	Escrita ou saída de dados
Stop	Fim de programa
<*bloco*>: Inicio .... Fim	Delimita uma sequência de instruções que funciona em bloco
<*operador de atribuição*>: ←	Atribui o valor que está do lado direito à variável que está do lado esquerdo
Se <*condição*> Então <*bloco1*> Senão <*bloco2*>	Estrutura condicional booleana: se a condição for verdadeira executa o *bloco1* senão executa o *bloco2*
Caso <*expressão*> Seja <*valor_1*> : <*bloco_1*> .... <*valor_n*> : <*bloco_n*> Senão <*bloco_senão*> FimCaso	Estrutura condicional múltipla: avalia o valor da expressão e em função disso selecciona e executa apenas um dos blocos
Para <*i ← expressão_inicial*> Até <*expressão_final*> Faça <*bloco*>	Estrutura repetitiva (ciclo): existe uma variável de ciclo *i* que vai tomar todos os valores entre *expressão_inicial* e *expressão_final*. Para cada valor de *i,* o *bloco* será executado.
Enquanto <*condição*> Faça <*bloco*>	Estrutura repetitiva (ciclo): enquanto a condição for verdadeira executa o bloco
Repita <*bloco*> AtéQue <*condição*>	Estrutura repetitiva (ciclo): executa o bloco até que a condição seja verdadeira
Vá para	Instrução de salto: o programa (o fio de execução) salta para a instrução referida

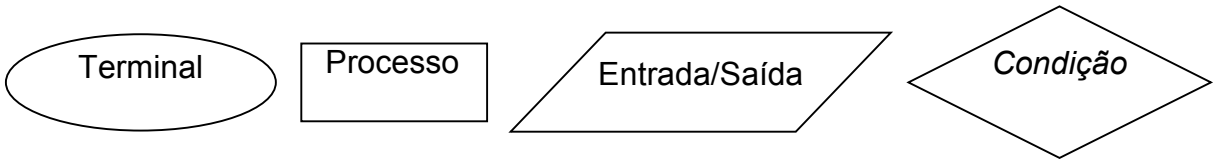
Exemplo 1: Algoritmo de cálculo do MDC

```
1. Leia(m, n) (Z+)
2. Se m < n Então min ← m Senão min ← n
3. mdc ← min
4. r1 ← mod(m, mdc)
5. r2 ← mod(n, mdc)
6. Se r1 = 0 E r2 = 0
7.         Então Escreva(mdc), Stop
8.         Senão mdc ← mdc-1, Vá Para 4
```

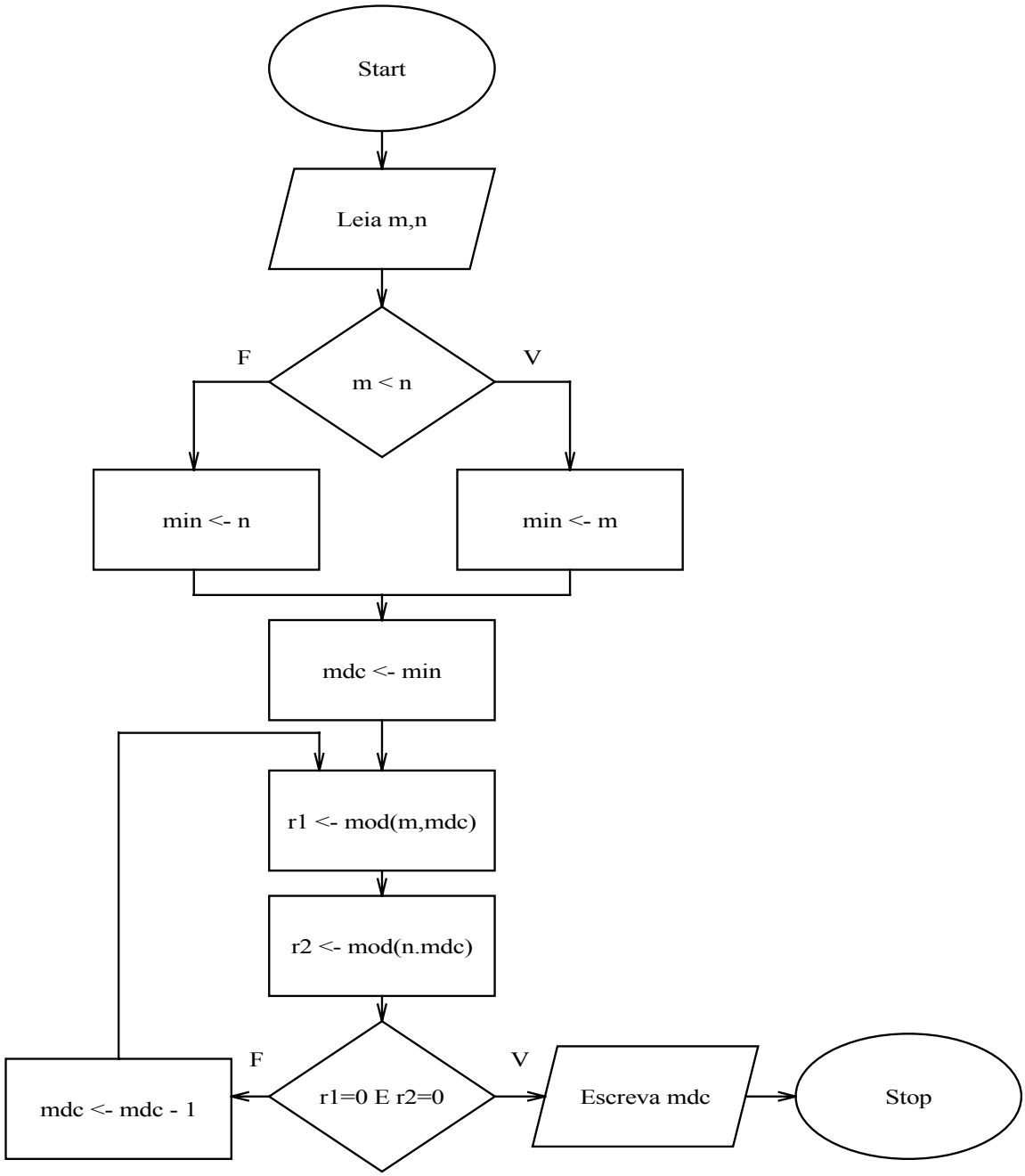
- Pode/deve usar avanços de parágrafo para salientar blocos de instruções dentro de outras instruções
- Pode/deve usar numeração no inicio de cada linha para mais facilmente testar e corrigir o algoritmo

Diagrama de Fluxo (fluxograma)

Símbolos Utilizados:

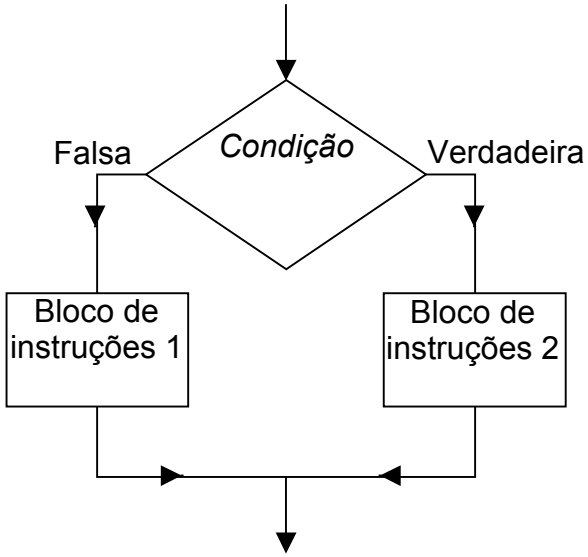


Exemplo 1: Algoritmo de cálculo do MDC

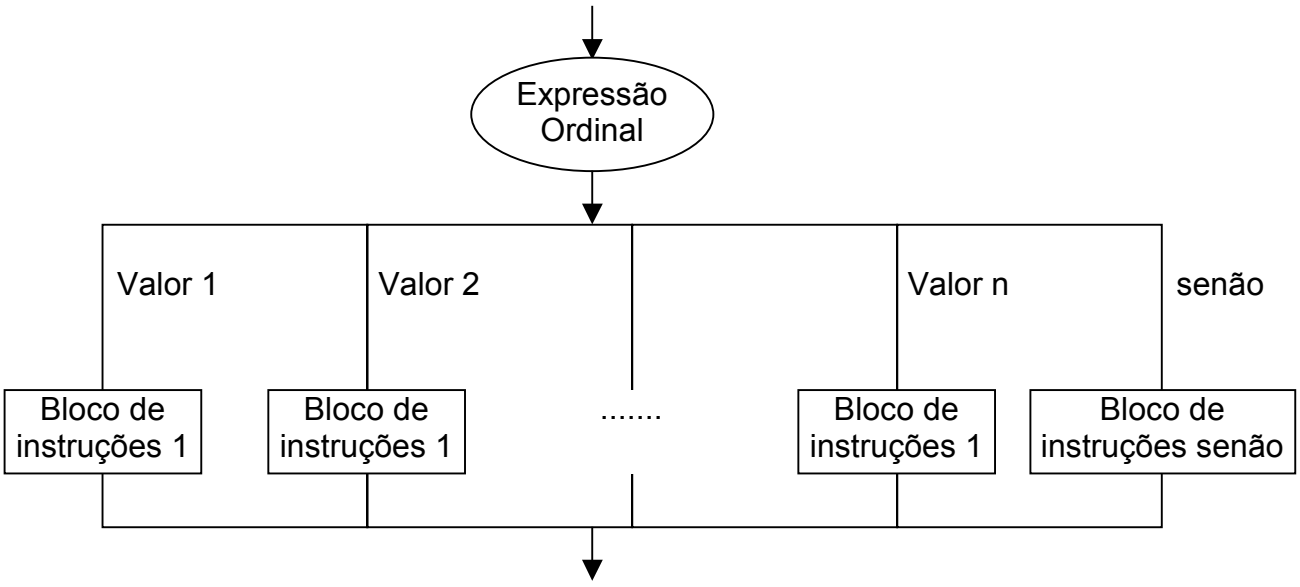


Resumo das estruturas de controlo de fluxo

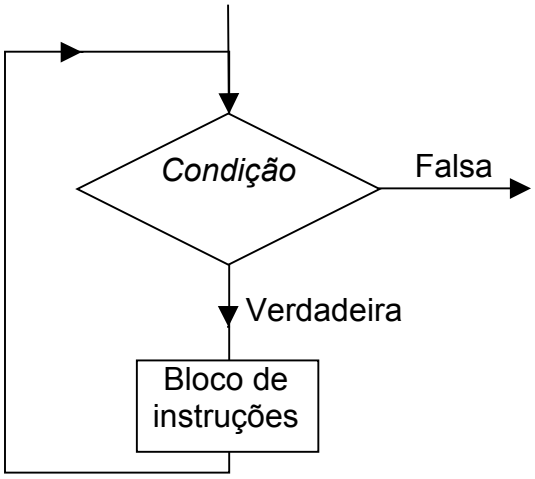
Se <*condição*> Então <*bloco_1*> Senão <*bloco_2*>



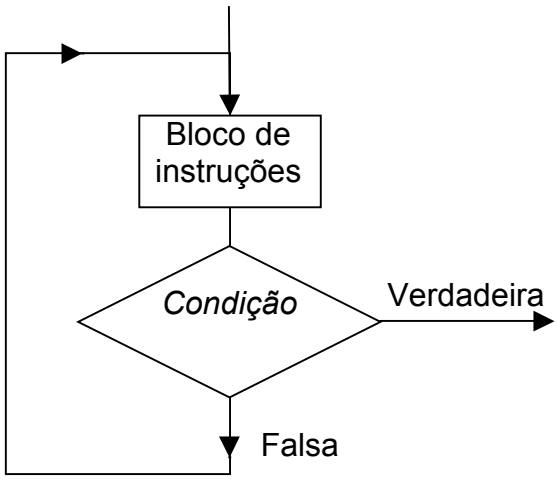
Caso <*expressão_ordinal*> Seja
<*valor_1*> : <*bloco_1*>
....
<*valor_n*> : <*bloco_n*>
Senão <*bloco_senão*>
FimCaso



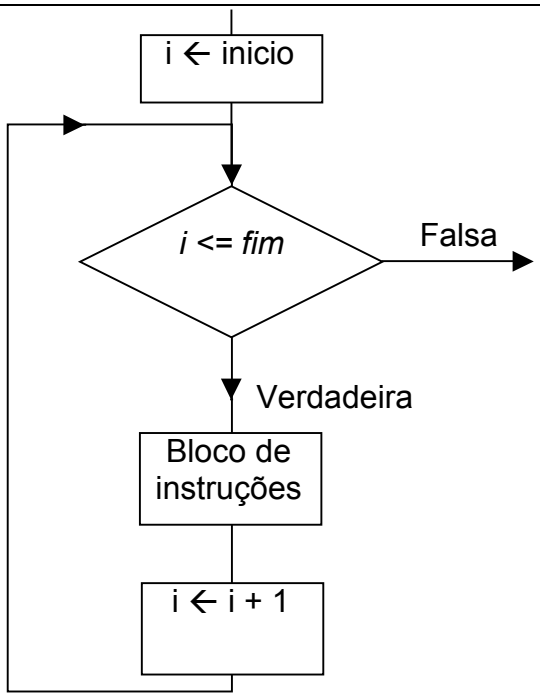
Enquanto <*condição*> Faça <*bloco*>



Repita <*bloco*> AtéQue <*condição*>



Para <*i* ← *expressão_inicial*> Até <*expressão_final*>
Faça <*bloco*>



Linguagem de Programação

O algoritmo desenvolvido para o cálculo do MDC pode facilmente ser implementado em qualquer linguagem: Pascal, C, Basic, Fortran, etc.

Linguagem Pascal

```
Program MaxDivComum;

Var n1, n2, mdc, r1, r2: integer;

Begin
  Write('Introduza o primeiro numero: ');
  Readln(n1);
  Write('Introduza o segundo numero: ');
  Readln(n2);
  If n1<n2 then mdc := n1 else mdc := n2;
  r1 := n1 mod mdc;
  r2 := n2 mod mdc;
  While (r1<>0) or (r2<>0) do
  Begin
          mdc := mdc - 1;
          r1 := n1 mod mdc;
          r2 := n2 mod mdc;
  End;
  Writeln('Maximo Divisor Comum: ', mdc);
End.End.
```

Linguagem muito simples e didáctica mas ao mesmo tempo - nas mais recentes implementações - muito poderosa. O código é extremamente semelhante ao pseudo-código apresentado.
Utiliza-se actualmente o Pascal por Objectos em ambiente de desenvolvimento Delphi.

Linguagem C

```
void main()
{
        int m, n, min, mdc, r1, r2;
        scanf("%d %d",&m,&n);
        if (m < n) min = m; else min = n;
        mdc = min;
        r1= m % mdc;
        r2= n % mdc;
        while (r1!=0 || r2!=0)
        {
                mdc--;
                r1 = m % mdc;
                r2 = n % mdc;
        }
        printf("%d\n",mdc);
}
```

Linguagem muito poderosa mas cujo código (muito condensado) já não se assemelha muito ao pseudo-código apresentado.
Utilizam-se actualmente compiladores como o OpenWatcom (software gratuito e opensource), Borland C++ ou o Microsoft Visual C++

Linguagem Basic com saltos

```
10 Input m, n
20 If m < n then min = m else min = n
30 mdc = min
40 r1 = m Mod mdc
50 r2 = n Mod mdc
60 if r1<>0 or r2<>0 then mdc = mdc - 1: Goto 40
70 Print mdc
80 Stop
```

Linguagem MS-Visual Basic (VBA para MS-Excel)

```
Sub mdc()
' declaração das variáveis
Dim n1 As Integer, n2 As Integer, mdc_val As Integer
Dim menor As Integer, maior As Integer
' selecciona "A1"
Range("A1").Select
' le valor de n1 na celula b1
n1 = CInt(Selection.Offset(0, 1).Value)
' le valor de n2 na celula b2
n2 = CInt(Selection.Offset(1, 1).Value)
   If n1 < n2 Then
   mdc_val = n1
   Else
   mdc_val = n2
   End If
   r1 = n1 Mod mdc_val
   r2 = n2 Mod mdc_val
   While (r1 <> 0) Or (r2 <> 0)
      mdc_val = mdc_val - 1
      r1 = n1 Mod mdc_val
      r2 = n2 Mod mdc_val
   Wend
' coloca resultado na celula b3
Selection.Offset(2, 1).Value = mdc_val
End Sub
```

Para programas muito simples é uma boa linguagem pois nas suas implementações mais simples dispensa a declaração de variáveis.
Utilizam-se essencialmente o Microsoft Visual Basic.

Nota: Em MS-Visual Basic, as linhas do programa que começam pelo símbolo ‘ são comentários do programador e são ignoradas pelo computador.

Prova e Teste de Algoritmos

Seguimento e Teste de Algoritmos ou Programas

Forma muito simples de o efectuar: Seguir a execução do programa passo a passo e verificar a evolução de todas as variáveis.

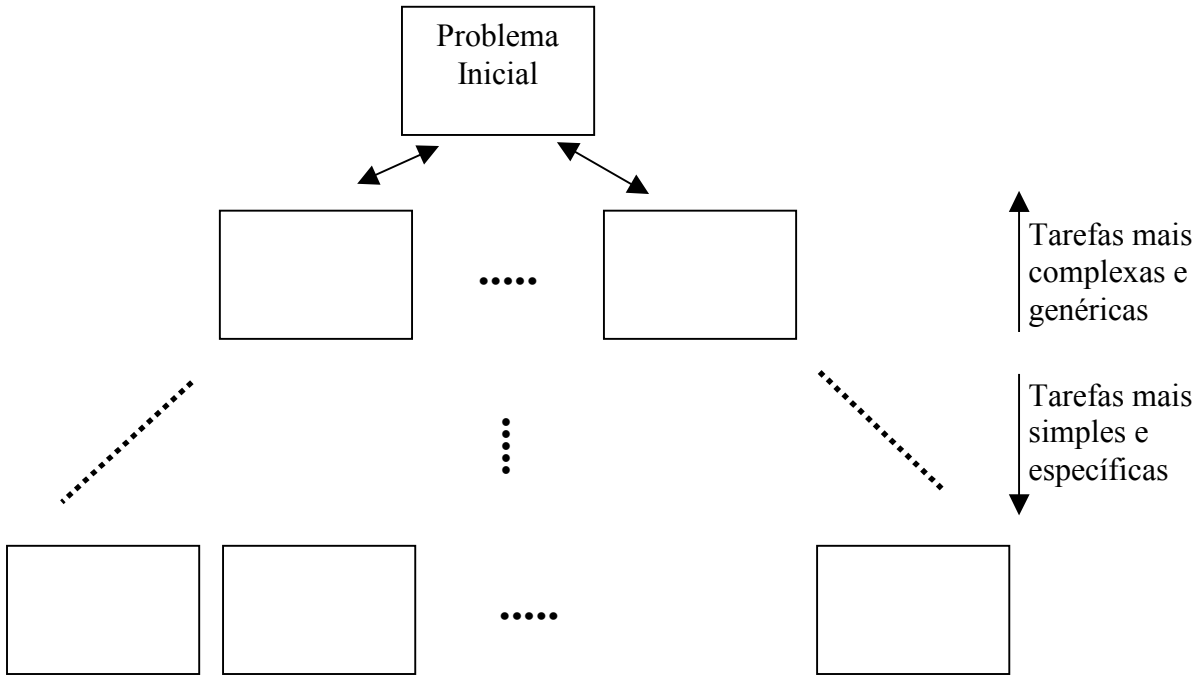
Exemplo 1: Seguimento do Algoritmo MDC (exemplo 1 expresso em pseudo-código) para m=8 e n=20

Passo	Variáveis						Testes		Entradas/ Saídas
	m	n	min	mdc	r1	r2	r1=0 e r2=0	m < n	
1	8	20							Leia(8,20)
2			8					V	
3				8					
4					0				
5						4			
6							F		
8				7					
4					1				
5						6			
6							F		
8				6					
4					2				
5						2			
6							F		
8				5					
4					3				
5						0			
6							F		
8				4					
4					0				
5						0			
6							V		
7									Escreva(4)

O seguimento (rastreio) de algoritmos pode detectar a presença de erros mas não pode comprovar a sua ausência (exceptuando em casos triviais).

Programação "top-down" e "bottom-up"

O desenvolvimento de grandes programas informáticos exige que em determinada altura haja um processo de decomposição de problemas ou tarefas complexas e muito abrangentes em tarefas mais pequenas e especificas, isto é, mais fáceis de lidar.
Esta técnica pode-se aplicar várias vezes de modo a se obter uma espécie de hierarquia ou árvore de tarefas em que na raiz reside o problema ou tarefa inicial e nas folhas as tarefas mais simples e implementáveis em sub-rotinas.



Esta metodologia de decomposição de um problema em partes mais pequenas claramente definidas e com mecanismos de interacção claros e bem limitados designa-se por programação “top-down”. A construção ou desenvolvimento do programa é feita implementando cada uma dessas partes.

Top-Down:

Tarefas genéricas/complexas → Tarefas específicas/simples

A programação “bottom-up” usa a lógica contrária, isto é, identifica num programa complexo, tarefas bem definidas e que foram reconhecidas como necessárias ao seu desenvolvimento. Depois da implementação dessas partes o programador irá reuni-las de modo a obter a solução para o problema mais complexo.

Bottom-Up:

Tarefas específicas/simples → Tarefas genéricas/complexas

De um modo geral o desenvolvimento de um programa poderá adoptar uma abordagem mista “top-down” e “bottom-up”.

Resolução de Problemas

Resolução de Problemas Simples

Confrontado com um problema, a sua solução computacional pode ser determinada seguindo os seguintes passos:

- Percepção do Problema (útil explicitar as saídas desejadas para diversas entradas)
- Formulação de um algoritmo geral para a resolução (dando pouca atenção a detalhes) e seu seguimento utilizando algumas das entradas anteriormente definidas de forma a confirmar que obtém os resultados esperados).
- Identificação e listagem de todas as variáveis necessárias (e dos seus tipos e descrições).
- Preenchimento dos detalhes nos passos do algoritmo geral definido (subdividindo passos e garantindo que efectuam a mesma função que os passos antes da subdivisão).
- Seguimento do algoritmo utilizando algumas das entradas definidas anteriormente.
- Implementação do algoritmo numa linguagem de programação específica (Pascal, C, Basic, etc).

Resolução de Problemas Mais Complexos

A resolução de problemas mais complexos, nomeadamente a implementação de sistemas informáticos de média ou grande dimensão exige os seguintes passos:

- Análise do Problema (perceber bem o problema)
- Especificação do Programa (definir estruturas de dados e algoritmos adequados)
- Codificação do Programa
- Teste e Refinamento do Programa
- Geração de Documentação
- Manutenção do Programa

Características de um bom Programa

Um bom programa deve ter as seguintes características:

- Satisfaz as especificações do problema
- Não contém erros e é robusto
- É legível e inteligível
- É portátil e de fácil manutenção

INTRODUÇÃO À PROGRAMAÇÃO

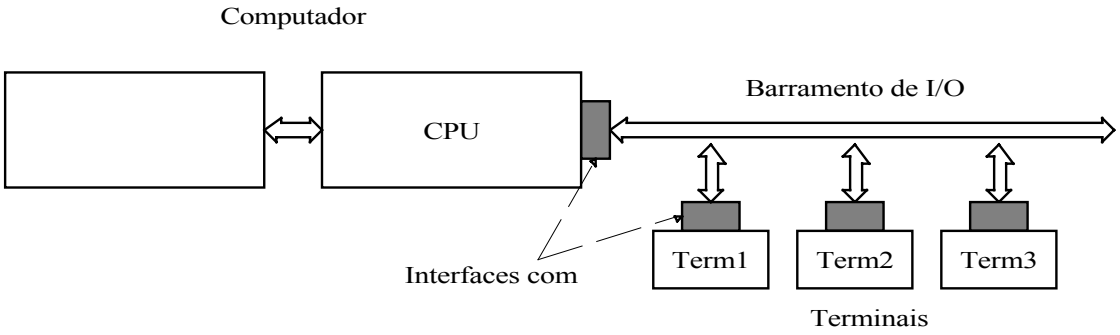
Estrutura Básica de um Sistema de Computação Digital

Sistema

Conjunto organizado de elementos que conjuntamente realizam determinados objectivos.

Sistema de Computação Digital

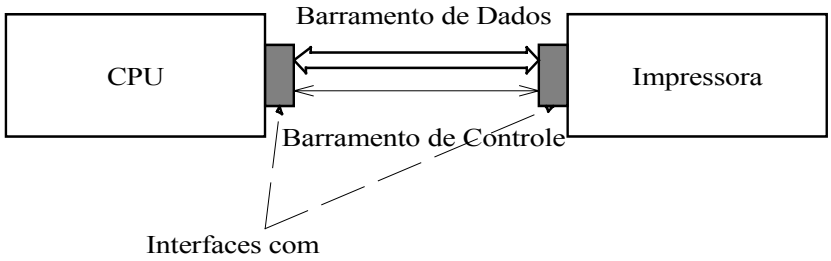
- Sistema constituído por peças de equipamento electrónico destinado a processar informação sob a forma digital.
- Informação é fornecida ao sistema através de terminais (ou periféricos) de entrada de dados (teclados, ratos, scanners, modems, accionadores de discos magnéticos ou de fita magnética, conversores A/D, etc).
- Informação é processada através da Unidade de Processamento Central (CPU) do sistema, estando localizada a informação a processar e as instruções de processamento, na Memória do computador.
- Depois de processada a informação, os resultados são transmitidos para o exterior através de terminais (ou periféricos) de saída de dados (monitores de vídeo, modems, impressoras, plotters, accionadores de discos magnéticos ou de fita magnética, conversores D/A, etc).



- **Memória**: Armazena informação e instruções de processamento a executar pelo CPU.
- **CPU**: Executa e coordena o processamento da informação.
- **Barramento (Bus)**: Via de comunicação de informação.
- **Periférico**: Realiza operações de entrada/saída de informação.
- **Interfaces**: Gerem as transferências CPU/Terminais, compatibilizando as diferentes velocidades de processamento. Dispõe de "buffers" de memória para armazenamento temporário.

Comunicação dos Periféricos com o CPU

- Os periféricos comunicam com o CPU e processam a informação a baixa velocidade.
- Exemplo: Comunicação CPU-Impressora. A informação é enviada a grande velocidade para a impressora e é escrita no papel a muito reduzida velocidade



Sequência de acções de processamento para realizar a comunicação entre a impressora e o CPU:

CPU: Impressora livre?
Impressora: Sim.
CPU: Envia Dado à Impressora.
Impressora: Comunica que recebeu o dado
CPU: Envia novo dado até que:
  Impressora: Impressora cheia
  CPU Envio concluído

Enquanto a Impressora despeja lentamente o Buffer, imprimindo os dados nele contidos, o CPU realiza a alta velocidade, outras operações. Mais tarde, quando o buffer tiver espaço disponível, o ciclo anterior é retomado.

Execução de um Programa num Sistema de Computação Digital

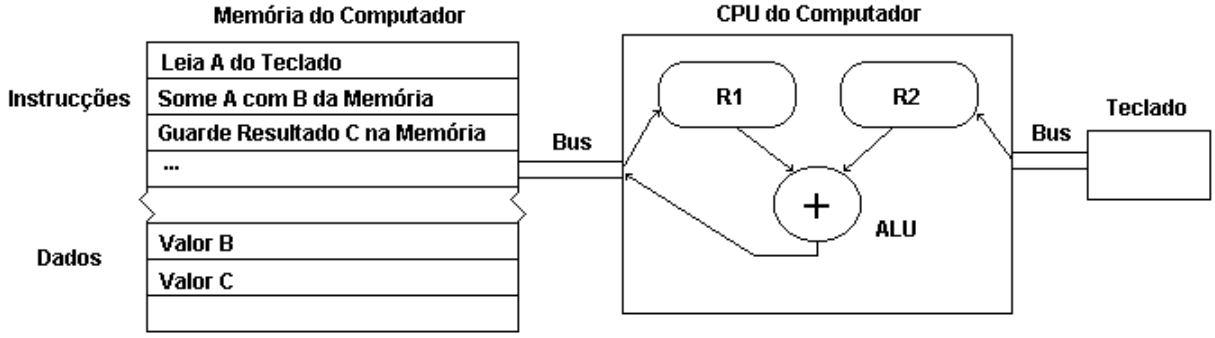
Programa: Conjunto organizado de instruções e dados, que implementa um determinado algoritmo.

Instrução: Dado interpretado como uma acção de processamento de informação

Exemplo: Soma de um número lido do teclado com um contido na memória.

Acções de processamento executadas ou coordenadas pelo CPU:

1. Leitura do número A do teclado e seu armazenamento num registo R1.
2. Transmissão do número B da memória para o registo R2.
3. Soma de R1 com R2 (executada na ALU), obtendo o número C
4. Transmissão do resultado C para a memória.



Processamento Digital

Processamento de dados representados por sequências de bits. Os dados são múltiplos ou submúltiplos da **palavra** (“Word”), unidade fundamental de processamento de um sistema. O comprimento da palavra é igual ao seu número de **bits** (número de fios do barramento de dados). Normalmente são múltiplos de um conjunto de 8 bits (**byte**):

- Memória Armazena palavras
- CPU Processa palavras
- Barramentos Transferem palavras

Os comprimentos mais usuais de palavras são os 8, 16, 32 e 64 bits. Exemplo de uma palavra de 8 bits: 0101 1001.

Tipos De Informação

Os sistemas de computação digital utilizam vulgarmente três tipos de informação:

- Informação Simbólica
- Informação Numérica
- Informação Contextual

Informação Simbólica

Símbolos	Letras	(**a** a **z**, **A** a **Z**)
	Dígitos Decimais	(**0** a **9**)
	Sinais de Operação	**+ - * /**
	Sinais de Pontuação	**, . ; : ? !**
	Símbolos Especiais	**“ # & % $ ‘ |**

Os **símbolos são codificados** em binário com um **código de comprimento n**. Os códigos usuais têm 8 bits (**n=8**, 256 símbolos representáveis), por exemplo o ASCII ou EBCDIC.

Informação Numérica

Os dados representam quantidades numéricas no sistema de numeração binário (ou num outro sistema codificado em binário).

$0101\ 1011 = 0*2^7 + 1*2^6 + 0*2^5 + 1*2^4 + 1*2^3 + 0*2^2 + 1*2^1 + 1*2^0 = 64+16+8+2+1 = 91$

Informação Contextual

Conteúdo informativo (significado) de um dado, depende do contexto. Por exemplo, o dado 0100 0001, pode representar:

65	Numa operação utilizando binário
41	Numa operação utilizando BCD (“Binary Coded Decimal”)
A	Símbolo (letra) ASCII
MOV A,B	Instrução do microprocessador INTEL 8080

O contexto pode depender do uso no problema particular, da posição numa sequência de dados ou da posição física no sistema de computação.

Unidade De Processamento Central

Funções:

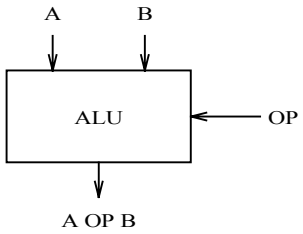
- Armazenamento temporário de informação
- Processamento de informação
- Controlo do fluxo de informação

Componentes Principais

1 - Registos: Conferem um significado especial a sequências de bits armazenadas temporariamente:

- Registo de Instrução **IR** : Determina (através da "control unit") o fluxo de informação no CPU
- Contador de Programa **PC** : Aponta para a instrução (na memória) a executar
- Registos de Dados **Ri** : Registos com valores, podendo ter um número variável. Por vezes um destes registos é designado de Acumulador **A**, sendo nele colocados os resultados das operações aritméticas.

2 - Unidade Lógica e Aritmética ALU : Efectua operações com os valores dos registos (somas, subtracções, operações lógicas, etc).



3 - Unidade de Controle CU : Gere o fluxo de informação no CPU

- Normalmente todos os registos têm o comprimento de uma palavra.
- Uma palavra lida da memória, pode ser transferida para IR ou para os registos de dados Ri
- A ALU tem duas entradas de operandos. Em geral uma das entradas está também ligada ao barramento de dados.
- Os registos do CPU podem ser lidos ou escritos na memória.
- Em geral, os dados e resultados das operações da ALU podem também ser lidos e escritos na memória directamente.
- O registo PC fornece a indicação da instrução a executar pelo CPU. IR selecciona na unidade de controle (CU), a sequência de acções que permite a execução da instrução.
- Os periféricos normalmente comunicam com o CPU através do barramento de dados.
- A unidade de controle envia todos os sinais (ordens) que permitem os acessos às vias de comunicação e determinam as operações a realizar.

Conceitos de Programa e Software

Programa:

Conjunto organizado de instruções e dados que determinam a realização de dado objectivo num sistema de computação digital.

Software

Conjunto de todos os programas de um sistema de computação.

Tipos de Programas:

Sistema: Necessários à operação eficiente e agradável do sistema de computação (ou seja, os sistemas operativos).

Aplicação: Destinados a realizar tarefas específicas necessárias ao utilizador (processadores de texto, folhas de cálculo, jogos, etc).

Linguagens Máquina, Assembly e de Alto Nível

Linguagem Máquina

- O programa (instruções e dados) reside na memória a partir de determinada posição de memória (origem do programa).
- O programa utiliza os códigos binários correspondentes a instruções e o formato de dados reconhecidos pelo CPU (diz-se que está escrito em linguagem máquina).

Exemplo 1: Programa simples para ler o conteúdo da porta 11, adicionar-lhe o valor 21 e colocá-lo novamente na porta 11 (instruções máquina para o microprocessador Z80).

1101 1011	Leia para o registo A o conteúdo da porta
0000 1011	número 11 (00001011_2)
1100 0110	adicione ao conteúdo do registo A o valor
0001 0101	21 decimal (00010101_2)
1101 0011	coloque o conteúdo do registo A na porta
0000 1011	número 11 (00001011_2)
0111 0110	Termine a execução do programa (“Halt”)

É mais habitual a escrita dos códigos das instruções em sistema de numeração hexadecimal (mais compacto e confortável). Para o exemplo anterior ficaria:

```
DB  0B
C6  15
D3  0B
76
```

Código Fonte:

Quanto se implementa ou escreve um programa informático geralmente está-se a adoptar uma (ou várias) linguagem de programação. O processo de escrita do programa nessa linguagem de programação dá origem ao “código fonte” ou simplesmente “fonte” do programa. O código fonte consiste em todos os ficheiros que contêm as diversas partes do programa escritas numa determinada linguagem previamente seleccionada.

Visto que vulgarmente um programa é composto por vários ficheiros fonte, é normal ao conjunto de todos os ficheiros designá-los por projecto.

Ambiente Integrado de Desenvolvimento (IDE):

O processo de escrita do programa tipicamente faz-se recorrendo a um ambiente gráfico de desenvolvimento (como por exemplo o ambiente gráfico oferecido pelo Delphi para Pascal, ou pelo MS Visual Studio para C++ ou para Visual Basic) que acelera esse processo de desenvolvimento.

Geralmente esse ambiente IDE inclui ferramentas/procedimentos de teste e depuração do programa a desenvolver.

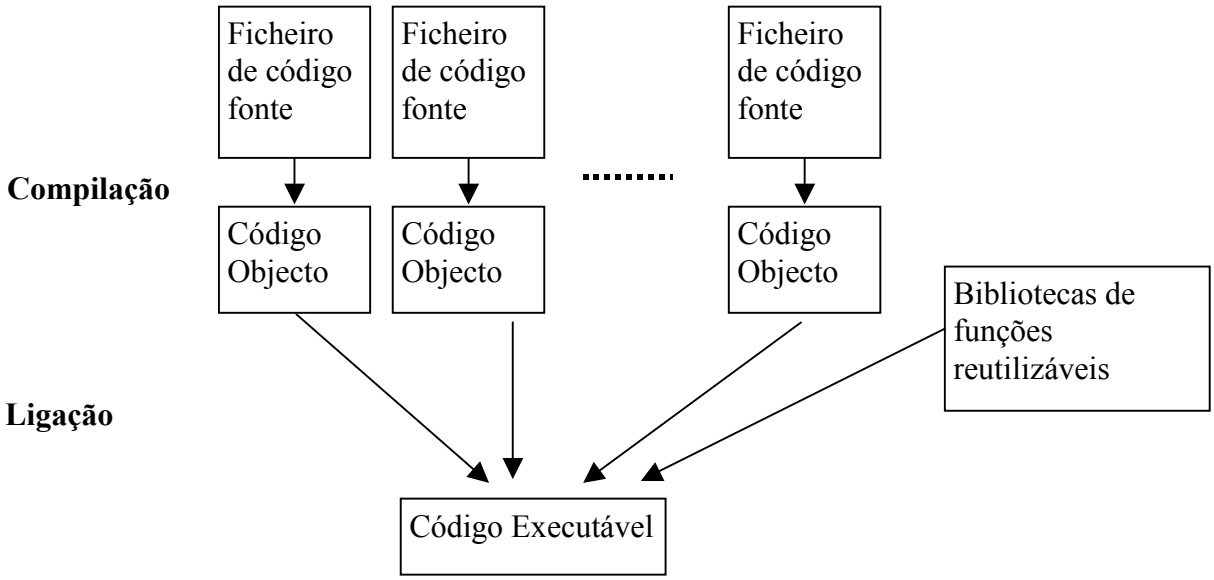
Depois do código fonte escrito a preparação e execução do programa vai depender do tipo de linguagem adoptada, isto é, se é compilada ou interpretada.

Linguagens compiladas

O código fonte de uma linguagem compilada é constituído por texto que não é directamente executável pelo computador.

Um programa pode estar distribuído por diversos ficheiros fonte.

A criação de um Programa Executável a partir de código fonte compilável geralmente faz-se a duas fases: 1º compilação (efectuada pelo programa compilador) e 2º ligação (efectuada pelo programa ligador).



O ficheiro de código executável pode ser executado ou corrido independentemente sem necessitar de nenhum programa externo. O ficheiro executável apenas pode ser executado na arquitectura computacional para a qual o compilador foi preparado.

Linguagens interpretadas

Numa linguagem interpretada o código fonte é executado directamente. No entanto precisa de um programa externo que vai transformar cada instrução da linguagem de alto nível em instruções que sejam entendidas pelo CPU. Esse programa é designado por **interpretador**.