Introdução à Computação

2º semestre 2004-2005

cursos: BQ, M, EFT, FQ, Q

---

aulas teóricas

1	15 Fev	Apresentação		
2	16 Fev	Computadores		
3	22 Fev	Memória		
4	23 Fev	PASCAL		
5	1 Mar	Variáveis, Write e WriteLn		
6	2 Mar	Atribuição, Constantes, Write formatado		
7	8 Mar	Read e ReadLn, Cálculo		
8	9 Mar	if ... then ... else, comparações		
9	15 Mar	Boolean álgebra, case ... of		
10	16 Mar	Ciclos I: For		
11	29 Mar	Ciclos II: While-Do Repeat-Until		
12	30 Mar	Programação Modular I Procedures		
13	5 Abr	Programação Modular II Funçoes		

mini testes

1		
2		
3		
4		
5		
6		
7		
8		
9		
10 11		
12 13		
14		
15		
16		
17 18		
19		
20		

aulas práticas

1	1,2 Mar	Windows 1 Windows 2 Internet	
2	8,9 Mar	Compilador Primeiros programas	soluções
3	15,16 Mar	Variáveis, Constantes WriteLn	soluções
4	29,30 Mar	Read, ReadLn, Atribuição, if ... then ... else	soluções
5	5,6 Abr	Case ... Of Ciclos I: For	soluções
6	12,13 Abr	Ciclos e Procedimentos	soluções
7	19,20 Abr	Matemática	soluções
8	26,27 Abr	Vários assuntos	soluções
9	3,4 Maio	Arrays / records	soluções
10	17,18 Maio	Vários assuntos alternativo	soluções soluções
11	24,25 Maio	Ficheiros	soluções
12	31-Maio, 1-Jun	Vários assuntos	soluções
13	7-8 Jun	Vários assuntos	

14	6 Abr	Funções matemáticas		
15	12 Abr	Âmbito das variáveis *Passing by value* / *Passing by reference*		
16	13 Abr	Programação recursiva		
17	19 Abr	Arrays		
18	20 Abr	Records		
19	19 Abr	Definir novos tipos		
20	27 Abr	Apontadores		
21	3 Maio	Ficheiros		
22	4 Maio	Algoritmos		
23	17 Maio	aula de dúvidas I		
24	18 Maio	Resumo		
	24 Maio	explicação de um exame		
24	25 Maio	Outras linguagens		
	31 Maio			
	1 Jun			
	7 Jun			
	8 Jun			

Avaliação:

regras de avaliação

exemplos de exames dos anos passados:
Frequência: 3 de Junho 2002: doc, pdf.
Exame época normal: 7 de Junho 2002: doc, pdf.
Frequência: 11 de Junho 2003 enunciado, solução

Exame época normal: 30 de Junho 2003 enunciado
Exame época normal: 25 de Junho 2004 enunciado, solução

exemplo de um outro exame: DOC, PDF

Horário:

aulas teóricas			
turno	*cursos*	*quando*	*sala*
IC-T1A	BQ, EFT, FQ, M, Q	Terça 12:00-13:00	CP Anf.C
IC-T1B	BQ, EFT, FQ, M, Q	Quarta 12:00-13:00	CP Anf.C

aulas práticas				
curso	*turno*	*quando*	*sala*	*docente*
BQ 2º ano	P3	Quarta 9:30-12:30	C1 0.23	Peter Stallinga
EFT 1º ano	P2	Terça 14:30-17:30	C1 0.23	João Lima
FQ 1º ano	P2	Terça 14:30-17:30	C1 0.23	João Lima
M 1º ano	P2	Terça 14:30-17:30	C1 0.23	João Lima
Q 1º ano	P2	Terça 14:30-17:30	C1 0.23	João Lima

Inscrições nos turnos das aulas práticas no secretaria 2 (esquerda) do edifício 2

Trabalho prático 2005:

English: click here
Português: clique aqui

Mirror sites:

pagina oficial na universidade: http://www.adeec.fct.ualg.pt/IC/

---

compilador pascal

o compilador DevPASCAL foi tirado deste site porque contem addware (Dialer.Gen)

Peter Stallinga, Fev. 2005

Introdução à Computação

Aula 1: apresentação 

Índice

Descrição e objectivos da cadeira

Esta é uma cadeira de introdução à computação e programação. Nas primeiras duas semanas de aulas iremos dar uma prespectiva global sobre as várias facetas do mundo da computação. Daremos as noções de hardware, software, sistema operativo, linguagens de programação, compiladores, programas de aplicação, a Internet e a sua utilização.

O resto das aulas vai incidir sobre os fundamentos de programação de computadores. Como linguagem de programação iremos usar a linguagem Pascal mas os conceitos que vão aprender aplicam-se a quase todas as linguagens de programação. No final da cadeira os alunos devem saber os fundamentos de programação e devem ser capazes de escrever programas simples. A cadeira não requer conhecimentos prévios na área de informática.

Nos apontes de aula e nas páginas na rede:

- Todos os blocos de texto em itálica são informação geral e não precissam de ser estudiados. Palavras isoladas em itálica estão escritas em Inglês. (por exemplo: *file* em vez de ficheiro)
- Qualquer texto com fonte com esta: `"fixed width"`, representa código em Pascal.

Professores

NOME	E-MAIL	HORÁRIO DE DÚVIDAS
Peter Stallinga	pjotr@ualg.pt	2ª e 6ª 9:30-11:30 Ed. 1 - sala 2.68 ou 2.78
João Lima	jlima@ualg.pt	Quarta-feira 16:30-18:30 Quinta-feira 10-12 Ed. 1 - gabinete 2.63

Os alunos devem tirar as suas dúvidas preferencialmente nas aulas. Só se a dúvida persistir é que devem então contactar os docentes no horário acima referido.

Bibliografia

- Apontamentos dados nas aulas estarão disponíveis na página web da cadeira, http://diana.uceh.ualg.pt/IC e *mirror site* http://w3.ualg.pt/~pjotr/ic
- "Programação em Pascal", Byron S. Gottfried, Schaum, McGraw Hill, ISBN 9729241589
- ("Schaum's Outline of Programming with Pascal", Byron S. Gottfried, ISBN 007023924X)
- "Pascal Programming Made Simple", P.K. McBride, ISBN: 0750632429
- "Learn Pascal" e "Learn Pascal in Three Days", Sam A Abolrous, ISBN: 155622706X e ISBN: 1556228058 resp.
- http://www.devq.net/pascal/

Regras de avaliação (2004-2005)

- Não há testes (frequências).
- As aulas práticas são obrigatórias.
- No fim do semestre os alunos devem entregar um trabalho. O enunciado do trabalho será dado em Maio.
- Os alunos podem ser chamados a qualquer altura para defender o trabalho (a afixação na porta do meu gabinete é só para comunicar as notas provisórias). Quem não consegue defender o trabalho obterá a classificação de 0 valores.
- Serão admitidos ao exame os alunos com uma nota do trabalho igual ou superior a 10 e uma assiduidade nas aulas práticas não inferior a 2/3 das aulas (9 em caso de 13 aulas, 8 em caso de 12 aulas).
- O trabalho prático conta 20%, o exame conta 80%.
- Nota mínima do trabalho prático: 10.0
- Nota mínima do exame: 9.0
- Nota final mínima: 9.5
- Os alunos repetentes já admitidos ao exame num ano passado estão dispensados de fazer o trabalho ou assistir às aulas práticas. Neste caso, o exame conta 100%. A nota do trabalho do ano passado não conta para a nota final.
- Os alunos repetentes podem optar por entregar o trabalho de este ano. A entrega significa aceitação da avaliação 20%-80%.
- Qualquer forma de fraude causa directamente a anulação de qualquer qualificação obtida e possivelmente um processo disciplinar no Conselho Pedagógico.

Data e local dos exames

a anunciar mais tarde.
(salas a anunciar oportunidamente)

Fraude

Quem copiar, deixar copiar, ou fizer qualquer outro tipo de fraude durante os momentos de avaliação, terá zero valores e leva um processo disciplinar para o Conselho Pedagógico.

Programa

Apresentação, descrição e objectivos da cadeira.
Noções introdutórias sobre computadores: tipos de computadores, componentes de um computador, características de um computador, sistema operativo, linguagens de programação, compiladores, programas de aplicação.
Noções e utilização da Internet.
Noções básicas de programação: constantes, variáveis, expressões, operadores, instrução de atribuição, instruções de input/output, funções pré-definidas.
Noções de programação estruturada: sequência, selecção, iteração.
Instruções de selecção: if, if-else, case.
Instruções de iteração: for loops, while loops, do-while loops, repeat-until loops.
Funções e procedimentos.
Vectores e matrizes: arrays de uma e duas dimensões.
(Caracteres e cadeias de caracteres.)
Records e definição de noos tipos de dados.
Noção de algoritmo. Algoritmos de ordenação simples. Algoritmo de pesquisa sequencial e de pesquisa binária.
(Apontadores. Passagem de parâmetros.)

Ficheiros de texto.

Recomendação para os alunos

A programação de computadores não é difícil. Pelo contrário, é uma tarefa relativamente fácil e divertida que envolve apenas meia dúzia de conceitos. No entanto, requer um tipo de raciocínio a que as pessoas normalmente não estão muito habituadas. Como tal, trata-se de uma tarefa que exige bastante prática e por isso recomendo que treinem bastante fora do horário das aulas. Se fizerem isso ao longo do semestre, nem sequer precisam de estudar para os testes e passarão à cadeira com boa nota quase de certeza.

Peter Stallinga. 17 fevereiro 2004

Aula 2: Computadores

Translated by Luís Pereira

O que é um computador? De acordo com a Encyclopedia Columbia é "*um dispositivo capaz de executar um conjunto de operações aritméticas e lógicas. Um computador distingue-se de uma máquina de calcular, como por exemplo uma calculadora electrónica, por ser capaz de guardar um programa (para que possa repetir as suas operações e tomar decisões lógicas), devido ao número e complexidade das operações que podem ser executadas, e pela possibilidade de executar, guardar e aceder dados sem intervenção humana.*"

Existem vários tipos de computadores. Hoje, quando falamos em computadores, queremos dizer computadores digitais. Ao longo da história, também existiram computadores mecânicos e analógicos (electrónico).

O Primeiro computador mecânico foi criado por Charles Babbage no principio do século 19 (por volta de 1815)! Por esta razão, Babbage é frequentemente chamado "O pai da computação". Famosa é também a *Difference Engine*. Se quer saber mais sobre Charles Babbage, clique aqui.
O primeiro computador electrónico, processando dados no formato digital foi o ENIAC pouco antes da segunda grande guerra (1939). O primeiro computador comercial disponível foi o UNIVAC (1951). Nessa altura pensava-se que uma mão cheia de computadores, distribuídos pelo mundo, seriam suficientes para efectuarem todos os cálculos necessários. Hoje em dia, a grande maioria das pessoas no Ocidente tem um computador em casa, muito mais poderoso que esses primeiros computadores.

Os computadores estão organizados por tamanho e número de pessoas que os pode usar ao mesmo tempo:

Supercomputadores	Máquinas sofisticadas desenvolvidas para executarem cálculos complexos a grande velocidade; são utilizados para modelar grandes sistemas dinâmicos, como por exemplo os padrões do tempo. Um exemplo é o Cray SV2 (ver figura), que tem as dimensões de uma sala média 
Mainframes (Servidor principal)	Os maiores e mais poderosos sistemas para fins comuns, são desenvolvidos para as necessidades de uma grande organização, servindo centenas de terminais ao mesmo tempo. Imaginem companhias de seguros com todos os seus documentos internos partilhados através de uma rede. Todos os empregados podem aceder e editar os mesmos dados.
Minicomputadores	Embora algo pequenos, também são computadores multiutilizador, pensados para satisfazer as necessidades de uma pequena empresa servindo até uma centena de terminais.

Microcomputers	Computadores servidos por um microprocessador, são divididos em computadores pessoais e estações de trabalho. Computadores pessoais são os que a maioria das pessoas têm em casa. Exemplos: IBM PC e compatíveis com um microprocessador como o Intel Pentium IV, 1.5 Ghz. Computadores portáteis são uma variante dos micro-computadores. Apple Macintosh incorporando processador da família Motorola.
Processors (processador dedicado)	Muitos electrodomésticos, como as máquinas de lavar e fornos, contêm um pequeno processador para controlar o equipamento. São computadores muito pequenos que foram programados nas máquinas ainda na fábrica e não podem ser programados pelos utilizadores. Podem assim não ser considerados computadores, mas convém ser referidos. Alguns electrodomésticos mais avançados, como receptores de satélite ou sistemas de cinema, correm programas muito sofisticados que seguem as regras a seguir apresentadas nesta aula.

Nota: Considerando a velocidade de avanço da tecnologia, podemos dizer que "*os supercomputadores de hoje são os microcomputadores(pessoais) de amanhã*"

Hardware vs. software

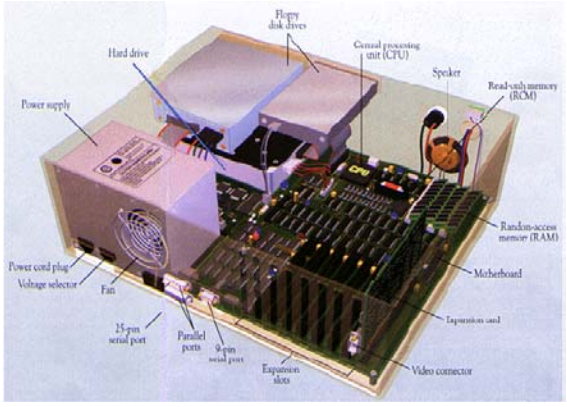
Dois termos que importa distinguir são hardware e software. Hardware é tudo o se pode tocar e sentir. Sofware são os programas que correm no hardware. Alguns exemplos a seguir.

Todos os computadores são constituídos pelo menos por:

1) Um processador
2) Memória para armazenar o programa
3) Um dispositivo de entrada: (ex.: teclado, rato, porta de ligação a Internet.)
4) Um programa
5) Um dispositivo de saída(ex.: monitor, impressora, fax, porta de ligação a Internet)

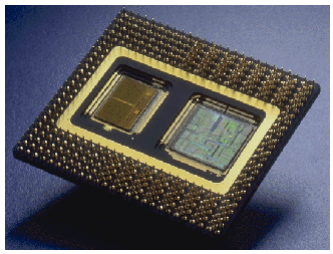
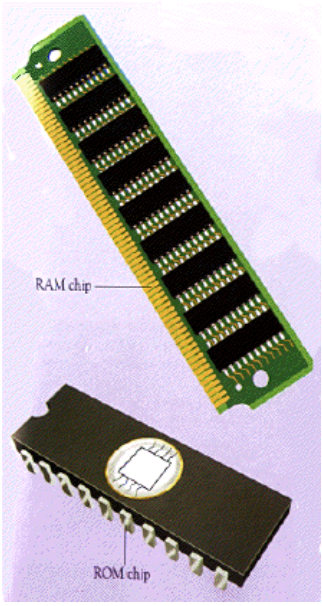
A maioria dos computadores também têm:

6) Um dispositivo de entrada, para mudar o programa, introduzir novos dados para serem processados ou controlar os processos correntes.



Vejamos alguns componentes do computador

Tabela: elementos de hardware

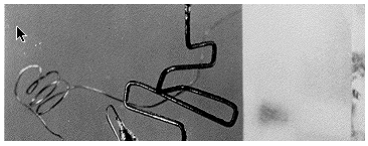
Rato		dispositivo de entrada	Para controlar os processos do computador
CPU		Processador	Central Processing Unit (Unidade Central de Processamento) Peça que faz o trabalho. Calcula, controlas os dados, etc.
Joystick		dispositivo de entrada	Controlador para jogos
Teclado		dispositivo de entrada	Para dar instruções ao computador ou introduzir dados para processamento
Memória		armazenamento	Guardar programas e dados para processamento
Monitor		Dispositivo de saída	Mostra os resultados dos processos
Impressora		Dispositivo de saída	Imprime os resultados dos processos

Modem		Dispositivo de entrada/saída	MOdulador-DEModulador comunicar com outros, computadores pela linha telefónica
Placa de rede		Dispositivo de entrada/saída	Comunicar com outros computadores através de uma rede
Disco rigido/disquete		armazenamento	Guardar dados e programas num formato não volátil (os dados permanecem quando se desliga o PC)
CD-ROM		dispositivo de entrada	Carregar programas ou dados na memória
Placa de som		Dispositivo de saída	Tocar música ou outros sons
Scanner		dispositivo de entrada	Digitalizar imagens

Camadas físicas e lógicas de um PC

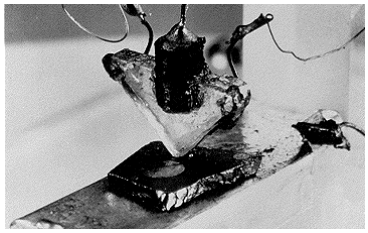
Ao nível mais baixo que podemos chamar de **Físico**, os electrões são responsáveis pela condução electrica dos materiais. Os materiais usados nos computadores chamam-se semicondutores, o que quer dizer que eles têm uma resistência entre os metais (como o cobre e ouro) e os isolantes (como o vidro e o plástico).

No nível seguinte temos a **Electrónica**. A



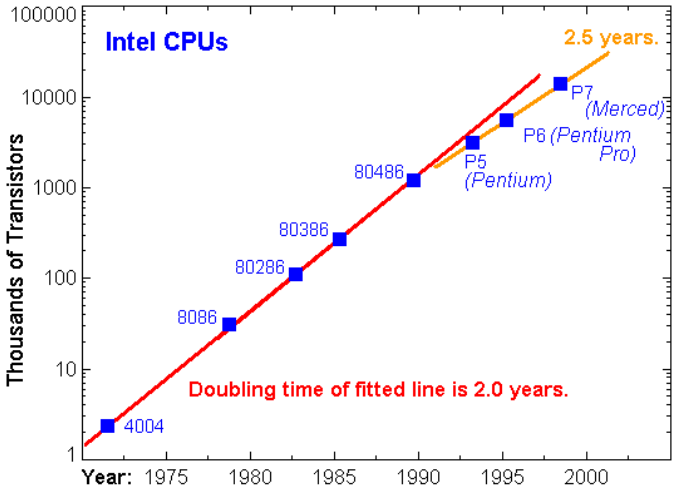
Fotografía do primeiro transístor, tal como foi inventado nos

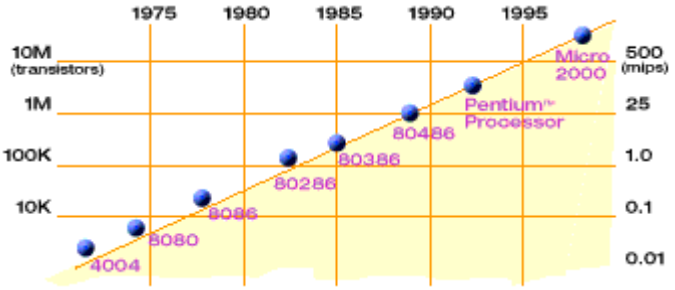
electrónica liga materiais com propriedades diferentes transformados num componente para sua utilização. Note por exemplo o díodo que conduz corrente numa só direcção (na imagem acima), e o transístor,
cuja conductividade é controlada por voltagem exterior.

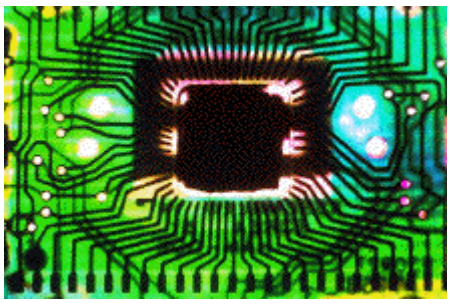


Laboratórios Bell em 1947

A seguir encontramos o nível da **Electrónica Digital**. São utilizadas as chamadas portas ("*gates*"): OR, NOR, AND, NAND, XOR, só para mencionar algumas. Estas portas são feitas de componentes de electrónica básica tais como transistores. As portas são os elementos fundamentais que estão na base dos computadores digitais. Temos que nos lembrar que todos os cálculos são feitos a este nível. Quando adicionamos 2 +2, algures no PC as portas estão mudando e processando cálculos do tipo "1 OR 1 = 1". Outro componente da electrónica digital é a memória. Também são formadas por transístores (e capacitadores no caso da RAM dinâmica) e podem guardar temporariamente informação.







O nível seguinte é o dos **Circuitos Integrados**. Nestes circuitos, milhões e milhões de portas estão ligadas o que permite a execução de programas complexos. Desde que foi manufacturado o primeiro circuito integrado, acontece que o número de transístores num único IC se vem duplicando de dois em dois anos aproximadamente. A esta regra chama-se *lei de Moore* que ainda é válida, embora os limites físicos estejam á vista. Ver a figura à esquerda.

Para fazer uma comparação popular. Se o mesmo avanço se tivesse verificado na industria automóvel, um carro moderno poderia andar a 5 milhões de km/h, consumia uma gota de combustível aos 100.000 km e poderia levar sentadas 10000 pessoas.

No nível seguinte começa a programação real. Começamos com programas binários, ou (em formato legível) **Micro-assembler**. Isto é programação directa no processador: coloca endereço xxx no registo de endereço, active linhas de endereço, esperar xx ns, adicionar registo X ao registo Y, etc. ("registos" são pequenas unidades de memória dentro do processador)

No nível seguinte temos a **Linguagem da Máquina**. Um exemplo:
101000100000101000
O que pode significar: põe o conteúdo do endereço 0000101000 no registo A. Esta linguagem é quase impossível de lêr para os humanos. Por isso, foram inventados os Macro-assembler o que já é um pouco

melhor.
Mas lembra que quando um programa vai correr, o código da linguagem da máquina vai ser carregado (sem mais traduições) na memória e executado. Ficheiros com nomes finalizados (chamada extensão) com as letras ".exe" e ".com" são deste tipo (como nos sistemas operativos MS-DOS e Windows).

Para o nível seguinte, que já se parece mais com programação real, temos o **Macro-assembler**. Neste nível nós instruimos o processador para executar os pequenos programas escritos em micro assembler. Estes tem por exemplo a forma seguinte:

```
ADDA $2050
```

ou numa forma menos legível (pelo menos para as pessoas);

```
1010001000001010000
```

o que significa: "adicionar o conteúdo da memória do endereço $2050 ao registo A". Linguagens de programação modernas como o Pascal e C são traduzidas para esta forma com a ajuda de compiladores.

No nível seguinte temos finalmente as **Linguagens de Programação**. Estas linguagens são regularmente chamadas de "linguagens de quarta geração", porque evoluíram das linguagens mais antigas como o assembler, etc. Muitas destas linguagens foram inventadas durante os anos 1960 e 1970. Em 1995 existiam cerca de 2500 linguagens de programação diferentes. (para quem estiver interessado, ver http://cui_www.unige.ch/langlist). Para programadores profissionais existe o C e C++, para aplicações simples o BASIC. Para fins educacionais foi inventado o Pascal, especialmente para ensinar ideias de programação.

Presentemente, novas gerações de linguagens evoluem. Todas elas envolvem o conceito de **Programação Orientada a Objectos**, um conceito que não iremos usar nas nossas aulas, mas que se tornou indispensável nos ambientes de programação modernos. Podemos chamar-lhes a "quinta geração de linguagens"

As linguagens de programação modernas são flexíveis. Podemos escrever uma grande variedade de aplicações nestas linguagens. Podemos, por exemplo, escrever um programa que simule o funcionamento de um díodo, ou calcular um transístor ao nível físico. Completamos assim o ciclo; podemos utilizar o computador para fazer os componentes básicos e melhores e mais rápidos computadores.

Exemplos:

BASIC	`IF A=20 THEN PRINT"Hello World"`
PASCAL	`if (a=20) then writeln('Hello World');`
C	`if (a==20) printf("Hello world\n");`
FORTRAN	`IF (A .EQ. 20) PRINT ,'Hello World'`

Não esquecer que se escrevermos em PASCAL

```
writeln('Hello world');
```

e corrermos o programa, estamos de facto a controlar o fluxo de electrões ao mais baixo nível. Isto pode-vos dar uma boa ideia de controlo ...

Compiladores

Como foi dito antes, as linguagens modernas de programação têm de ser traduzidas da forma que os utilizadores compreendem para a linguagem dos computadores. Quando escrevemos

```
writeln('Benfica - Sporting   3 - 0');
```

Isto tem de ser traduzido em

```
MOVAI $0102       ; carregar 'B' no registo A
MOVAO $1245     ; mover o conteúdo do registo para a placa de vídeo
```

ou, a um nível mais profundo

```
00110111000111111100011110100011111
```

Para este propósito existem compiladores. Eles traduzem o que é legível para as pessoas para a o que é executado pelo computador Quando começamos um ficheiro contendo o nosso programa meuprograma.pas, traduzimos com o compilador para um ficheiro chamado meuprograma.exe que pode ser executado escrevendo só meuprograma e o resultado aparecerá no ecrão quando tudo correr bem.

Benfica - Sporting 3 - 0

Na maioria das versões modernas das linguagens de programação, iremos trabalhar no chamado IDE (integarted development environment - ambiente integrado de desenvolvimento), o que significa que podemos escrever um programa e com um simples toque podemos compilar o programa, ver os erros da nossa escrita, e, caso não existam erros durante a compilação, executar o programa e ver os resultados. Esses ambientes proporcionam grande velocidade de desenvolvimento de software, mas não nos devemos esquecer que o compilador está traduzindo o programa para nós. Iremos discutir os compiladores e o seu uso mais tarde nas aulas práticas.

Sistemas Operativos

Sistemas Operativos são programas que estão constantemente a correr no nosso computador e estão interpretando os comandos que nós damos. Por exemplo, quando nós queremos "correr" o nosso programa, podemos escrever o seu nome ou premir no seu ícone, ou de qualquer outra forma. O sistema operativo vai então:

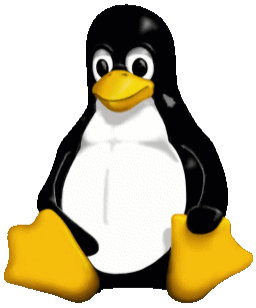
1) carregar o programa do disco para a memória
2) executa-lo

Quando o nosso programa está terminado, é removido da memória novamente, mas o sistema operativo continua a correr, esperando pela próxima instrução. De facto, um computador por si só não faz mais nada que verificar se nós digitamos ou premimos alguma tecla. Que desperdício de energia....

O Sistema operativo mais famoso terá sido possivelmente o MS-DOS da Microsoft. Este era uma sistema operativo em linha de comando, quer dizer, tinha que se escrever as instruções no computador através do teclado. Por exemplo DIR C:\
Mais tarde, uma interface gráfica foi adicionada ao MS-DOS e foi chamada Windows. Por baixo dela, ainda tínhamos o mesmo sistema de linha de comandos do MS-DOS , mas os clicks do rato nos ícones e objectos traduziam-se em comandos. Podiamos premir numa 'pasta' e ver o seu conteúdo. Clicar numa pasta era igual a escrever "DIR" mas com os resultados apresentados em modo gráfico. Ao longo dos anos, Windows tornou-se mais avançado e nos dias de hoje é um sistema operativo de multitarefa (o que quer dizer que mais do que um programa pode ser executado ao mesmo tempo) e é utilizado pela maioria das pessoas mundialmente.

Por causa do monopólio detido pela Microsoft, os 27% de acções que o co-fundador Bill Gates representam 20 biliões de dólares (20.000.000.000 dólares) em 1995. Em 2000 subiu para 65 biliões. Note que se trata de 10 dólares (o mesmo valor sensivelmente em euros de hoje) por cada pessoa na terra, seja ele americano ou um chinês numa aldeia remota da China.

Alternativas para o Windows são o Unix/Linux, que tem a vantagem de ser gratuito, e o MacOS para os Macintosh.



.

Mini teste:

Para testar os conhecimentos adquiridos nesta aula, prima aqui para um teste imediato. Note que esta não será a forma do teste final!

Peter Stallinga, Universidade do Algarve, 6 fevereiro 2002

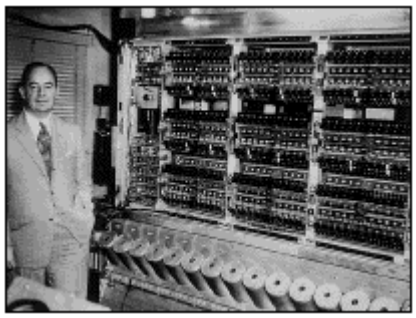
Aula 3: Unidades de Informação / Memória

Translated by Luís Pereira

Como descrito na aula anterior, a memória é uma parte essencial do computador. Guarda

- o programa
- os dados com que o programa está trabalhando

Nota: A ideia de separar o programa dos dados com que está a trabalhar e o hardware do software (ou "a máquina" e o "programa") vem de Von Neumann. Ele desenhou o primeiro computador electrónico capaz de correr um programa flexivel (1940-1952). Todos os computadores modernos são computadores Von Neumann.



Antes de escrever programas, é útil ver mais de perto a memória.

A memória está cheia de **informação**. Esta informação pode ser código de programação ou dados.

BIT

A menor quantidade de informação é um bit. Um bit pode conter a informação do tipo "TRUE ou FALSE". Por exemplo, pode conter a informação
"O aluno pagou as propinas, sim ou não?",
"O aluno pode fazer a frequência, sim ou não?",
"*x* é maior que *y*, sim ou não?".
Temos que nos lembrar que, ao nível da electrónica, o computador está calculando com estes bits de informação. Na aula anterior vimos como os componentes electrónicos digitais ("AND gates", etc) geriam estes bits de informação. Para estes componentes electrónicos, existem dois niveis: 0V e +5V (ou qualquer par de níveis de voltage). Na nossa linguagem, podemos chamar 'TRUE' e 'FALSE', ou '1' e '0', ou 'green' e 'red', ou qualquer par de nomes simbólicos que queiramos atribuir. Porque um bit de informação só pode ter dois valores, podemos chama-lhe unidade binária (**binary** unit). Como todas as unidades de informação derivam do bit, chama-mos ao computador calculadora binária. Embora se possam construir computadores baseados noutras unidades de informação (como por exemplo ternary or quaternary), todos os computadores modernos são do tipo calculadoras binárias.

Níveis electrónicosde AND gates	0 V	+5 V
binary	0	1
logical	FALSE	TRUE
bicolor	verde	vermelho

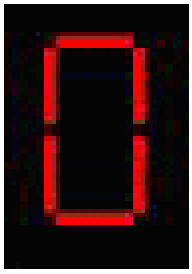
Estes principios podem ser misturados á nossa vontade. Por exemplo, PASCAL usa o principio 'TRUE' e 'FALSE' para calculos lógicos, enquanto o C usa '1' e '0'.

NIBBLE

A seguinte unidade de informação é o **nibble**. Trata-se de um conjunto de 4 bits. Nestes 4 bits podemos por exemplo guardar informação do tipo 0..9. Nibbles são usados em muitos mostradores digitais, como por exemplo relógios de alarme, em que cada digito é um nibble. Podemos chamar a este código binary-coded-decimal (**BCD**):

binary	*BCD*
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7

binary	*BCD*
1000	8
1001	9
1010	not used
1011	not used
1100	not used
1101	not used
1110	not used
1111	not used



examplo dum display de LED

Pela tabela podemos ver que para o código binário

abcd

o código decimal é

a*8 + b*4 + c*2 + d

ou, mais genericamente:

$a*2^3 + b*2^2 + c*2 + d$

Esta, iremos ver, é sempre a relação entre binários e decimais.

Note também que algumas das combinações possiveis de bits não são usadas em BCD. Uma forma de representar estes quatro bits de informação com todas as combinações possiveis é o sistema **hexadecimal**. A combinação de bit de '1010' até '1111' são representadas pelas letras de A a F. Na representação hexadecimal temos a seguinte tabela de traduçã

binary	*hexa-decimal*
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7

binary	*hexa-decimal*
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

Como vemos todos os 16 binários têm um correspondente no sistema hexadecimal. O sistema hexadecimal é frequentemente usado na tecnologia dos computadores. Como exemplo: 21F no sistema hexadecimal é igual a $2*16^2 + 1*16^1 + 15*16^0 = 2*256 + 1*16 + 15*1 = 543$.

BYTE

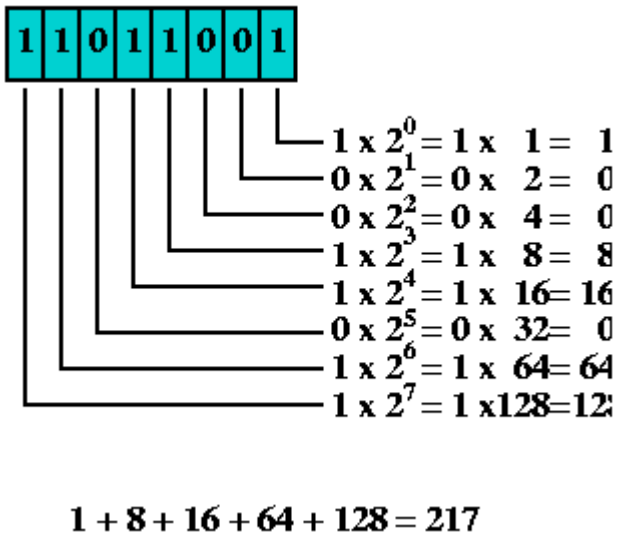
A seguinte unidade de informação é o byte. Byte é a combinação de dois nibbles e portanto de 8 bits. Nestes podemos guardar numeros de 0..255 porque

$00000000 = 0*2^7 + 0*2^6 + 0*2^5 + 0*2^4 + 0*2^3 + 0*2^2 + 0*2^1 + 0*2^0 = 0$

$11111111 = 1*2^7 + 1*2^6 + 1*2^5 + 1*2^4 + 1*2^3 + 1*2^2 + 1*2^1 + 1*2^0 =$
$= 128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 =$
$= 255$

another example:

$11011001 = 1*2^7 + 1*2^6 + 0*2^5 + 1*2^4 + 1*2^3 + 0*2^2 + 0*2^1 + 1*2^0 =$
$= 128 + 64 + 0 + 16 + 8 + 0 + 0 + 1 =$
$= 217$



De outra forma, um byte pode representar todas as letras do alfabeto, em maiusculas ('A' .. 'Z') e minusculas('a' .. 'z'), mais todos os digitos '0' .. '9', alguns caractere especiais com '{', '}', '(', ')', space, etc, e outras coisas como códigos de controle. A mais usual de o fazer é através do **ASCII** (American Standard Code of Information Interchange) no qual por exemplo, 'A' é 65 (decimal), ou 01000001 (binário), ou 41 (hexadecimal). Outros exemplos são:

binary	*decimal*	*hexadecimal*	*ASCII*
01000001	65	41	'A'
01000010	66	42	'B'
01100001	97	61	'a'
00010000	32	20	' '

Memory

Em muitos computadores, o byte é a unidade mais pequena que pode ser 'endereçada'. Para percebermos esta ideia, vejamos como a memória está organizada. Imaginem a memória como uma rua (muito grande) com casas. Cada casa tem um endereço. Se quisermos colocar algo na casa, ou tirar, temos que indicar o endereço da casa. No computador, a memória toma o lugar da rua e o byte toma o lugar da casa. Em cada 'casa', vivem 8 bits, ou um byte.
As we will se later, some 'people' (units) are very big and occupy two or even more adjacent houses. These 'people' are called 'integers', 'words', 'reals', etc. Still, addresses are in most computers 1 byte and thus 8 bits apart. (*Note [not needed to study]: especially in supercomputers the distance between two addresses can be much longer than a byte, for example 63 bits instead of 8. We call this distance the 'word length' of a*

computer.)

Notem que as três unidades descritas até agora são também unidades de comida em Inglês. Bit, Nibble, Byte. As seguintes já não seguem este principio.

Tamanho de Memória dos Computadores

> Muitas vezes podemos ler em publicidade:
> Computador, com processador 1.7 GHz , 256 MB RAM, 40 GB harddisk, disquete 1.4 MB

1.7 GHz specifica a velocidade do processador (CPU - central processing unit). 1.7 GHz significa que pode fazer 1.7 milhões de instruções simples por segundo.
(Como a maioria dos comandos dados ao processador precisam de mais do que uma instrução, o numero efectivo de comandos por segundo é inferior. Por exemplo a adição de dois numeros de virgula flutuante pode consistir em 1) carregar o primeiro numero da memória, 2) carregar o segundo da memória, 3) adicionar o numero (possivelmente em vários passos), 4) guardar o resultado em memória. No entanto, a velocidade global do computador é determinada pela velocidade do processador e a velocidade com que ele pode carregar os dados da memória.)

Os outros numeros determinam o tamanho da memória do computador (RAM = Random-access memory), o disco e a disquete respectivamente, em numero de (B). Para dar uma ideia do que estes numeros representam, analisemos calmamente:

BYTE: A unidade básica para descrever a memória é um byte (B). Como foi dito antes, um byte é suficiente para conter uma letra, ou um numero de 0 .. 255.

KILOBYTE: 1024 bytes são um kilobyte (kB). Em ciência, 'kilo' quer dizer 1000, um numero redondo no sistema decimal. Para os computadores 'kilo' representa um pouco mais, 1024. Por ser baseado em 2^{10} que corresponde a 1024 ou em binário 10000000000. Para dar uma ideia de quanto é um megabyte: uma página de texto A4 tem aproximadamente 4 kB.

MEGABYTE: 1024 kilobytes são um megabyte (MB). É igual a 1024 x 1024 bytes, ou 1048476 bytes. Para dar uma ideia da dimensão de um megabyte: 250 páginas de texto, ou digamos um livro. A maioria das disquetes têm 1.4 MB seria suficiente para guardar um livro com 350 páginas. (só texto sem imagens etc.)

GIGABYTE: 1024 megabytes são um gigabyte (GB). Seria suficiente para guardar uma boa biblioteca com milhares de livros A maioria dos CD têm 650 MB (0.65 GB), suficiente para guardar uma pequena biblioteca

Os discos modernos têm por volta dos 40 GB. Podem armazenar uma grande biblioteca, cerca de 40.000 livros.

TERABYTE: 1024 gigabytes são um terabyte. Embora discos deste tamanho não existam ainda, algumas empresas têm sistemas de computadores com muitos discos cujo total de espaçõ em disco é da ordem dos terabytes. Seria suficiente para guardar todos os livros do mundo.

Para dar uma ideia do espaço em disco existente no mundo imaginem: existem 500 milhões de utilizadores. Em média cada disco tem 10 GB. Faz um total aproximado de 5.000.000.000.000.000.000 bytes. Uma pessoa levaria mais de um bilião de anos para ler 10 livros por dia para ler toda essa informação!

Mini teste:

Para testar os conhecimentos adquiridos nesta aula click aqui para fazer um teste online. Atenção que esta **não** será a forma do teste final!

Peter Stallinga, Universidade do Algarve, 13 fevereiro 2002

Aula 4: Introdução ao PASCAL

PASCAL

Pascal é a sigla utilizada para "Program Appliqué à la Selection et la Compilation Automatique de la Literature", é uma linguagem de programação de alto nível. Criada por Niklaus Wirth no ano de 1960 com o objectivo de auxiliar o ensino de programação computacional. Ainda hoje continua a ser utilizada em universidades e não só, sendo esta uma linguagem bastante apropriada para ensinar. Muitos dos programadores profissionais, utilizam hoje em dia linguagens tais como C ou C++.
Pascal era o nome dum importante matemático francês do século XVII - Blaise Pascal.

Blaise Pascal (1623-1662)

Filósofo e matemático Françês. Contribuiu para o desenvolvimento na área de hidráulica, cálculo e teoria matemática de probabilidade. A sua maior invenção foi provavelmente o "Triângulo de Pascal":
Cada número é a soma dos dois números imediatamte em cima de si, esquerda e direita tal como ilustra a figura:

```
      1
    1   1
  1   2   1
1   3   3   1
1   4   6   4   1
```

Início

Um programa é uma sequência de instruções, que informam o computador sobre uma tarefa específica que se quer concretizar. A maioria das linguagens de programação modernas apresentam um formato bastante legivel de fácil percepção, assemelhando-se ao Inglês corrente, facilitanto aos humanos a leitura dos programas assim como a sua escrita, contrastando com as primeiras linguagens de programação, que se apresentavam como um tipo de linguagem próxima da linguagem máquina, tal como ilustra o exemplo em linguagem assembler (aula 2). PASCAL é a linguagem que mais se assemelha com a linguagem natural humana o que a torna uma linguagem de eleição no ensino da arte de programação.

Todos os programas em PASCAL têm o mesmo formato, tal como é mostrado em seguida:

```
PROGRAM Title;
begin
  program_statement_1;
  program_statement_2;
          |
  program_statement_n;
end.
```

Vamos dar uma vista de olhos neste programa.

- Todos os porgramas são iniciados com a palavra **PROGRAM** seguida pelo título do programa
- **begin** define o ponto de iníciuo do programa
- **end.** define o fim dum ficheiro e por sua vez o fim do programa.É de notar o ponto final "." no fim.O mesmo só é utilizado para o comando "end" do final do programa.Todos os outros comandos são finalizados com ponto e vírgula ";".
- Entre os comandos "begin" e "end" serão colocadas as intruções que formam um bloco.Todas as intruções entre estes dois comandos são tratados posteriormente pelo compilador como uma instrução só.
- Entre "begin " e "end " podemos colocar as nossas próprias instruções. Estas intruções podem ser intruções que o PASCAL já conhece, ou intruções que nós iremos definir (chamam-se procedimentos e funções, o que será discutido mais tarde).
- PASCAL não é um caso sensitivo: "Program", "PROGRAM", "program", etc, são interpretadas da mesma forma , ou seja o PASCAL não diferencia letras maiúsculas e minúsculas.

Identificadores

Identificadores, tal como o nome indica, servem para identificar algo. Podem ser o nome de **programas,** nome de **procedimentos , funções** e nomes de **variáveis** e **constantes.** Mais tarde será visto nas aulas.Tal como na maioria das linguagens, nomes de identificadores têm algumas restrições:

- Deverão começar por uma letra. "PROGRAM 20hours;" não é permitido
- Seguido por qualquer combinação de letras, dígitos ou caractér underscore "_".
- Espaços não são permitidos, tal como caracteres como "(", "{", "[", "%", "#", "?", etc, excepto "_", isto porque estes caractéres são utilizados em PASCAL para outro tipo de coisas. São chamados **caracteres reservados.**

```
{ } [ ] ( ) - = + / ? < > . , ; : ' " ! @ # $ % ^ & * ~ ` \ |
```

- Identificadores não podem ser igual ás **palavras reservadas** do PASCAL, tal como "program", "begin", "end". Notar que identificadores tais como "program1", ou "program_" são permitidos, sendo que o seu uso é desaconselhado para prevenir confusão que possa vir a ser gerada nos nomes. Nota: em muitos ambientes de programação (tal como Turbo Pascal 7), o uso de palavras reservadas é anunciado porque estas mesma palavras, quando são escritas aparecem com cor diferente.
- Escolha bem os seus identificadores. Quando um programa serve para calcular taxas, chame-o por exemplo"PROGRAM Taxas;" e não "PROGRAM program1;". Apesar de não ser errado usar este nome a sua escolha não é muito inteligente, pois desta forma não é identificado aquilo que o programa faz. Usando identificadores indicados ajuda outras pessoas a entederem o seu programa, ou voçê mesmo quando passou muito tempo sem mexer nesse programa.
- O comprimento minimo do identificador é 1, o máximo comprimento é 255. Faça uso da possibilidade de utilização de nomes extensos, mas lembre-se também que nomes muito grande poderão tornar o programa pouco perceptível. Tal como o provérbio diz nem 8 nem 80, logo tente ser claro e sucinto nos nomes escolhidos. Qual dos seguintes nomes voçê acha mais adequado:

```
r := r + a;
money := money + interest;
themoneyintheaccountofpersonwithnameJohnson :=
themoneyintheaccountofpersonwithnameJohnson +
thecurrentinterestrateatthetimeofthiswriting;
```

- They are not case sensitive, "i" is equal to "I", etc. To make programs more readable, follow a

convention all through your program(s). The most often used convention is lowercase for variables and UPPERCASE for CONSTANTS.

Programação estruturada

A coisa mais importante na programação é escrever duma forma clara, lógica e programas estruturados. Aqui vêem alguns avisos:

- Utilize nomes significativos para as variáveis, procedimentos e funções
- Utilize identação. Compare os programas a seguir ilustrados:

```
PROGRAM Myprogram;begin writeln("Hello world!");end.
```

e

```
PROGRAM MyProgram;

begin
  writeln("Hello world!");
end.
```

Ambos os programas fazem exactamente o mesmo, mas o segundo é muito mais legível. A diferença é:

  - Apenas utiliza uma instrução por linha
  - Usa identação: Coloca 2 espaços extras no início da linha todas as vezes que estamos num nível abaixo na estructura.
  - Separação de blocos de texto (funções e procedimentos) com linhas em branco.
- Evitar o uso de intruções "goto label". Com estas instruções, o programa rapidamente começará a parecer-se com esparguete.

Comentário: Sendo o PASCAL muito próximo da língua inglesa, o programa por si só deverá auto-explicar-se. Em zonas onde a ideia do programa não é muito clara para o programador, deverá recorrer á utilização de comentários. Em PASCAL os comentários são colocados entre chavetas "{ }". O compilador não interpreta a informação contida a seguir à primeira chaveta "{" até à chaveta seguinte "}".Logo qualquer texto poderá ser inserido entre estas chavetas.Em algums compiladores tais como o "Turbo Pascal" os comentários são inseridos entre "(*"e"*)".

Utilize procedimentos e funções de forma a fazer com que o texto se torne mais organizado. Se em vários sitios do programa terá de ser repetida a mesma acção (por exemplo a leitura duma linha de texto de um ficheiro),considere a hipótese de criar um procedimento ou função para a execução desta mesma tarefa (por exemplo PROCEDURE le_linh_fich). Isto tornará o programa mais claro, mais eficiente e mais pequeno.

Palavras reservadas em Turbo Pascal

As palavras seguinte não podem ser utilizadas para identificadores. A maioria destas palavras chave são explicadas nas aulas nos capítulos descritos na segunda coluna.

keyword	*subject*
and	boolean algebra
asm	
array	arrays
begin	introduction
case	if .. then
const	variables and constants
constructor	
destructor	
div	algebra
do	loops
downto	loops
else	if .. then
end	introduction
exports	
file	
for	loops
function	procedures and functions

keyword	*subject*
goto	
if	if .. then
implementation	
in	
inherited	
inline	
interface	
label	
library	
mod	algebra
nil	pointers
not	boolean algebra
object	
of	if .. then
or	boolean algebra
packed	
procedure	procedures and fucntions

keyword	*subject*
program	introduction
record	records
repeat	loops
set	
shl	
shr	
string	variables
then	if .. then
to	loops
type	variables
unit	
until	loops
uses	
var	variables
while	loops
with	records
xor	boolean algebra

As palavras seguinte estão relacionadas com variáveis e constantes. A utilização destas palavras em identificadores é desaconselhável.

boolean
byte
char
double
integer

real
string
text
word

Teste rápido:

Para testar o conhecimento que tem adquiriu nesta lição click aqui para fazer um teste on-line. Note que este não tem a mesma forma que a frequência final.

Peter Stallinga. Universidade do Algarve, 17 fevereiro 2002

Aula 5: Variáveis

Write e Writeln

As instruções `write` e `writeln` são usadas para mostrar informação no ecrã: texto, números, valor das variáveis, etc. Elas são bastante utilizadas e quase todos os programas fazem uso de uma intrução `write` ou `writeln`. Como exemplo, considere o seguinte programa:

```
PROGRAM WritelnExample;

begin
  writeln('Today is a very nice day');
end.
```

quando nós compilamos e executamos o programa o ecrãn mostrará

```
Today is a very nice day
_
```

(nota, o caracter underscore _ representa a posição do cursor).

As intruções `write` e `writeln` não são usadas só para texto, tal como no exemplo anterior, mas também mostram, números ou outro tipo de informação. Mais tarde será discutido como controlar o formato de saída (a forma como a informação será mostrada). De momento, vamos ver o irmão do `writeln`, chamado `write`: A diferença entre `writeln` e `write` é que o `writeln` posiciona o cursor numa nova linha, de forma que a próxima saída apareça a por baixo do texto inicial.Com a intrução `write`, o cursor fica logo a seguir ao texto:

```
PROGRAM WriteExample;

begin
  write('Today is a very nice day');
end.
```

quando nós compilamos e executamos o programa o ecrã mostrará

```
Today is a very nice day_
```

Quando nós queremos imprimir várias coisas ao mesmo tempo, podemos fazê-lo com a mesma intrução `write`

```
PROGRAM WriteMultiple;

begin
  write('Today is ', x, ' a very nice day');
end.
```

que deverá mostrar(quando x é igual a 34)

```
Today is 34 a very nice day_
```

Compare os seguintes programas:

program

```
PROGRAM WriteLnsExample;

begin
  writeln('Today is a ');
  writeln('very nice day');
end.
```

output

```
Today is a
very nice day
_
```

program

```
PROGRAM WritesExample;

begin
  write('Today is a ');
  write('very nice day');
end.
```

output

```
Today is a very nice day_
```

program

```
PROGRAM WritesExample;

begin
  writeln('Today is a ');
  writeln;
  write('very nice day');
end.
```

output

```
Today is a

very nice day_
```

program

```
PROGRAM WritesMulti;

begin
  writeln('Today ','is',' a ');
  write('very nice day');
end.
```

output

```
Today is a
very nice day_
```

Tipos de Variáveis

Variáveis guardam valores e informação. Elas permitem aos programas efectuar cálculos e guardar os resultados para utilizar mais tarde.Imagine uma variável como uma caixa que pode conter informação. Quando nós queremos saber essa informação, abrimos a caixa e lê-mos o valor. No final, colocamos a informação de volta na caixa e deixamo-la lá, até necessitar-mos outra vez dessa informação. Para simplificar a identificação, e para prevenir confusões, todas as caixas têm de ter um nome, um identificador (ver aula 4).

O tipo de caixa define o tamanho da caixa e o tipo de informação que poderá ser encontrada lá. As variáveis podem ser de

vários tamanhos e gostos.

As variáveis podem guardar números, nomes, textos, etc. Em versões modernas do PASCAL existem muitos tipos de variáveis. Os mais importantes são

grupo	tipo	gama	tamanho
I	boolean	TRUE / FALSE	1 bit
II	byte	0.. 255	8 bit = 1 byte
II	integer	–32768 .. 32767	16 bit = 2 bytes
II	word	0 .. 65535	16 bit = 2 bytes
II	longint	–2147483648 .. 2147483647	32 bit = 4 bytes
III	real	–2.9E–39 .. 1.7E38	48 bit = 6 bytes
III	double	–5.0E–324 .. 1.7E308	64 bit = 8 bytes
III	extended	–3.4E–4932 .. 1.1E4932	80 bit = 10 bytes
IV	char	#0 .. #255	8 bit = 1 byte
IV	string	text	256 bytes

Note a convenção de expoentes em notação científica:
2.9E–39 é 2.9 x 10^{-39}, 1.7E308 é 1.7 x 10^{308}, etc.

I Boolean é usado para guardar e manipular informação do tipo true-orfalse. Tal como vimos na aula 3, isto é um bit de informação.

II Byte, **integer**, **word**, e **longint** guardam números inteiros. Isto é usado para coisas que podem ser

contadas; número de pessoas na sala, número de portas de um carro, número de chamadas telefónicas que alguém efectuou, ano lectivo, dia do mês, etc. **Byte** e **word** apenas guardam números positivos, enquanto que **integer** e **longint** podem guardar números positivos e negativos. O tipo de variável byte é o que ocupa menos memória, apenas 8 bits, mas o alcançe dos valores está muito limitado, apenas de 0 .. 255. Se precisar-mos de guardar números inteiros positivos maiores, teremos que utilizar **word**. Se queremos guardar números positivos e negativos grandes e teremos que utilizar **longint**. Nota: Como **byte** tem 8 bits, pode armazenar 2^8=256 números diferentes: 0 .. 255 Podemos fazer o mesmo cálculo para unidades word e **integer** de 16-bit: 2^{16} = 65536, números a partir de 0 .. 65535 para **word**, e –32768 .. 32767 para **integer**. Lembre-se dos cálculos da aula 3.

III **Real**, **double** e **extended** são exemplo de váriáveis que podem guardar números de vírgula flutuante (por exemplo 3.1415926535). Estes são usados para
números fraccionários, tais como o comprimento do carro, o tempo perdido entre dois eventos, a altura de um edifício, a raiz quadrada de 3, etc. O mais pequeno é **real**, e ocupa apenas 6 bytes, tendo como custo uma pequena precisão nos cálculos. A melhor é **extended**, com 80 bit (10 bytes), os cálculos deverão ter elevada
precisão, mas os cálculos serão mais lentos e ocuparão mais espaço em memória. Uma boa escolha entre estas é a **double**.

IV Os últimos dois tipos são usados para guardar texto. **Char** é utilizado para um caractér simples, enquanto que a **string** pode guardar até 255 caractéres. Mais tarde iremos aprender como nós poderemos definir o nosso próprio tipo de variáveis, agora vamos ver como é que nós as usamos em PASCAL.

Variáveis: Declaração com VAR

Na maioria das linguagens modernas de compilação, todas as variáveis que nós utilizamos têm de ser primeiramente definidas. Isto é chamada a **declaração**. Para
ser mais preciso, declaração quer dizer reservar espaço em memória e associar um nome a ele, para que mais tarde nós possamos utilizar esse nome e não o seu
endereço de memória. quando desejar-mos aceder á informação. Em PASCAL declaramos variáveis com a intrução VAR, seguida pelas variável(eis) com o tipo que nós queremos usar. O sítio para fazer isso é antes do início do programa, mas depois da declaração do nome do programa. Por exemplo:

```
PROGRAM VarExample;

VAR i: integer;
    a: real;
    b: word;

begin
  writeln('Today is day ',i);
end.
```

Se queremos definir mais variáveis do mesmo tipo, podemos fazer isso com várias intruções ou fazê-lo apenas numa linha. Nota que em qualquer caso, nós não deveremos ter que escrever VAR novamente, se bem que não é proíbido fazê-lo:

```
PROGRAM VarsExample;

VAR i, j, k: integer;
    a1, a2: real;
VAR b1: word;
    b2: word;
    b3: word;
```

```
begin
  writeln('Today is day ',i);
end.
```

Variáveis: Inicialização

A instrução VAR não atibui um valor á variável, apenas reserva espaço para tal em memória! Neste último programa de exemplo acima, a saída poderia ser

```
Today is day 23741
```

Quando um computador é ligado, a memória é normalmente preenchida com 0's, mas passado um bocado, depois de vários programas terem feito uso da memória e terem deixado lá o seu "lixo", o conteúdo de um endereço de memória é imprevisíve.Para garantir que nós estamos a trabalhar com valores bem definidos deveremos sempre atribuir um valor a cada variável. Na próxima aula iremos aprender como é que nós podemos designar instruções no programa. Aqui é suficiente dizer que "não assuma que no início as suas variáveis são estabelecidas"

Nota: Em muitas linguagens de programação é possível atribuir um valor a uma variável no momento da declaração. Também em PASCAL isso é possível (através de variáveis constants), mas isto só irá criar confusão logo é melhor evitá-lo.

Teste rápido:

Para testar o seu conhecimento sobre aquilo que aprendeu nesta lição, click aqui para um teste on-line. Nota que esta não á a forma do teste final!

Peter Stallinga. Universidade do Algarve, 26 fevereiro 2002

Aula 6: Atribuição e constantes

`Write` and `Writeln` formatado

Utilizar write e writeln sem especificar o formato irá mostrar os valores na forma standart, que é a notação científica para reais e 'com tanto sítios quanto os necessários ' para inteiros.

```
PROGRAM WritelnUnformatted;

Var r: real;
    i: integer;

begin
  writeln('Real r = ', r);
  writeln('Integer i = ', i, '!!');
end.
```

saída:

```
Real r = 3.00000E-2
Integer i = 1!!
```

Para mudar a forma como os valores são apresentados, poderemos especificar a largura do texto, e , para as variáveis do tipo vírgula flutuante (real, double, extended), o número de casas depois do ponto flutuante, tal como mostra o programa abaixo:

```
PROGRAM WritelnFormatted;

Var r: real;
    i: integer;

begin
  writeln('Real r = ', r:8:6);
  writeln('Integer i = ', i:5, '!!');
end.
```

saida:

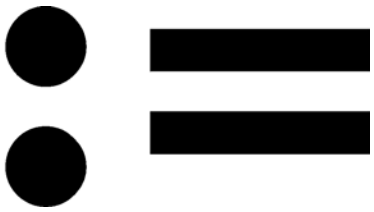
```
Real r = 0.030000
Integer i =     1!!
```

O real r é mostrado com um total de 8 espaços reservados, e 6 dígitos depois do ponto flutuante. O inteiro i é escrito com 5 espaços de texto. Como neste caso apenas 1 dígito é sufuciente, o resto, 4 espaços, é preenchido por espaços vazios.

Atribuição

Na aula passada (aula 5), tínhamos visto como é que podemos atribuir um valor inicial a uma variável. Agora iremos ver como o valor de uma variável pode ser alterada no meio do programa.

Atribuir um novo valor a uma variável estamos a fazer uma **atribuição.** Em PASCAL isto é feito com o operador



No lado esquerdo deste operador colocamos a variável, no lado direito coloca-se o novo valor ou expressão que produzirá o novo valor. Exemplos:

```
a := 3;
b := 3*a;
c := Cos(t);
```

O símbolo := é pronunciado como "será" ou "virá a ser". Isto para distingui-lo do usual símbolo matemático =. Nesta altura, é interessante fazer uma comparação entre o símbolo matemático = e o símbolo de atribuição nas linguagens de programação (:= no PASCAL). Como exemplo repare na seguinte equação matemática

$$a = a + 1$$

Isto, como nós sabemos, não tem uma solução para a. (Na comparação, outro exemplo: $a = a^2 - 2$, tem uma solução: a = 2)

Em linguagens de programação, de qualquer modo , deveremos ler a equação de uma forma diferente:

```
a := a + 1;
```

que quer dizer: (o novo valor de a) (será) (o antigo valor de a) (mais 1)
Ou, por outras palavras: primeiro o valor de a é carregado a partir da memória, a seguir 1 é adicionado a ele, e finalmente o resultado é colocado de volta na memória.Isto é fundamentalmente diferente da equação matemática. Exactamente por esta razão, em PASCAL o símbolo := foi inventado para a atribuição. Para prevenir confusão. Na maioria de outras linguagens de programação o símbolo = é usado , o que é confuso, especialmente para o programador iniciante.É por isto que o estudante é aconselhado a pronunciar o símbolo de atribuição como "será" ou "virá a ser" em vez de "é" ou "igual".

Constantes

As constantes diferem das variáveis por não poderem sofrer uma atribuição durante o decorrer de um programa. Uma vez atríbuido um valor, o mesmo não pode ser alterado. A vantagem disto é que podemos definir um valor num sítio do programa e a partir do momento que foi definido pode ser utilizado assim que seja necessário em qualquer parte do programa. Considere o exemplo de π.É muito mais fácil definir este valor apenas uma vez do que estar a escrever 3.1415926535 cada vez que precisamos dele. O

programa tornarse-á mais legível desta forma.
A definição das constante fazemos no memso sítio onde são definidas as variáveis, mas com a palavra CONST, logo por exemplo:

```
PROGRAM ConstExample;

VAR angle: real;
    tan: real;
CONST PI = 3.1415926;

begin
  angle := 45.0;
  angle := PI*angle/180.0;
  tan := Sin(angle)/Cos(angle);
end.
```

Note a forma como a constante é escrita, em letras MAÍUSCULAS. Lembre-se que PASCAL é caso não sensitivo e nós poderemos misturar maíusculas e minúsculas quando bem queremos. Ainda assim, o uso de maíusculas para constantes é uma convenção útil a que o estudante é aconselhado a seguir.Qualquer programador lendo o seu programa pode rapidamente visualizar que está a trabalhar com uma constante e que não pode alterar o seu valor.

Detalhes técnicos interessantes: Para ser mais preciso, para constantes nenhum espaço em memória é reservado tal como nas variáveis. Em vez disso, o compilador, cada vez que encontra o nome da nossa constante no nosso texto, irá substitui-la pelo seu valor.Não temos que nos preocupar com os detalhes.Resumindo temos apenas que assumir que a constante é uma variável que não pode mudar o seu valor.

Consider o seguinte programa:

```
PROGRAM InterestRateExample;

VAR money: real;
CONST TAXA = 4.3;

begin
  money := 99.0;
  money := money*(1.0+TAXA/100.0);
end.
```

No caso da interest rate mudar, nós só teremos que alterar o nosso programa num sítio (provavelmente algures no início). Isto evita erros e inconsistências no programa alterado.Se nos esquecer-mos de alterar isto num sítio o programa irá erros criar erros na saída, e agora imagine um programa com milhares de linhas de código PASCAL.

Exemplo

Em seguida é mostrado o exemplo de um programa que utiliza intruções de atribuição, variáveis e constantes. O lado direito mostra o valor das variáveis depois de cada linha

	valor de `a` depois da execução da linha	valor de `b` depois da execução da linha

`PROGRAM Assign;`
 	
`VAR a: real;`		

`    b: real;`
`CONST C = 4.3;`
 	
`begin`
`  a := 1;`
`  b := C;`
`  a := a + b + C;`
`end.`
	`undefined`
	`1.0`
	`1.0`
	`9.6`
		`undefined`
		`undefined`
		`4.3`
		`4.3`

Teste rápido:

Para testar o seu conhecimento sobre o que aprendeu nesta lição, click aqui para um teste on-line. Nota que esta **não** é a forma do teste final.

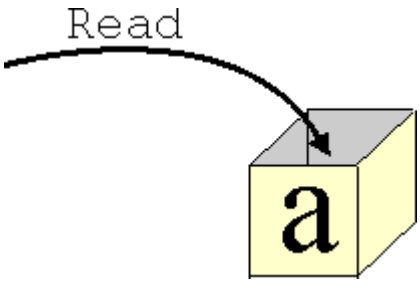
Peter Stallinga. Universidade do Algarve, 23 fevereiro 2002
translated by Eduardo Bentes

Aula 7: Matemática

Read e ReadLn

Nas aulas anteriores (veja aula 5 e aula 6) we aprendemos como mostrar os resultados dos nossos cálculos no ecrã. Com `write` e `writeln` os nossos programas podem haver *output*. Cada programa deve haver *output*, mas em muitos casos um programa também precisa de *input*. O utilizador dá o seu nome, ou dá números para processar. Ou mais simples, o utilizador quer haver controlo sobre o programa. Por exemplo, o utilizador pode parar a execução do programa com o botão 'escape'.

PASCAL: Com `Read` e `ReadLn` podemos obter informação via o teclado. Estas funções conseguem ler valores e caracteres do teclado e directamente armazenar-los nas variáveis. A forma geral das instruções é




```
Read(variable_list);

ReadLn(variable_list);
```

Aqui `variable_list` é uma lista de variáveis a ler, seperado pelas vírgulas. Por exemplo

```
Read(a, b, i);
```

Nota que as variáveis não são obrigatoriamente do mesmo tipo. No exemplo acima podem ser do tipo real, boolean e integer respectivamente (ou outros tipos). Como exemplo

```
PROGRAM ReadExample;

VAR n1, n2: integer;

begin
  WriteLn('Faz o favor de dar dois numeros seperados pelo espaco');
  Read(n1);
  Read(n2);
  WriteLn('n1 = ', n1 ,'    n2 = ', n2);
end.
```

Quando, corremos o programa o ecrã mostrará a mensagem

```
Faz o favor de dar dois numeros seperados pelo espaco
```

O programa para e o utilizador deve dar dois números

```
128 31<return>
```

(nota as convenções que nos vamos usar nestas aulas. O texto que o utilizador vai escrever aparece em *verde cursivo*, e `<return>` significa tocar o botão 'return', o botão de nova linha). Depois o programa continuará e mostrará

```
n1 = 128  n2 = 31
```

A diferença entre `Read` e `ReadLn` é que `ReadLn` põe do lado toda a outra informação até o botão 'return'. Se o utilizador deu mais informação na mesma linha, o programa vai por isto do lado e não vai utilizar esta informação. `Read` não faz isso. No exemplo acima, se substituimos as instruções `Read` com `ReadLn`, o progrma vai atribuir 128 a `n1`, depois não vai utilizar o valor 31 e consequentemente vai parar e esperar para o utilizador dar um valor para `n2`.

Matemática

Depois aprender como atribuir um valor a uma variável em aula 6, podemos ver como fazer cálculos. As operações mais basicas são adicionar, subtrair, dividir e multiplicar:

operator	operation
+	addition
–	subtraction
*	multiplication
/	division

Estes quatro *operators* (operadores), quando usado para cálculos, precisam dois *operands*. Em PASCAL fica um operador em ambas os lados. Exemplos:

bom	*mal*
`3 * a`	`* a`
`a + b`	`3 a +`

Estas instruções resultam em valores que podemos usar nas atribuições (aula 6). Exemplo:

```
c := 3 * a;
```

Outra veze nota a estrutura de atribuição: no lado esquerdo do símbolo := temos uma (só, unica, 1) variável e no outro lado pomos uma expressão que vai resultar num valor para a variável. Este valor deve ser do mesmo tipo do que a variável. Por exemplo, não faz sentido atribuir um valor inteiro a uma variável do tipo boolean: `i := TRUE;`

Matemática dos inteiros

Os operadores da secção anterior são usados para cálculos do tipo *floating point* (vírgula flutuante). Para cálculos do tipo inteiro (byte, word, integer, longint, etc), o operação divisidir não existe. Em vez, existem dois novos operadores. Imagine queremos calcular 7/3. Na escola primária aprendemos que isto é igual a **2** com um resto de **1** a dividir por 3:

$$\frac{7}{3} = 2 + \frac{1}{3}$$

Em PASCAL existem dois operadores `Div` e `Mod` que reproducem estes resultados. Exemplo:

expression	result
`7 Div 3`	2
`7 Mod 3`	1

Este operadores substituem o operdor / para números inteiros. Os outros operadores ficam iguais (*, +, -).

Prioridade

Em caso de mais de um operador numa só expressão, as regras de prioridade de matemática normal applicam-se. Os operações multiplicar e dividir têm mais alta prioridade. Por isso, quando escrevemos

```
a := 1 + 3 * 2;
```

O resultado será 7.
Se queremos mudar a ordem de execução, podemos sempre usar parênteses (and). Por exemplo

```
a := (1 + 3) * 2;
```

dá 8. Pôr parênteses nunca faz mal!

```
a := (1 + 3) - (4 + 5);
```

Exemplos

```
PROGRAM SimpleCalculations;

VAR x, y, sum, diff, divis: real;

begin
  WriteLn('Da o valor da variavel x:');
  ReadLn(x);
  Writeln('Da o valor de variavel y:');
  ReadLn(y);
  sum := x + y;
  Writeln('A soma de ', x:0:4, ' e ', y:0:4, ' e ', sum:0:4);
  diff := x - y;
  Writeln('A diferenca entre ', x:0:4, ' e ', y:0:4, ' e ', diff:0:4);
  divis := x / y;
  Writeln(x:0:4, ' dividir por ', y:0:4, ' da ', divis:0:4);
end.
```

will produce, when running:

```
Da o valor da variavel x:
3.4
Da o valor da variavel y:
1.8
A soma de 3.4000 e 1.8000 e 5.2000
A diferenca entre 3.4000 e 1.8000 e 1.6000
3.4000 dividir por 1.8000 da 1.8889
```

Nota o formato (:0:4), como explicado em aula 6.

```
PROGRAM IntegerCalculations;

VAR x, y, modd, divv: integer;

begin
  WriteLn('Da o valor da variavel x:');
  ReadLn(x);
  Writeln('Da o valor de variavel y:');
  ReadLn(y);
  sum := x + y;
  Writeln('A soma de ', x, ' e ', y, ' e ', sum);
  diff := x - y;
  Writeln('A differenca entre ', x, ' e ', y, ' e ', diff);
  divv := x Div y;
  modd := x Mod y;
  Writeln(x, ' dividir por ', y, ' da ', divis, ' mais ', modd, '/', y);
end.
```

will produce, when running:

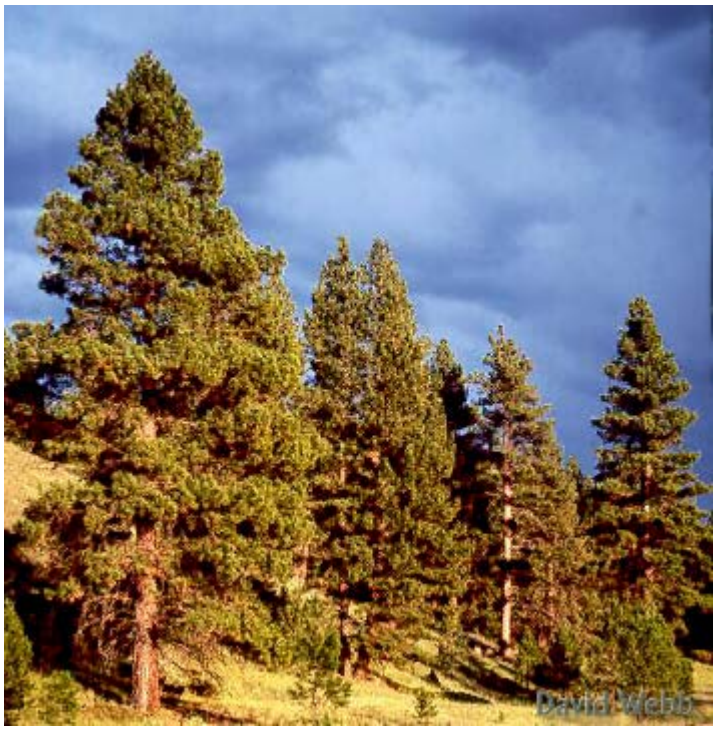
```
Da o valor da variavel x:
13
Da o valor da variavel y:
5
A soma de 13 e 5 e 18
A differenca entre 13 e 5 e 8
13 dividir por 5 da 2 mais 3/5
```

Mini teste:

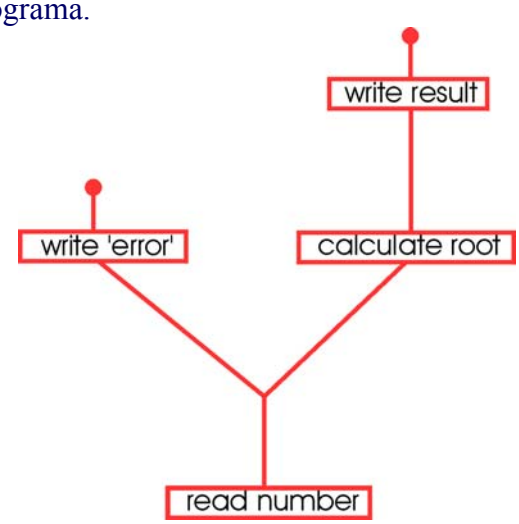
Para testar o seu conhecimento, sobre o que aprendeu nesta aula, clique aqui para um teste on-line.

Peter Stallinga. Universidade do Algarve, 16 Maio 2002

Aula 8: Bifurcações I (if ... then ... else)



Até já, todas as instruções do programa foram sempre executadas. Além disso, foram executadas exactamente na ordem como escrito. A primeira linha do código foi executada ao primeiro, depois a segunda linha, etc. Isto não é sempre assim. Com *branching* (*branch* é tronco em inglês; parte de um árvore) podemos controlar a execução do programa.



Imagine pretende execrever um programa para calcular o raiz de um número. Tomar o raiz de um número negativo não faz sentido. Por isso, queria gerar um erro se o utilizador deu um valor negativo. Queria que aparece o texto

```
Negative numbers are not allowed!
```

no ecrã. Obviamente, não quer que estetexto vai aparecer sempre, mas sim só quando o utilizador dá um número negativo. Se o utilizador dá um número positivo quer o raiz deste número no ecrã:

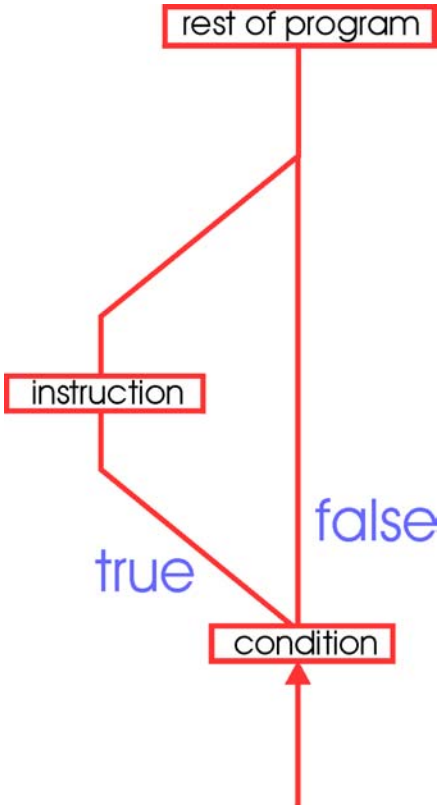
```
O raiz de 5 is 2.23607
```

Queria haver uma maneira de verificar se um número é negativo e dependente de este verificação executar partes diferentes do programa. Exactamente isto é possível com *branching*. Vamos ver as isntruções "if-then", "if-then-else" e "case-of".

if ... then ...

A maneira mais simples de haver controlo sobre a execução das instruções é com o *statement* if .. then. O síntaxe completo é

```
if condition then
instruction;
```



Para `condition` podemos pôr a nossa condição e para `instruction` substituimos a instrução (ou instruções) a ser executa se (e só se) a condição é verdade.
Nota que `condition` é uma expressão que retornerá um valor do tipo boolean (veja aula 4). Isto pode ser uma variável. Por exemplo se `b` for declarado como `boolean`, a seguinta linha de código é correcta:

```
if b then instruction;
```

No outro lado, mais vulgar são condições com expressões que comparam coisas, por exemplo

```
if (x = y) then instruction;
if (x < y) then instruction;
```

comparison	meaning
`(a = b)`	`a` equal to `b`
`(a <> b)`	`a` not equal to `b`
`(a < b)`	`a` smaller than `b`
`(a > b)`	`a` larger than `b`
`(a <= b)`	`a` smaller or equal to `b`
`(a >= b)`	`a` larger or equal to `b`

Lembra que se nós queremos executar mais do que uma instrução podemos juntar-os com a combinação begin ... end. Assim, para a estrutura if ... then, as instruções aparecem como uma só instrução:

```
if (a = b) then
  begin
    instruction1;
    instruction2;
  end;
```

Neste caso, ambas as instruções `instruction1` e `instruction2` serão executadas se a é igual a b.

A execução normal do programa continua depois o bloco de instruções. No exemplo a seguir, `instruction3` e `instruction4` serão executadas, apesar da condição (a = b).

```
if (a = b) then
  begin
    instruction1;
    instruction2;
  end;
instruction3;
instruction4;
```

a ser executado:

(a=b)	(a<>b)

`instruction1`
`instruction2`
`instruction3`
`instruction4`
	`instruction3`
	`instruction4`

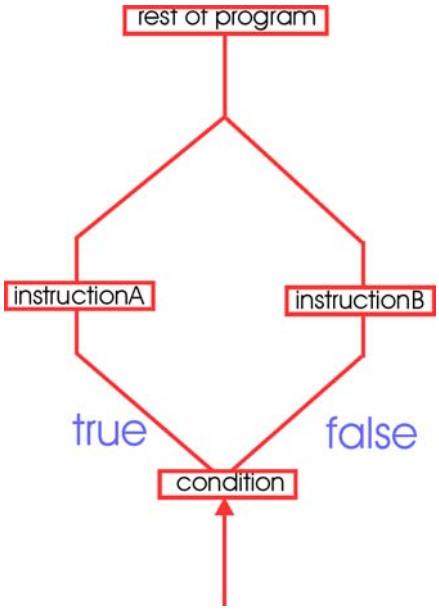
Nota que aqui acaba a semelhança com os troncos de um árvore. Num árvore, os troncos nunca se

juntaram; uma vez dividido nunca mais será possível juntar com o tronco prinicpal.

---

if ... then ... else ...

Se nós também queremos executar coisas quando a condição é falsa podemos fazer isto com a estrutura if ... then ... else. A forma geral desta estrutura é




```
if condition then
  instructionA
else
  instructionB;
```

Nota a peculiaridade de PASCAL: **a instrução a antes else não é finailzda pela ponto-e-vírgula ;**

exemplo:

```
if (a=b) then
  begin
    instruction1;
    instruction2;
  end
else
  begin
    instruction3;
    instruction4;
  end;
instruction5;
instruction6;
```

a ser executado:

(a=b)	(a<>b)
instruction1	instruction3
instruction2	instruction4
instruction5	instruction5
instruction6	instruction6

Aqui vem um exemplo completo que calcula o raiz de um núemro dado pelo utilizador:

```
PROGRAM SquareRoot;

Var x: real;
       root: real;

begin
  writeln('De um numero');
  readln(x);
  if (x<0) then
    writeln('Numeros negativos nao sao permitidos!')
  else
    begin
      root := Sqrt(x);
      writeln('O raiz de ', x:0:4, ' e ', root:0:4);
    end;
  writeln('Bom dia');
end.
```

Running the program; two examples:

```
De um numero
3.68
O raiz de 3.6800 e 1.9183
Bom dia
```

```
De um numero
-3.68
Numeros negativos nao sao permitidos!
Bom dia
```

Aula 8: ... de raizes e troncos

Mini teste:

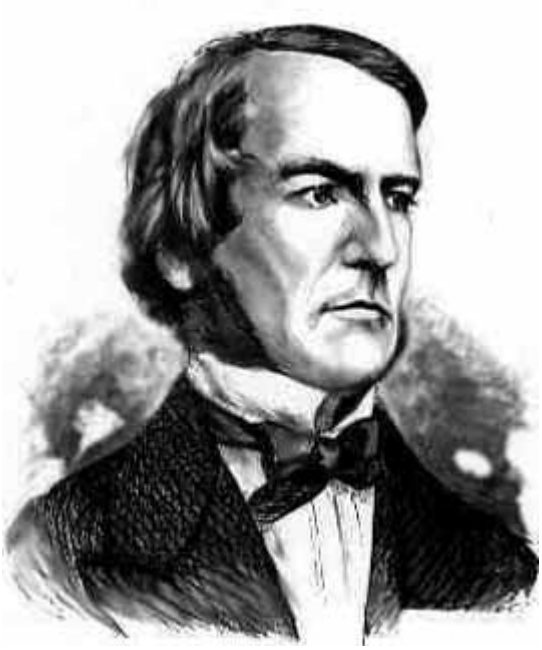
Para testar o seu conhecimento, sobre o que aprendeu nesta aula, clique aqui para um teste on-line.

Peter Stallinga. Universidade do Algarve, 16 Maio 2002

Aula 9: Bifurcações II / Boolean Algebra

Traduzido pelo Peter Stallinga

Boolean Algebra



George Boole (1815-1864)

Matemático Inglês. A sua obra "The mathematical analysis of logic" (A análise matemática do lógico) (1847) estabeleceu a base do lógico matemático moderno, e a sua algebra Boolean pode ser usado para desenhar computadores modernos.
O sistema de Boole é essencialmente bi-valored. Isto pode ser representado por

0 ou 1	*"representação binária"*
VERDADE ou FALSA	*"representação verdade"*
0 V ou 5 V	*"Electrónica TTL (transistor-transistor logic)"*
0 pC ou 1 pC (pC = pico-Coulomb)	*"a carga num condensador, a unidade elementária na RAM (dinâmica)"*

Na aula anterior (aula 8) vimos como a controlar a corrente de um programa através as instruções `if ... then` e `if ... then ... else`. Usamos condições como (x<1.0). Agora, imagine usamos isto num cálculo do raiz de ($x^2 - 4$). Obviamente, não existe soluções quando *x* fica entre –2 e +2. Seria útil verificar se *x* fica neste intervalo. Uma solução poderia ser

```
if (x<2) then
  if (x>-2) then
    WriteLn('Error');
```

Mais bonito seria fazer isto numa só condição única. Então é possível com

```
if ((x<2) AND (x>-2)) then WriteLn('Error');
```

o que, obviamente, significa que ambas as condições (*x*<2 e *x*>–2) devem ser satisfazadas para tornar a condição inteira verdade.
Isto é um exemplo do cálculo Boolean

```
condition3 := condition1 AND condition2;
if condition3 then instruction;
```

Outros operadores de **algebra Boolean** usados em PASCAL são `OR` and `XOR`. (Outras não implementados incluem NAND and NOR). A função dos operadores `AND` e `OR` é obvio. O operador `XOR` é um pouco mais complicado: `(a XOR b) significa` "a ou b verdade, mas não ambas!"
O último operador Boolean implementado em PASCAL é o invertador `NOT`. `(NOT a)` retorne o contrário do a: se `a` for `TRUE`, `(NOT a)` é `FALSE`, e se `a` for `FALSE`, `(NOT a)` é `TRUE`. Nota que, enquanto `AND`, `OR` and `XOR` levam dois operandos (por examplo `a XOR b`), `NOT` leva só um (`NOT a`).
Com estes quatro operadores podemos calcular qualquer condição necessário.

Resumindo: Aqui estão as tabelas de verdade ("truth tables") dos operadores Boolean das linguagens modernas de programação:

Boolean operador
AND
OR
XOR
NOT

AND

a	b	a AND b
TRUE	**TRUE**	**TRUE**
TRUE	FALSE	FALSE
FALSE	**TRUE**	FALSE
FALSE	FALSE	FALSE

OR

a	b	a OR b
TRUE	**TRUE**	**TRUE**
TRUE	FALSE	**TRUE**
FALSE	**TRUE**	**TRUE**
FALSE	FALSE	FALSE

XOR

a	b	a XOR b
TRUE	**TRUE**	FALSE
TRUE	FALSE	**TRUE**
FALSE	**TRUE**	**TRUE**
FALSE	FALSE	FALSE

NOT

a	NOT a
TRUE	FALSE
FALSE	**TRUE**

Exemplos:

```
a := 1;
b := -2;
c := 3;
d := 3;

if (a>0) then
  WriteLn('VERDADE')
else
  WriteLn('FALSA');
```

```
(a>0)                       VERDADE
(b>0)                       FALSA
(a>0) AND (b>0)             FALSA
(a>0) OR (b>0)              VERDADE
(a>0) XOR (b>0)             VERDADE
NOT (a>0)                   FALSA
(NOT (a>0)) OR (b>0)        FALSA
(2=b) OR (NOT (2=b))        VERDADE
(NOT (c>0)) XOR (d>0)       VERDADE
```

Boolean algebra para números inteiros

Podemos também aplicar as regras da Bolean algebra aos números inteiros (byte, word, integer, longint, veja aula 5). Embora de ser fora da idea original de Boole, podemos fazer os mesmos cálculos com os números, desde que fazemos isso "one bit at a time" e usamos a convenção "1 = TRUE" e "0 = FALSE". Com esta convenção, as tabelas de verdade para cálculos Boolean tipo bit tornam-se.

AND

a	b	a AND b
1	**1**	**1**
1	0	0
0	**1**	0
0	0	0

OR

a	b	a OR b
1	**1**	**1**
1	0	**1**
0	**1**	**1**
0	0	0

XOR

a	b	a XOR b
1	**1**	0
1	0	**1**
0	**1**	**1**
0	0	0

NOT

a	NOT a
1	0
0	**1**

49 = 1 1 0 0 0 1
24 = 0 1 1 0 0 0
49 OR 24 = 1 1 1 0 0 1 = 57

Un exemplo, imagine queremos calcular 49 OR 24
Ao primeiro temos de converter os números ao sistema binário (veja aula 3):
49 = 1*32 + 1*16 + 0*8 + 0*4 + 0*2 + 1*1 = 110001
24 = 0*32 + 1*16 + 1*8 + 0*4 + 0*2 + 0*1 = 011000
Depois fazemos os cálculos (um *bit* por vez) com a convenção mencionada acima (1 OR 1 = 1, 1 OR 0 = 1, 0 OR 1 = 1, 0 OR 0 = 0), resultado: 111001
Este resultado deve ser convertido ao sistemo decimal system:
111001 = 1*32 + 1*16 + 1*8 + 0*4 + 0*2 + 1*1 = 57

Bifurcações multiplas: Case ... Of

Na aula anterior (veja aula 8) usamos a instrução `if ... then` para escolher entre os dois ramos do nosso programa. Às vezes queremos que existe mais do que duas possibilidades a continuar o programa. Por isso existe a instrução `Case ... Of`.
Imagine o programa seguinte pede o utilizador (o aluno) o ano ao qual ele pertence:

```
PROGRAM Years;

VAR ano: integer;

begin
  WriteLn('From what year are you?');
  ReadLn(ano);
  if (ano=1) then
    writeln('primeiro ano: MAT-1, CALC-1')
  else
    if (ano=2) then
      writeln('segundo ano: INF, LIN-ALG')
    else
      if (ano=3) then
        writeln('terceiro ano: ELEC, FYS')
      else
        if (ano=4) then
          writeln('quarto ano: QUI, MAT-2')
        else
          if (ano=5) then
            writeln('quinto ano: PROJECTO')
          else
            writeln('>5: AINDA NAO ACABOU?');
end.
```

(Nota a estrutura do programa, com indentações. Também nota que nçã há ; antes cada `else`).
O programal correrá sem problemas

```
From what year are you?
 1
primeiro ano: MAT-1, CALC-1

From what year are you?
 4
quarto ano: QUI, MAT-2
```

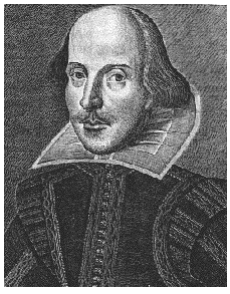
No entanto, a estrutura do programa não fica bem legível. Para melhorizar-lhe, usamos a estrutura `Case ... Of`.

```
Case expression of
  value1: instruction1;
  value2: instruction2;
        |
  valueN: instructionN;
  else instructionE;
end;
```

Enquanto que em "`if condition then instruction1`" a condição é necessariamente do tipo boolean (TRUE ou FALSE), em `Case ... Of` podemos usar qualquer `expressão` que dá um valor do tipo "contável", ou seja qualquer tipo de valores discretos (por exemplo: integer, byte, mas também char. No outro lado, os string e real não tomam valores discretos e por isso são proíbidos nas estruturas `Case ... Of`). Os valores `value1` ... `valueN` devem ser do mesmo tipo.
O mesmo programa acima, agora usando a estrutura `case ... of`:

```
PROGRAM Years;

VAR ano: integer;

begin
  WriteLn('From what year are you?');
  ReadLn(ano);
  Case ano of
    1: writeln('primeiro ano: MAT-1, CALC-1');
    2: writeln('segundo ano: INF, LIN-ALG');
    3: writeln('terceiro ano: ELEC, FYS');
    4: writeln('quarto ano: QUI, MAT-2');
    5: writeln('quinto ano: PROJECTO')
    else writeln('>5: AINDA NAO ACABOU?');
  end;
end.
```

O output do programa será igual, mas agora fica mais legível.



Aula 9: (2=B) OR (NOT (2=B))

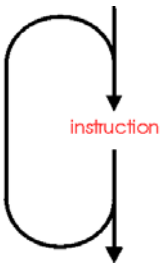
Quick test:

Para testar o seu conhecimento sobre aquilo que aprendeu nesta lição, click aqui para um teste on-line. Nota que esta **não** á a forma do teste final!

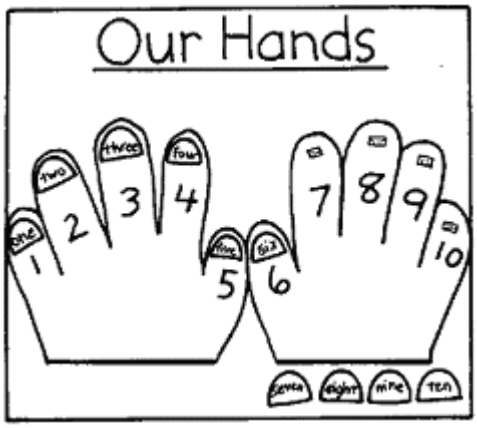
Aula 10: Ciclos I: For

translated by Peter Stallinga

Ciclo For



O ciclo mais usado em PASCAL é o ciclo For. Dentro do ciclo vão ser executadas a(s) Instrução(ões) um número de vezes predeterminado, numa maneira **contável**. A contrastar temos os ciclos que correm enquanto uma certa condição é verdade, tal como nos vamos ver nas aulas seguintas. A estrutura geral do ciclo For é:




```
For variable := start_value To end_value Do
     instruction;
```

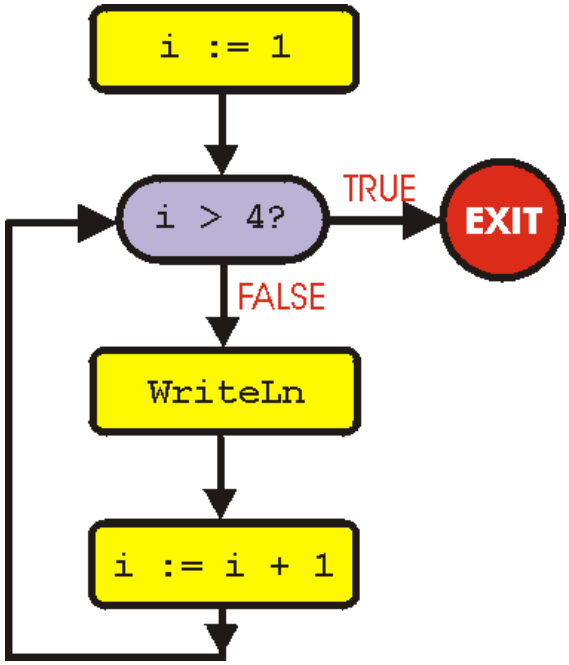
A instrução `instruction` vai ser repetada um certo número de vezes, determinado pelos parámetros de controlo `start_value` e `end_value`.
Porque o ciclo faz coisas numa maneira **contável**, os parámetros de controlo `variable`, `start_value` e `end_value` devem ser de um tipo inteiro. Também não esquecer de declarar a variável (veja aula 5)
Exemplo:

código PASCAL

```
PROGRAM ForLoopExample;

Var i: integer;

begin
  For i := 1 To 4 Do
    WriteLn('Ola');
end.
```

output

```
Ola
Ola
Ola
Ola
```



Este programa faz o seguinte
1) atribui 1 a `i`
2) verifica se `i` é maior que 4
3) se for verdade: SAI do CICLO, senão
4) executa `WriteLn('Ola');`
5) adiciona 1 a `i`
6) volta para o passo 2)

Instruções multiplas

Tal como na estrutura `if ... then ... else` podemos juntar instruções com `begin` e `end` nos ciclos:

código PASCAL

```
For i := 1 To 4 Do
  begin
    WriteLn('Ola');
    WriteLn('Eu chamo-me Peter');
  end;
```

output

```
Ola
Eu chamo-me Peter
Ola
Eu chamo-me Peter
Ola
Eu chamo-me Peter
Ola
Eu chamo-me Peter
```

Usar a variável de controlo

Entre o ciclo podemos usar a variável de controlo, mas **não faz porcarias**

Bom código:

código PASCAL

```
For i := 1 To 4 Do
  WriteLn(i, ' Ola');
```

output

```
1 Ola
2 Ola
3 Ola
4 Ola
```

Mau código: (altera variável no ciclo for)

código PASCAL

```
For i := 1 To 4 Do
  begin
    WriteLn(i, ' Ola');
    i := i + 1;
  end;
```

output

```
1 Ola
3 Ola
```

O programa no lado direito é um exemplo de mau código. Este estilo de programar, embora poupe espaço e tempo a correr (às vezes), torna o seu programa mal estruturado, ilegível e impossível de perceber para os seus colegas! Se pretende realizar o *output* de lado direito, use os ciclos do tipo `while` ou `repeat-until`, ou, melhor, usa a estrutura abaixa

program code

```
For i := 1 To 2 Do
  WriteLn(2*i-1, ' Ola');
```

output

```
1 Ola
3 Ola
```

Expressões

Os parámetros `start_value` e `end_value` podem também ser variáveis ou expressões das quais resultam valores do tipo inteiro, por exemplo:

código PASCAL

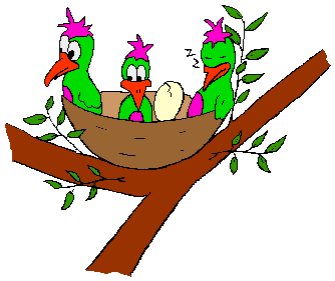
```
PROGRAM ForLoopExample;

Var i: integer;
    j: integer;

begin
  j := 5;
```

output

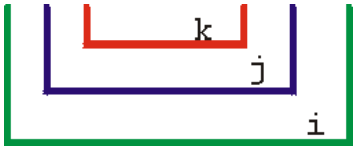
```
5 Ola
6 Ola
7 Ola
8 Ola
9 Ola
```

```
  For i := j To 2*4+1 Do
    WriteLn(i, 'Ola');
end.
```

Ciclos dentro de outros ciclos



Os ciclos For (e também os outros ciclos que vamos conhecer) podem ser 'nested' ou seja ciclos dentro de outros ciclos. É possível criar ciclos duplos, ou ciclos triplos (tal como é mostrado nas figuras abaixo) ou com outros níveis. Estas estruturas parecem ninhos dos pássaros. Por isso colocar ciclos dentro de ciclos chama-se '**nesting**' ('nest' é ninho em inglês). Aqui vêm alguns exemplos



código PASCAL

```
PROGRAM NestedLoop3;

Var i, j, k: integer;

begin
  For i := 1 To 2 Do
    For j := 1 To 2 Do
      For k := 1 To 2 Do
        WriteLn('i=',i,' j=', j,
          ' k=',k);
end.
```

output

```
i=1 j=1 k=1
i=1 j=1 k=2
i=1 j=2 k=1
i=1 j=2 k=2
i=2 j=1 k=1
i=2 j=1 k=2
i=2 j=2 k=1
i=2 j=2 k=2
```

código PASCAL

```
PROGRAM NestedLoop3;

Var i, j, k: integer;

begin
  For i := 1 To 2 Do
    For j := 1 To 2 Do
      begin
        For k := 1 To 2 Do
          WriteLn('i=',i,' j=', j,
            ' k=',k);
        For k := 1 To 2 Do
          WriteLn('i=',i,' j=', j,
            ' k=',k);
      end;
end.
```

output

```
i=1 j=1 k=1
i=1 j=1 k=2
i=1 j=1 k=1
i=1 j=1 k=2
i=1 j=2 k=1
i=1 j=2 k=2
i=1 j=2 k=1
i=1 j=2 k=2
i=2 j=1 k=1
i=2 j=1 k=2
i=2 j=1 k=1
i=2 j=1 k=2
i=2 j=2 k=1
i=2 j=2 k=2
i=2 j=2 k=1
i=2 j=2 k=2
```

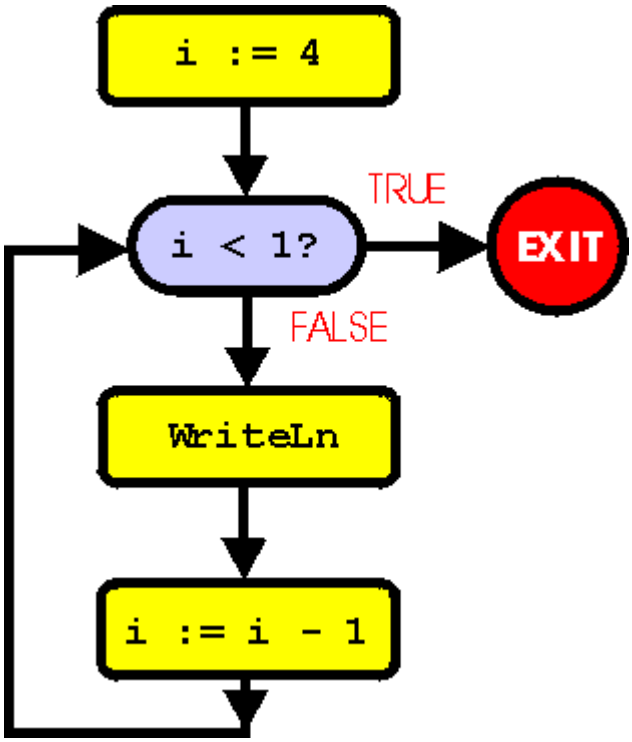
Ciclos com 'DownTo'

Em vez de um ciclo com uma variável aumentanda, podemos escrever um ciclo onde a variável sempre desce. Fazemos isto através a substituição da palavre 'To' pela palavre 'DownTo'. Por examplo:

código PASCAL	*output*

`For i := 4 DownTo 1 Do WriteLn(i, ' Ola');`	`4 Ola`
	`3 Ola`
	`2 Ola`
	`1 Ola`

Este programa faz o seguinte
1) atribui 4 a `i`
2) verifica se `i`é menor que 1
3) se for verdade: SAI do CICLO,
   senão
4) executa `WriteLn('Ola');`
5) subtrai 1 de `i`
6) volta para o passo 2)



Mini teste:

Para testar o seu conhecimento sobre o que aprendeu nesta lição, click aqui para um teste on-line. Nota que esta não é a forma do teste final.

Peter Stallinga. Universidade do Algarve, 9 março 2002

Aula 11: Ciclos II: `While ... Do` e `Repeat ... Until`

Aula

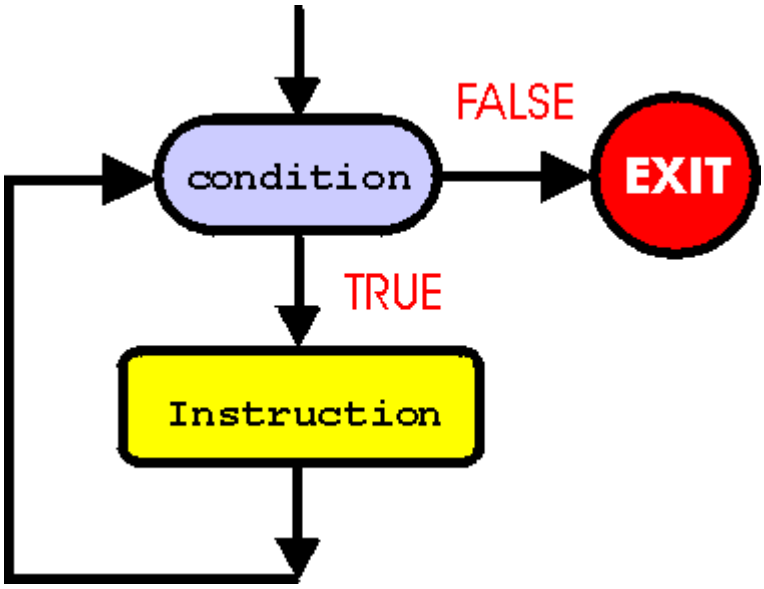
traduzido pelo Eduardo Bentes

`While ... Do`

Os ciclos abordados nesta aula, while-do e repeat-until são utilizados para repetições que não são exactamente contáveis.Normalmente usamos estes ciclos quando não é claro a altura em que o ciclo terminará, por exemplo quando o controlo das variáveis são alteradas no interior do ciclo (é estritamente desaconselhado em ciclos For).Também, quando queremos o ciclo sobre algo com variáveis de vírgula flutuante utilizamos os ciclos while-do e repeat-until.

O formato geral de um ciclo while-loop é

```
While condition do
  Instruction;
```



Esta estrutura repete a instrução enquanto a condição é verdadeira (true). A condição (`condition`) é qualquer condição que resulte num valor booleano (True ou False), tal como foi discutido na aula 8. Isto pode ser uma comparação, ou qualquer outra coisa (informação num ficheiro?se o utilizados pressionou uma tecla?, etc.). Nota que a estrutura while-do não atribui um valor inicial a qualquer variável tal como o ciclo for o fazia.Logo, temos que fazê-lo por nós mesmos.
Exemplo:

código do programa	*saída*

```
PROGRAM WhileExample;

Var x: real;

begin
  x := 0.0;
  While (x<=1.0) do
    begin
      WriteLn(x:0:1);
      x := x + 0.1;
    end;
end.
```

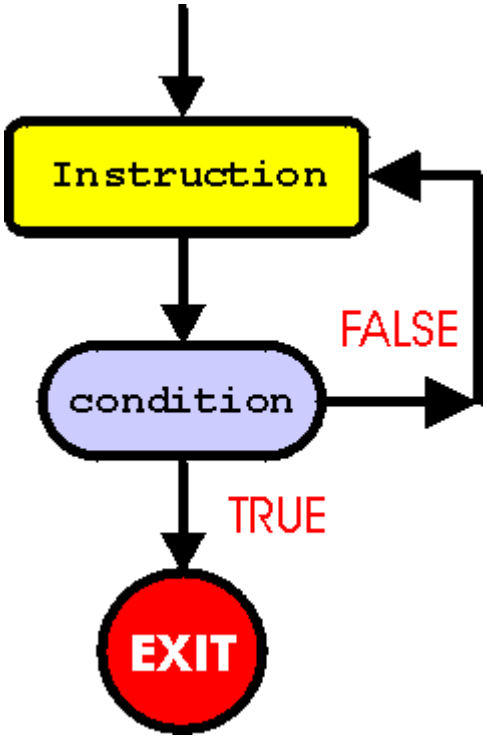
```
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
```

---

repeat ... until

A estrutura repeat-until é muito similar à estrutura while-do.A estrutura repeat-until repete uma instrução (ou instruções), até que a condição é verificada.O formato geral do ciclo repeat- until é:

```
Repeat
  Instruction;
until condition;
```



Outra diferença de grande importância entre while-do e repeat-until é que com while-do a condição é verificada no **início** do ciclo,e no cilclo repeat-until é verificada no **fim.** Posto isto, **as intruções no ciclo repeat-until são executadas pelo menos uma vez.**
Repare nos programas mostrados em seguida (também está incluído um exemplo com um ciclo for). Apenas o código com a estrutura repeat-until tem saída.

código do programa

```
x := 100;
repeat
  writeln('Ajax');
  x := x + 1;
until (x>10);
```

código do programa

```
x := 100;
while (x<=10) do
  begin
    writeln('Ajax');
```

código do programa

```
for x := 100 to 10 do
  begin
    writeln('Ajax');
  end;
```

```
    x := x + 1;
  end;
```

saída	*saída*	*saída*
`Ajax`		

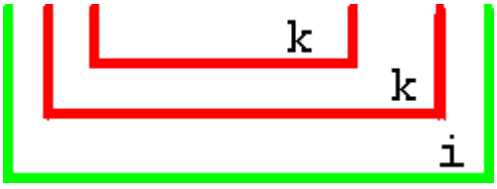
Note também as condições. A condição para repeat-until (x>0) é o oposto da condição para while-do (x<=10).

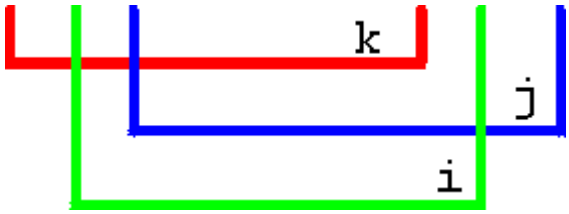
Aninhamento II

Agora que conhecemos todo os tipo de ciclos, vamos dar uma vista de olhos nas regras-de-bom-comportamento relacionadas com aninhamento.

- Cada ciclo For deve usar a sua própria variável de controlo separada
- O ciclo interior deve começar e acabar na sua totalidade dentro do ciclo exterior.

Exemplos de mau código






```
for i := 1 to 10 do
  for  k := 1 to 10 do
    begin
      for k := 1 to 10 do
         writeln('Hello');
    end;
```

```
for  k := 1 to 10 do
  begin
    i := k;
    repeat
      j := i-1;
      while (j<10) do
        begin
  end; {end of for}
  i := i + 1;
    until i>20;  {end of repeat-until}
          j := j + 1;
        end; {end of while-do}
```

Boa identação do seu programa irá prevenir erros deste tipo.

Sumário

Três regras para ciclos:

- cada ciclo com variável de controlo seperada
- cada ciclo inteiramente dentro do outro ciclo
- dê uma possibilidade de sair do ciclo. Lembra a anedota de um programador encontrada morte na banheiro com uma garrafa de champô "lava, enxaguar e repita!"

Quick test:

Para testar o seu conhecimento, sobre o que aprendeu nesta aula, clique aqui para um teste on-line.

Peter Stallinga. Universidade do Algarve, 15 março 2002

Aula 12: Programação Modular I:

Procedures

traduzido pelo Peter Stallinga

Até já usamos só coisas que PASCAL nos deu. Nos aprendemos escrever ciclos (for, while-do, repeat-until), usar *input* e *output* (readln, writeln), como a controlar o programa (if-then, if-then-else, case-of), como declarar variáveis (`var`), constantes (`const`), como atribuir valores (`:=`) e como a calcular, até calculos complicados do tipo álgebra Booleana. Mas, nunca usamos coisas NOVAS. Com procedimentos (*procedures* e *functions*) podemos fazer exactamento isso, definir novas instruções.
Já encontramos alguns procedimentos da linguagem PASCAL, nomeademente `Write`, `WriteLn`, `Read` e `ReadLn`. Agora vamos definir os nossos procedimentos próprios.

Procedures (procedimentos) e *Functions* (funções) são pequenos programas ou **módulos** dentro do programa principal. Cada módulo faz uma tarefa específica. Isto ajuda organizar o programa. Faz-o mais lógico e aumenta a eficiência através evitar repetições de partes do programa.

Existem dois tipos de módulos:

1. Módulos que retornam nada. Em PASCAL chamam-se **Procedures**. Têm dois sub-tipos
   - *Procedures* que aceitam parámetros e
   - *Procedures* que não usam parámetros (*input*)
2. Módulos que retornam valores. Em PASCAL chamam-se **Functions**. Outra vez, existem dois sub-tipos
   - *Functions* que aceitam parámetros e
   - *Functions* que não usam parámetros (*input*)

Procedures

Módulos que retornam nada chamam-se **Procedures**. Eles fazem coisas sem retornar nada. Eles podem aceitar parámetros (e trabalhar com eles) ou fazer o trabalho sem dados de entrada. *In any case*, Procedures são como programas dentro do programa. Eles têm

- O nome. As mesmas regras para identificadores aplicam. Veja aula 5
- Variáveis e constantes. temos de declarar-as no procedimento.
- uma combinação `begin` - `end;` Nota que este end é finalizado com `;`
- O código

O protótipo de um *procedure* é

```
Procedure ProcedureName;

Var <variable_list>;
Const <const_list>;

begin
```

```
  instructions;
end;
```

O lugar para declarar um *procedure* é **dentro** do programa (então **depois** a declaração do nome do programa), mas **antes** o começo do programa principal, então antes o `begin`.

Chamar

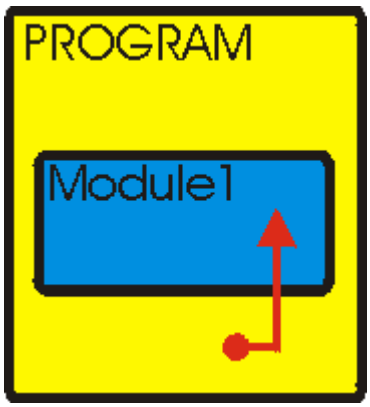
Depois a declaração do procedimento podemos usar-o no programa principal. Isto chama-se **calling** (chamar) o procedimento. Fazemos isto através escrever o nome do procedimento. Um exemplo inteiro

código PASCAL

```
PROGRAM WithProcedure;

Var x: real;

PROCEDURE Module1;
var y: real;
begin
  WriteLn('Now I am entering Procedure Module1');
  WriteLn('Give a value for y:');
  ReadLn(y);
  Writeln(y/3:0:3);
end;

begin
  WriteLn('Starting the program');
  Module1;
end.
```

output

```
Starting the program
Now I am entering Procedure Module1
Give a value for y:
 12
4.000
```



Um programa chama um dos seus procediementos

Procedures chamam *procedures*

Procedimentos podem ser chamados pelos outros procedimentos ou funções. Normalmente, em PASCAL estes procedimentos devem ser declarados depois o procedimento a chamar. Análise bem o programa a seguir com o seu output:

código PASCAL

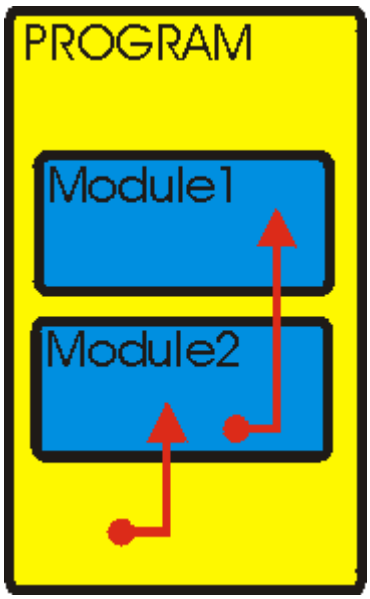
```
PROGRAM TwoProcedures;

Var x: real;
```

output

```
Now I am starting the program
Now I am entering Procedure Module2
Now I am entering Procedure Module1
```

```
PROCEDURE Module1;
var y: real;
begin
  WriteLn('Now I am entering Procedure Module1');
  WriteLn('Give a value for y:');
  ReadLn(y);
  Writeln(y/3:0:3);
  WriteLn('Leaving Module1');
end;

PROCEDURE Module2;
var z: real;
begin
  WriteLn('Now I am entering Procedure Module2');
  Module1;
  WriteLn('Give a value for z:');
  ReadLn(z);
  Writeln(z*2:0:3);
  WriteLn('Leaving Module2');
end;

begin
  WriteLn('Now I am starting the program');
  Module2;
end.
```

```
Give a value for y:
 12
4.000
Leaving Module1
Give a value for z:
 10
20.000
Leaving Module2
```



Um programa chama um dos seus procedimentos que chama um outro procedimento.

Usar as variáveis

Sim: Procedimentos podem usar as variáveis do programa principal, mas

Não: O programa principal não pode (não é possível) usar as variáveis do procedimento. Imagine o **procedimento como uma caixa com janelas do tipo espelho-uma sentido**. Dentro consegues ver tudo fora, mas fora não consegues ver nada do que está a acontecer interior.

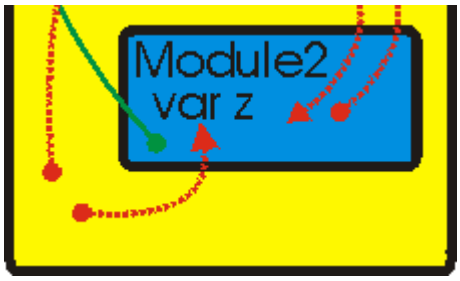
Não: Procedimenots não podem usar variáveis dos outros procedimentos. O mesmo rociocínio aplica. Um procedimento consegue ver os outros procedimentos, mas não consegue ver dentro dos procedimentos. **Procedimentos são caixas negras**.

Aqui vem um exemplo. Todas as instruções escritas com fonte ***vermelho grosso italico*** são proíbidas.

código PASCAL

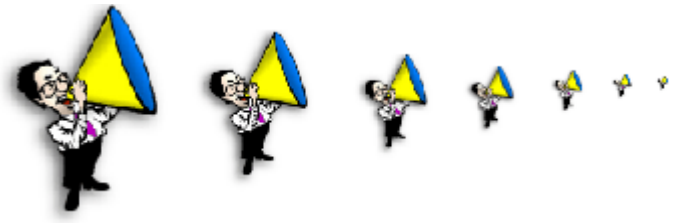
```
PROGRAM ProcedureVariables;

Var x: real;

PROCEDURE Module1;
var y: real;
begin
```

```
  WriteLn(x);
  WriteLn(y);
  WriteLn(z); (* this is not allowed *)
end;

PROCEDURE Module2;
var z: real;
begin
  WriteLn(x);
  WriteLn(y); (* this is not allowed *)
  WriteLn(z);
end;

begin
  WriteLn(x);
  WriteLn(y); (* this is not allowed *)
  WriteLn(z);   (* this is not allowed *)
end.
```



Proíbido (setas vermelhas dashed) e permitido (setas verdes solidas) use das varáveis.

Mais tarde vamos estudar o assunto de âmbito (*scope*) das variáveis em mais pormenor.



Nesting

Em muitas linguagens de programação podemos fazer *nesting* (aninhamento) dos procedimentos.
Podemos escrever (por) procedimentos dentro dos procedimentos dentro ....
Examina o progrma abaixo. O procedimento dentro o procedimento é escrito em fonte **grosso**. tal como as variáveis do procedimento Module1, o sub'procedimento não é visível fora do Module1.

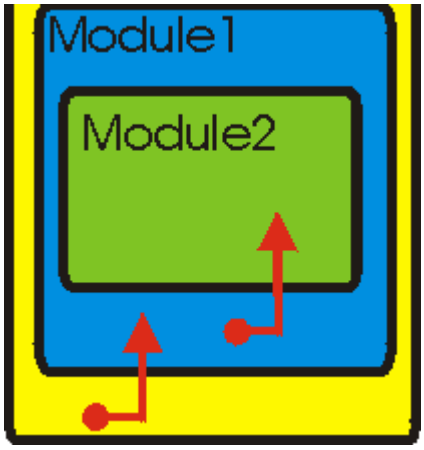
código PASCAL

```
PROGRAM NestedProcedures;

Var x: real;

PROCEDURE Module1;
var y: real;
  PROCEDURE Module2;
  var z: real;
  begin
    WriteLn('Now I am entering Procedure
Module2');
    Module1;
    WriteLn('Give a value for z:');
    ReadLn(z);
    Writeln(z*2:0:3);
    WriteLn('Leaving Module2');
  end;
```

output

```
Now I am starting the program
Now I am entering Procedure
Module1
Give a value for y:
 12
4.000
Now I am entering Procedure Module2
Give a value for z:
 10
20.000
Leaving Module2
Leaving Module1
```

```
begin
  WriteLn('Now I am entering Procedure Module1');
  WriteLn('Give a value for y:');
  ReadLn(y);
  Writeln(y/3:0:3);
  Module2;
  WriteLn('Leaving Module1');
end;

begin
  WriteLn('Now I am starting the program');
  Module1;
    (* cannot use variables or procedures *)
    (*  of Module1 here:                  *)
    (* cannot call Module2 here           *)
    (* cannot use variable y or z here    *)
end.
```



Mini teste

Quick test:Para testar o seu conhecimento, sobre o que aprendeu nesta aula, clique aqui para um teste on-line.

Peter Stallinga. Universidade do Algarve, 15 março 2002

Aula 13: Programação Modular II:

Procedimentos com *input* e *output*

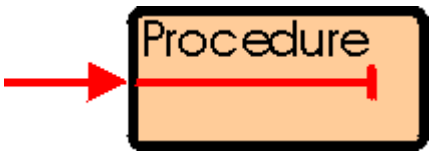
traduizido pelo Peter Stallinga

Na aula anterior (aula 12) vimos Procedures que não aceitam parâmetros ou retornam valores. Aqueles foram procedimentos simples. Agora vamos ver procedimentos que aceitam parâmetros e procedimentos que geram valores (funções).

Parâmetros de Procedures

Nos podemos passar parâmetros aos procedures. O procedure pode usar e trabalhar com este parâmetro. Em PASCAL pomos os parâmetros do procedure depois do nome do procedure, entre parêntesis:

```
 Procedure
ProcedureName(lista_de_parametros);

 Var <variable_list>;
 Const <const_list>;

 begin
   instructions;
 end;
```



Os parâmetros na lista dos parâmetros são declarados da mesma forma como as variáveis normais do programa ou procedimento, nomeademente temos de especificar o tipo do cada parâmetro. Dentro do procedimento podemos usar o parâmetro tal como foi uma variável normal. Podemos calcular com eles, usar nas condições e mesmo mudar os valores.

No exemplo abaixo, o programa calcula e mostra o quadrado de uma variável `x`: Nota a maneira de declarar e usar o parâmetro `r`.

código PASCAL

```
PROGRAM WithParameters;

Var x: real;

PROCEDURE WriteSquare(r: real);
var y: real;
begin
  r := r*r;
  y := r;
  Writeln('The suare of ',r:0:1,' is ',y:0:1);
end;
```

output

```
The square of 4.0 is 16.0
The square of 3.0 is 9.0
```

```
begin
  x := 4;
  WriteSquare(x);
  WriteSquare(3.0);
end.
```

O exemplo também mostrou como chamar o procedimento com parâmetros; com uma variável tal como em `WriteSquare(x)` ou com uma constante tal como em `WriteSquare(3.0)`.

Um outro exemplo, onde o procedimento tem dois parâmetros:

código PASCAL

```
PROGRAM WithParameters;

Var x: integer;
    y: integer;

PROCEDURE WriteSum(i1, i2: integer);
  ( will write the sum of i1 and i2 *)
var j: integer;
begin
  j := i1+i2;
  Writeln('The sum of ',i1,' and ',
    i2, ' is ', j);
end;

begin
  x := 4;
  y := 5;
  WriteSum(x, y);
  WriteSum(3, 4);
end.
```

output

```
The sum of 4 and 5 is 9
The sum of 3 and 4 is
```

Finamente, um exemplo com uma lista de parâmetros dos tipos sortidos. Tal como na declaração normal, parâmetros na lista dos parâmetros são separados com ;

código PASCAL

```
PROGRAM WithParameters;

PROCEDURE WriteNTimes(r: real; n: integer);
  (* Will write n times the real r *)
var i: integer;
begin
  for i := 1 to n do
    WriteLn(r:0:3);
end;


PROCEDURE WriteFormatted(r: real; n: integer);
  (* Will write the real r with n decimal cases *)
begin
  WriteLn(r:0:n);
end;
```

output

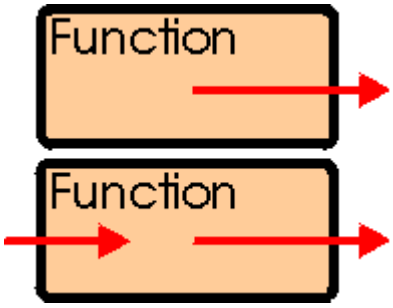
```
3.000
3.000
3.000
3.000
5.00000
5.0
```

```
begin
  WriteNTimes(3.0, 4);
  WriteFormatted(5.0, 5);
  WriteFormatted(5.0, 1);
end.
```

Funções

Funções são procedimentos que retornam um valor de *output*. O tipo do valor a retornar deve ser especificado na altura da declaração da função, depois a lista dos parâmetros (se tem), e precedido pelo **:** Exemplo:

```
 Function
FunctionName(parameter_list):
   type;

 Var <variable_list>;
 Const <const_list>;

 begin
   instructions;
 end;
```



Funções sem e com parâmetros de *input*.

Em algum lugar a função deve especificar o valor a retornar. A forma de especificar isto é

```
FunctionName := Value;
```

Com FunctionName igual a nome da função, exactamente como foi declarado. Por exemplo

```
FUNCTION Square(r: real): real;
  (* retornara o quadrado do parametro r *)
begin
  r := r*r;
  Square := r;
end;
```

No lugar onde o programa chama a função, devemos atribuir o valor que a função retorna a uma variavel (do mesmo tipo), por exemplo

```
y := Square(3.0);
```

ou usar numa expressão, por exemplo

```
y := 4.0 * Square(3.0) + 1.0;
```

ou usar numa outra função ou procedimento, por exemplo

```
Writeln(Square(3.0):0:1);
```

A full example:

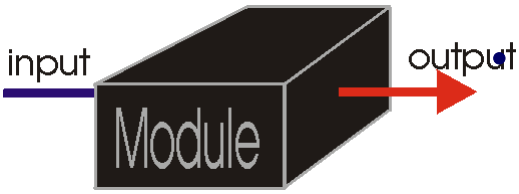
código PASCAL | *output*

```
PROGRAM WithParameters;

Var x, y: real;

FUNCTION Square(r: real): real;
  (* will return the square of of the parameter r *)
begin
  r := r*r;
  Square := r;
end;

begin
  x := 4.0;
  y := Square(x);
  WriteLn('The square of ', x:0:1, ' is ', y:0:1);
  WriteLn('The square of ', 3.0:0:1, ' is ',
    Square(3.0):0:1);
end.
```

```
The square of 4.0 is 16.0
The square of 3.0 is 9.0
```

Porquê?

Agora a grande questão é "porquê?". Porquê escrever procedimentos se é possível fazer as coisas com linhas de instruções normais? Certo, as primeiras linguagens de programção não tiveram a possibilidade de escrever procedimentos (por exemplo BASIC) e ainda foi possível atacar qualquer problema. Portanto, existem duas razões de usar procedimentos:



- Com módulos, porque são **caixas negras**, é possível distribuir as tarefas de programação. Podemos pedir uma pessoa de escrever parte de um programa, por exemplo diagonalizar um matriz, e não precisa de preocupar com os pormenores da implementação. Só precisa de concordar a lista dos parâmetros a passar.

  Da mesma forma, podemos usar partes dos outros programas (ou biblitecas). Idealmente fazemos só um "link" aos procedimentos que o nosso programa precisa, sem saber come eles funcionam. (Obviamento com saber o que é que eles fazem e como chamar eles).
- Com módulos o programa fica mais pequeno, mais eficiente e mais legível por evasão das repeticões de código e organização mais logica.

Mini teste:

Quick test:Para testar o seu conhecimento, sobre o que aprendeu nesta aula, clique aqui para um teste on-line.

Peter Stallinga. Universidade do Algarve, 2 Abril 2002

Aula 14: Funções Matemáticas

Quase cada linguagem de programação tem muitas funções de matemática implemetadas. Como exemplo, vamos aprender as funções mais úteis da PASCAL. Não é muito importante saber o formato exacto de estas funções. melhor é lembrar que as funções existe e quando precisa de uma função lê o muanual da linguagem que está a usar.

Função	*descrição*	*argumento*	*resultado*	*exemplos*
Abs	Valor absoluto do argumento. Argumento pode ser real ou inteiro Abs retornerá o mesmo tipo.	real integer	real integer	Abs(-23.2) = 23.1 Abs(12.3) = 12.3 Abs(-10) = 10
Cos	Cosine do argumento. Argumento em radians (2π rad = 360^o)	real	real	Cos(1.0) = 0.5403
Sin	Sine do argumento. Argumento em radians (2π rad = 360^o)	real	real	Sin(1.0) = 0.8415
ArcTan	Inverso tangent do argumento	real	real	ArcTan(1.0) = $\pi/4$
Exp	Exponent do argumento	real	real	Exp(1.0) = 2.718
Ln	Logaritmo natural do argumento	real (>0)	real	Ln(10.0) = 2.303
Odd	Determine se o argumento é impar	integer	boolean	Odd(3) = TRUE
Round	Arredondamento do argumento para o inteiro mais perto	real	integer	Round(3.4) = 3 Round(3.5) = 4
Int	Arredondamento do argumento para o inteiro abaixo	real	real	Int(3.99) = 3.00
Frac	Retorno a parte do número depois o ponto décimal	real	real	Frac(3.99) = 0.99
Trunc	Arredondamento do argumento para o inteiro abaixo	real	integer	Trunc(3.99) = 3
Sqrt	Raiz do argumento	real (>0)	real	Sqrt(3.0) = 1.732
Sqr	Quadrado do argumento	real	real	Sqr(2.0) = 4.0
Random	Gera número aleatório	integer	real integer	Random = 0.0234 Random(10) = 3
Randomize	Randomizar o gerador dos números aleatórios			

Funções trigonométricas

As funções trigonométricas implementadas em PASCAL são `Sin` e `Cos` que retornam o sine e cosine do argumento, respectivamente. Nota que a função Tan não existe, mas (eu espero) nos sabemos que Tan(x) é igual a Sin(x)/Cos(x). Nota também que os argumentos (os parámetros a passar à função) têm unidades de radians (2π rad = 360^o). Exemplo:

código PASCAL

```
PROGRAM Trigonometry;

Var xdeg, xrad, ysin, ycos, ytan: real;

begin
  xdeg := 45.0;
  xrad := Pi*xdeg/180.0;
   (* convert to radians! Note that the
    value Pi is defined in PASCAL *)
  ysin := Sin(xrad);
  ycos := Cos(xrad);
  ytan := ysin/ycos;
  Writeln('x = ',xdeg:0:3);
  WriteLn('Sin = ', ysin:0:4);
  WriteLn('Cos = ', ycos:0:4);
  WriteLn('Tan = ', ytan:0:4);
end.
```

output

```
x = 45.000
Sin = 0.7071
Cos = 0.7071
Tan = 1.0000
```

Esixte só uma única função trigonométrica inversa, nomeademente Tan^{-1} qual chama-se `ArcTan` em PASCAL (e `ATan` nas algumas outras linguagens). As outras funções inversas (ArcCos and ArcSin) podem ser derivadas de ArcTan. Como? (Na aula seguinta vou dar a solução).

código PASCAL

```
PROGRAM InverseTrigonometry;

Var x, y, angle: real;

begin
  x := 0.5;
  y := ArcTan(x);
  angle := 180.0*y/Pi;
  Writeln('x = ', x:0:3);
  WriteLn('ArcTan = ', angle:0:4);
end.
```

output

```
x = 0.500
ArcTan = 26.5651
```

Funções exponenciais e x^n

A função `Exp` retorna o expoente do argumento: $\mathrm{Exp}(x) = e^x$ (com e o número de Nepper)
`Ln` retorna logaritmo (natural) do argumento. O argumento deve ser maior que zero, obviamento.
A função ${}^{10}\mathrm{Log}(x)$, ou geralmente as funções ${}^{n}\mathrm{Log}(x)$ e x^n não existem em PASCAL, mas podem ser facilmente derivadas das funções `Ln` e `Exp` acimas:

$$x^n = \mathrm{Exp}(\mathrm{Ln}(x^n)) = \mathrm{Exp}(n*\mathrm{Ln}(x))$$

$${}^{n}\mathrm{Log}(x) = {}^{n}\mathrm{Log}(e^{\mathrm{Ln}(x)}) = \mathrm{Ln}(x)* {}^{n}\mathrm{Log}(e) = \mathrm{Ln}(x) / {}^{e}\mathrm{Log}(n) = \mathrm{Ln}(x) / \mathrm{Ln}(n)$$

Tem linguagens com funções do tipo x^n, por exemplo em C: pow(x,n), mas de qualquer modo, esta função é implementado da forma acima no nível abaixo (Linguagem da máquina).
Nos computadores modernos com coprocessadores matemáticos (todos os processadores Pentium têm um incorporado) os cálculos complicados Ln, Exp, mas também Sin e Cos e ArcTan são executados muito rápido, porque usam tabelas com valores precalculados dentro do processador.
Há linguagens, onde o logaritmo natural chama-se Log. Isto pode dar confusão, e por isso: sempre LÊ AS INSTRUÇÔES!
Exemplo:

código PASCAL

```
PROGRAM ExponentAndLogarithm;

Var x, y: real;

begin
  x := 100.0;
  y := Ln(x)/Ln(10.0);
  WriteLn('Log(',x:0:1,') = ',y:0:1);
  x := 3.0;
  y := Exp(x*Ln(10));
  WriteLn('10^',x:0:1,' = ',y:0:1);
end.
```

output

```
Log(100.0) = 2.0
10^3.0 = 1000.0
```

Embora da possibilidade de exprimir x^n com funções Exp e Ln, duas formas são usadas tão muito que merecem as suas implementações próprias, nomeademente: `Sqr(x)` dá x^2 e `Sqrt(x)` dá $x^{1/2}$. Exemplo:

código PASCAL

```
PROGRAM SquareRootAndSquare;

Var x, y: real;

begin
  x := 3.0;
  y := Sqrt(x);
  WriteLn('The square-root of ', x:0:3,
    ' is ',y:0:1);
  y := Sqr(x);
  WriteLn('The square of ',x:0:3,
    ' is ',y:0:1);
end.
```

output

```
The square-root of 3.0 is 1.732
The square of 3.0 is 9.000
```

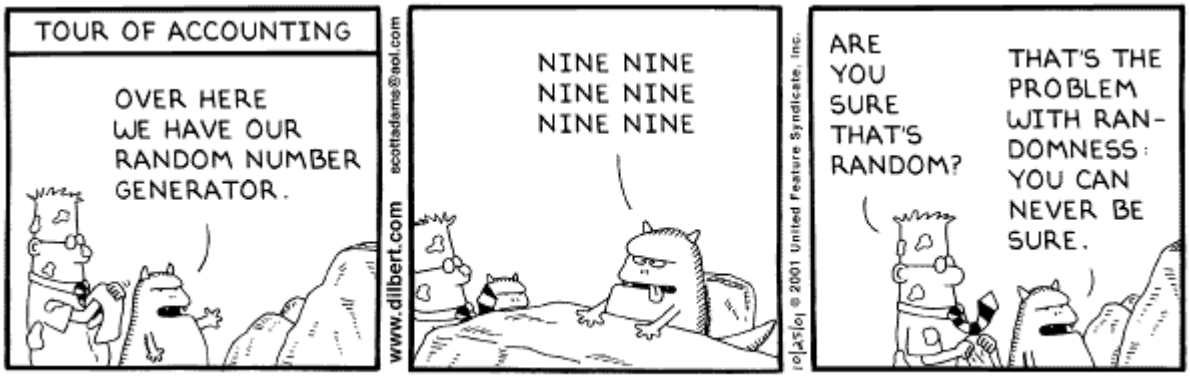
Arredondamento

Existem quatro funções de arredonademnto em PASCAL, Round, Int, Frac, e Trunc.
`Round` retorna o número inteiro mias perto. O valor retornado é do tipo integer. Exemplos: `Round(1.2)` = 1, `Round(2.5)` = 3, `Round(4.99)` = 5.
`Int` retorna a parte do número antes o ponto flutuante, então, o número inteiro imediatemente abaixo. O tipo do valor retornado é real. Isto pode dar alguma confusão; `Int` não retornerá um valor do tipo integer (como, por exemplo em C)!. Exemplos `Int(1.2)` = 1.0, `Int(2.5)` = 2.0, `Int(4.99)` = 4, `Int(5.0)` = 5.0.
`Frac` retorna a parte do número depois o ponto flutuante, então um número entre 0 e 1. Por isso, é sempre do tipo real. Exemplos: `Frac(1.2)` = 0.2, `Frac(2.5)` = 0.5, `Frac(4.99)` = 0.99, `Frac(5.0)` = 0.0.
`Trunc` é igual a Int, mas converte a integer. Exemplos: `Trunc(1.2)` = 1, `Trunc(2.5)` = 2, `Trunc(4.99)` = 4, `Trunc(5.0)` = 5.

A função Odd

Uma função estranha é `Odd`. `Odd` retorna TRUE se o argumento é impar, FALSE se contrário. Então, esta função `Odd(x)` é igual a `(x MOD 2)>0` que tammbém retorneria TRUE ou FALSE, dependente da imparidade do argumento x.

Números aleatórios



Uma das coisas mais interessante da programação é usar números aleatórios. Até já o resultado do programa foi sempre previsível (embora de ser difícil). Com números aleatórios mesmo o programador próprio não sabe o resultado do seu programa. É parecido com atirar uma moeda no ar. Mesmo se sabemos todas as leis da física, não sebemos o resultado de um experimente de atirar uma moeda no ar. Gerar números aleatórios é nada fácil. Computadores são exelente em fazer coisas numa maneira previsível, mas fazer coisas numa maneira imprevisível é muito difícil (com os humanos é o contrário). Os geradores de números aleatórios dos computadores na verdade não são muito bons, mas servem para nossos aplicações. Em PASCAL existem três funções para gerar números aleatórios, `Random`, `Random(x)` e `Randomize`.

Para gerar números aleatórios entre 0 and 1 podemos usar a função Random. Por exemplo, chamar a função cinco vezes poderia gerar a série 0.3354, 0.2134, 0.2200, 0.9876, 0.0230.

Se queremos números inteiros entre 0 e n–1 podemos usar as funções da secção acima: `Trunc(n*Random)`. Em veze de usar Trunc podemos usar a fução Random com argumento, `Random(n)`, o que retornerá exactamente a mesma coisa, um valor inteiro entre 0 e n–1.

Nota que cada vez o programa corre, o resultado será igual. A função gerará cada vez a mesma série. Para evitar isso podemos chamar `Randomize`.

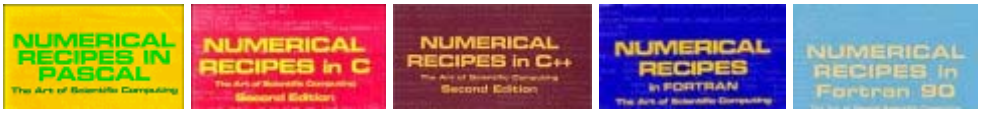
código PASCAL

```
PROGRAM RandomNumbers;

Var x, y: real;
    i: integer;

begin
  Randomize;
  for i := 1 to 5 do
    Writeln(Random:0:4);
  for i := 1 to 5 do
    WriteLn(Random(10));
end.
```

output

```
0.7132
0.5111
0.0638
0.7837
0.3810
8
5
0
8
1
```

Com estas funções podemos escrever programa de jogos com cartas, simular trânsito nas estradas, simular decay nuclear ou qualquer outro processo aleatório. A função Random é homogénio, o que significa que cada número entre 0 e 1 tem a mesma probabilidade de ser gerado. Se queremos outras distribuições podemos usar a função Random como função da fonte. Nas outras cadeiras de programação vão apredender isto. Quem já quer saber mais, eu recomendo o livro "Numerical Recipes in Pascal" (ou C ou Fortran).



Quick Test

Para testar o seu conhecimento, sobre o que aprendeu nesta aula, clique aqui para um teste on-line.

Peter Stallinga. Universidade do Algarve, 17 março 2002

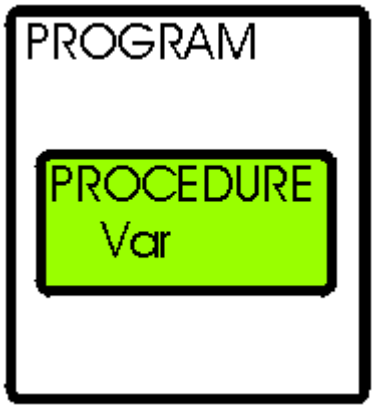
Aula 15: de variáveis e procedimentos

Esta aula:

- Âmbito das variáveis: global e local
- Passagem por valor, passagem por referência.

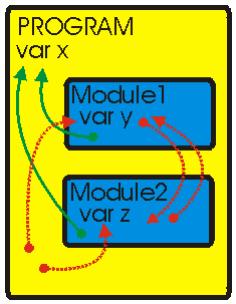
Âmbito das variáveis: Global e local

variáveis Globais são variáveis que têm definição no programa inteiro.

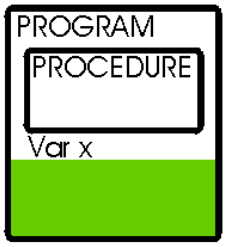


variáveis Locais só têm definição dentro de um procedimento ou função.

Com esta nova informação é muito mais fácil saber quais as variáveis nos podemos usar onde. Dentro dos procedimentos podemos usar as variáveis globais e locais. Fora dos procedimentos podemos só usar variáveis globais. Lembra aula12 (veja a imagem aqui ao lado). O programa principal não pode (consegue) usar as variáveis dos módulos, mas os módulos podem usar as variáveis globais.



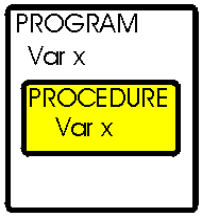
Aviso: Evita usar variáveis globais nos procedimentos! Porquê? É simples. Com o uso das variáveis globais, cópiar procedimentos para uso num outro programa será mais difícil. Provavelmente o outro programa não tem as mesmas variáveis globais. Por isso, só usar variáveis locais é melhor. Se quer usar as variáveis globais num procedimento, passa-lhes como parâmetros ao procedimento. Idealmente, o procedimento é uma unidade independente do resto do código e pode ser compilado seperado do programa.



Mais uma observação, tipicamenta para PASCAL e linguagens parecidas (*single-pass compilers*): variáveis só podem ser usadas nos lugares depois as suas declarações. Então, se pudemos a declaração de uma variável depois um procedimento, este procedimento não consegue ver a variável.
Vamos analisar alguns exemplos. Ao primeiro o exemplo da aula 13:

```
PROGRAM WithParameters;
 (* global variables x and z *)
Var x, y: real;

FUNCTION Square(r: real): real;
 (* local variable localr *)
var localr: real;
begin
  localr := r*r;
  x := localr;
  Square := localr;
end;

begin
  x := 4.0;
  y := Square(x);
end.
```

Prioridade



x é uma variável local e global.
Dentro do procedimento, a versão local será usada.

Se existem variáveis locais e globais com o mesmo nome, a variável local tem mais alta prioridade e, portanto, vai ser usada no procedimento. De qualquer modo, isto dá confusão e por isso **evita nomes iguais para as variáveis e constantes**!

Há linguagens (por exemplo BASIC) que não têm uma distinção das variáveis locais e globais. isto significa que não podemos usar a mesma variável duas vezes.

Passagem por valor ou passagem por referência

Há duas maneiras para passar parâmetros aos procedimentos, pode ser "*passing by value*", ou "*passing by reference*".
Passagem por valor (*passing by value*):
Até já usámos este tipo de passar os parâmetros aos procedimentos (procedures e functions). Nesta maneira, só o valor será passado. Não importa o que acontece dentro do procedimento com este valor, o valor da variável original usada para chamar o procedimento ficará igual. Como exemplo, para esclerecer isto um pouco mais, assume que nos temos um procedure que escreve o quadrado do parâmetro p no ecrãn. Para calcular o quadrado atribuimos um novo valor (p*p) a p. Então, o valor de p mudará dentro do procedimento:

```
PROCEDURE WriteSquare(p: real);
begin
  p := p*p;
  WriteLn(p:0:1);
end;
```

Agore, se nos chamamos este procedimento com a variável x, o valor de este variável não mudará. No programa principal:

```
begin
  x := 2.0;
  WriteSquare(x);
  WriteLn(x:0:1);
end.
```

Depois de voltar do procedimento, o valor de x não mudou. Então, o output completo do programa acima será

```
4.0
2.0
```

Passagem por referência (*Passsing by reference*):

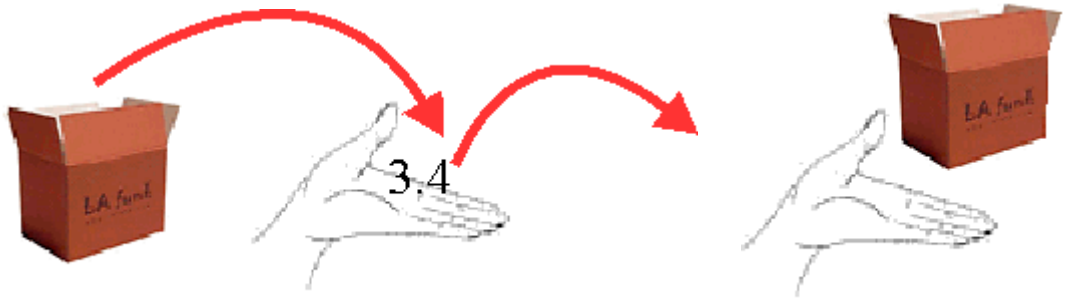
Por outro lado, se nos queremos mudar o valor da variável usada para chamar o procedimento, podemos especificar isto na altura de definição do procedimento. Basta pôr a palavra **Var** em frente do cada parâmetro que deve mudar a variável permanentemente:

```
PROCEDURE WriteSquare(Var p: real);
begin
  p := p*p;
  WriteLn(p:0:1);
end;
```

Agora, se corremos o programa, o output será

```
4.0
4.0
```

porque o valor de `x` mudou simultaneamente com o parâmetro `p`.



Passing by value *Passing by reference*

No análogo de visiualizar variáveis com caixas: *passing by reference* é dar a caixa (variável) ao procedimento e o procedimento pode usar e mudar o valor na caixa. No fim, o procedimento devolve a caixa com o novo valor. Enquanto, *passing by value* é equivalente com abrir a caixa, cópiar o conteudo (valor da variável) e só dar este valor ao procedimento. Obviamente, o valor original ficará na caixa e não mudará.

Num outro exemplo: Posso te dizer o saldo da minha conta no banco, o que podes usar para calcular o valor equivalente em dolares, ou vou te dar a direita de mudar o conteudo da minha conta. Provavelmente o saldo vai mudar permanentemente.

Mini Teste

Para testar o seu conhecimento, sobre o que aprendeu nesta aula, clique aqui para um teste on-line.

Peter Stallinga. Universidade do Algarve, 6 Abril 2002

Aula 16: Programação recursiva

Recursivo

Um procedimento é recursivo se está definido em termos de si própria

Exemplo 1: Factorial

O exemplo clássico é o cálculo do factorial *n*! Uma solução seria fazer isto com um ciclo, tal como está presentado nas aulas 11 e 12:

```
FUNCTION Factorial(n: integer): integer;
 (* returns n! *)
var result: integer;
    i: integer;
begin
  result := 1;         (* initialize the variable *)
  for i := 1 to n do
    result := i*result;
  Factorial := result;  (* return result *)
end;
```

De facto, esta função, retorna o factorial do argumento, por exemplo `Factorial(5)` = 120. Verifica isto!

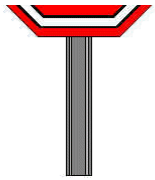
Mais elegante é uma solução que usa recursividade. Vamos escrever uma função que está definida em termos de si própria, ou seja chama si própria, tal como nós aprendemos na escola,

$n! = n*(n-1)!$

Vamos fazer exactamento isso:

```
FUNCTION Factorial(n: integer): integer;
 (* returns n! *)
begin
    (* the value to be returned is expressed in terms of itself: *)
  Factorial := n*Factorial(n-1);
end;
```

Esta função já está quase correcta. O único problema é que ... nunca vai acabar. Por exemplo, se nós chamamos a função com Factorial(4), a função tentará calcular 4*Factorial(3) e, por isso, vai chamar Factorial(3). Factorial(3) tentará calcular 3*Factorial(2), o que implica chamar Factorial(2), ... que vai chamar Factorial(1) ... que



vai chamar Factorial(0) ... que vai chamar Factorial(−1) ... que vai chamar Factorial(−2) ... e isto nunca vai acabar. A função nunca vai retornar nada e o computador vai *crash*er, provavelmente com o erro "stack overflow". Obviamento temos de incluir uma maneira de parar o ciclo recursivo. Agora, lembra que em matemática tivemos também uma condição de parar, para o Factorial este semente foi a definição

$1! = 1$

Vamos incluir a mesma coisa na nossa função:

```
FUNCTION Factorial(n: integer): integer;
 (* returns n! *)
begin
  if n=1 then
    Factorial := 1
  else
    Factorial := n*Factorial(n-1);
end;
```

A idea nós podemos extrair deste exemplo é que **sempre precisa de incluir uma maneira de acabar chamar a função recursiva**. Tal como nos ciclos repeat-until e while-do temos de dar a possibilidade de sair dos cálculos. Senão o programa nunca vai acabar.

Exemplo 2: Fibonacci

Lembra das aulas práticas, a definição dos números Fibonacci é também dado em termos de si própria:
$f_n = f_{n-2} + f_{n-1}$
Com as condições de parar (os sementes)
$f_1 = 1$
$f_2 = 1$
Por exemplo:
$f_3 = 1 + 1 = 2$
$f_4 = 1 + 2 = 3$
$f_5 = 2 + 3 = 5$
$f_6 = 3 + 5 = 8$
$f_7 = 5 + 8 = 13$

Podemos implementar isto numa função recursiva. Nota a condição de parar:

```
FUNCTION Fibonacci(n: integer): integer;
begin
  if (n=1) OR (n=2) then
    Fibonacci := 1
  else
    Fibonacci := Fibonacci(n-2) + Fibonacci(n-1);
end;
```

Variáveis

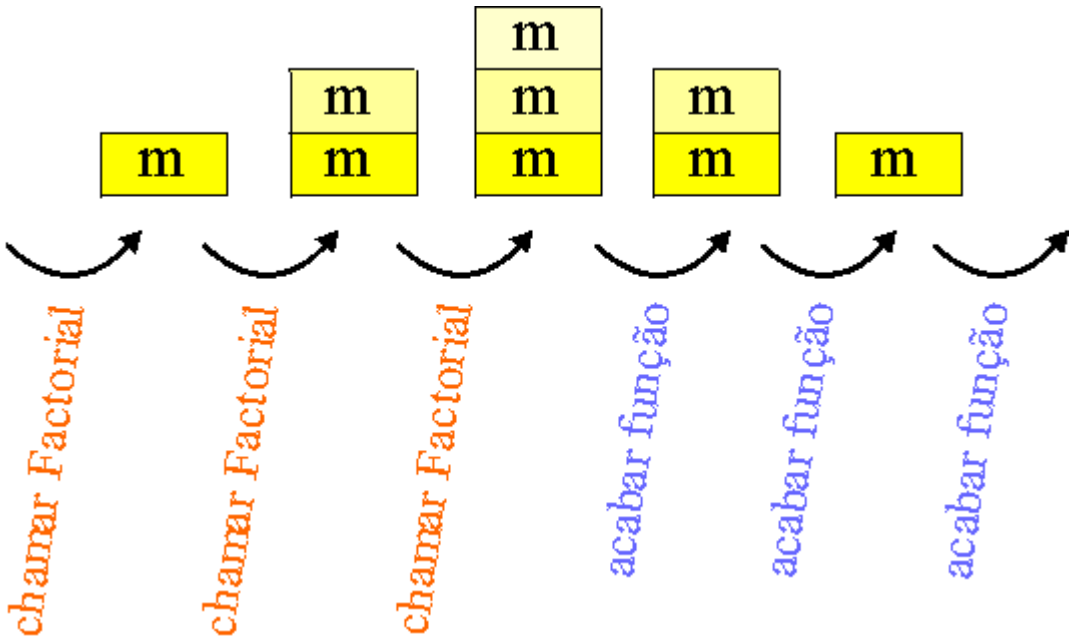
Variáveis declaradas dentro de um procedimento recursivo são todas variáveis locais. Além disso, cada vez o procedimento será chamado, uma nova versão ou cópia (em inglês: *instance*), ou "caixa" da variável será creada o que existe até o procedimento acabará. Cada versão, embora de ter o mesmo nome, fica num endereço diferente na memória, ou seja são caixas diferentes. Como exemplo, vamos introduzir uma

variável na nossa função Factorial:

```
FUNCTION Factorial(n: integer): integer;
Var m: integer;
begin
  m := 2*n;
  if n=1 then
    Factorial := 1
  else
    Factorial := n*Factorial(n-1);
  WriteLn(m);
end;
```

Se vamos chamar esta função com argumento 3 a seguinte vai acontecer:

```
  Factorial(3) é chamada
    uma variável com nome m é criada
    2*3 é atribuido a este m
    Factorial(2) é chamada
      uma variável com nome m é criada (diferente do que o m acima!)
      2*2 é atribuido a este m
      Factorial(1) é chamada
        uma variável com nome m é criada (diferente do que os m acima)
        2*1 é atribuido a este m
        ... chega a condição de parar e Fortorial retorna 1;
        O procedimento mostra o valor de m: 2
      saí da Factorial(1)
      O procedimento mostra o valor de m: 4
    saí da Factorial(2)
    O procedimento mostra o valor de m: 6
  saí da Factorial(3)
```



O que nós podemos aprender aqui é que a variável `m` não é um objecto estático na memória, sempre no mesmo endereço, mas sim a variável será criada cada vez que o procedimento será chamado. Existem quantas versões da variável igual ao nível de profundidade de chamar a função. Além disso, só a última versão pode ser usada na função. Só temos acesso à última versão (até o momento que este nível de chamar a função acebará).
(*em C é possivel impedir a criação das novas versões da variável; se nós sempre queremos usar a mesma "caixa" podemos por a palavra "static" em frente da declaração da variável*)

Mini Teste

Para testar o seu conhecimento, sobre o que aprendeu nesta aula, clique aqui para um teste on-line.

Peter Stallinga. Universidade do Algarve, 7 Abril 2002

Aula 17: Array

Array

Imagine queríamos fazer um programa para calcular a média de uma lista de 10 números. Com a matéria que aprendemos até agora, terias de fazer qualquer coisa deste estilo

```
Var a1, a2, a3, a4, a5, a6, a7, a8, a9, a10: real;
    average: real;
```

e depois o cálculo:

```
average := (a1 + a2 + a3 + a4 + a5 + a6 + a7 + a8 + a9 + a10) / 10;
```

Isto é uma grande chatice. Ainda pior, imagine queremos pedir ao utilizador o número de números a usar no cálculo da média:

```
Var n: integer;
   |
ReadLn(n);
Case n of
   1: average := a1;
   2: average := (a1 + a2) / 2;
   3: average := (a1 + a2 + a3) / 3;
        |
  10: average := (a1 + a2 + a3 + a4 + a5 + a6 + a7 + a8 + a9 + a10) / 10;
end;
```

É aqui que surge o conceito de array. Um array pode guardar muitas variáveis do mesmo tipo com acesso fácil, através um índice. Tal como em matemática, onde a_i é elemento número i de um vector ou série a, `a[i]` é elemento i de um array `a`.

Um array é uma colecção indexada de variáveis do mesmo tipo.

Declaração de um array

Para declarar um array usamos a seguinta sintaxe:

```
Var name: array[startindex .. endindex] of type;
```

`name` é o identificador (nome) do array, da mesma forma dos nomes paras as outras variáveis. **`startindex`** e **`endindex`** definem os limites dos índices do array. Nota que, em PASCAL, o começo do array não é necessariamente igual a 0. Em vez disso, o array pode começar à vontade. Isto é simpático, porque pessoas normalmente gostam de trabalhar com índices começando com 1 em vez de 0. Al«em disso, o array pode começar com qualquer outro índice (13, 100, –100, tanto faz).

`type` é qualquer tipo da variável que nós já conhecemos, por exemplo real ou integer, mas pode também ser um outro array.

Exemplos:

```
Var account: array[1..100] of real;
```
Isto podia guardar a informação de 100 contas bancárias.

```
Var prime: array[1..10] of longint;
```
Isto podia guardar os primeiros 10 números primos.

```
Var propinas: array[1000..2000] of boolean;
```
Isto podia guardar alguma informação (do tipo boolean) sobre a condição dos alunos com números entre 1000 e 2000, por exemplo se eles pagáram propinas este ano. Com certeza, a nossa universidade tem um array deste tipo no algum sítio.

Usar um array

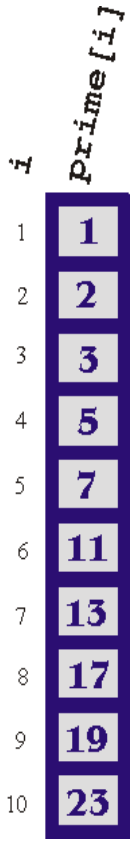
Dentro do programa podemos usar um elemento de um array através

name[index]

`name[index]` retornerá o valor do elemento `index` do array `name`. A declaração do array determine o tipo de valor do cada elemento
Exemplos: Os array da secção anterior:

```
account[20]
```
é o valor - do tipo real - do elemento 20 do array com nome `account`.

```
prime[8]
```
é o valor ' do tipo longint ' do elemento 8 do array com nome `prime`. Os números aqui ao lado direito podiam representar este array. Então, elemento 8 seria igual a 17. Está claro que o nosso programa deve preencher este array do qualquer modo. Senão o array não conterá os números primos.




```
propinas[1055]
```
é TRUE ou FALSE (tipo boolean). Elemento 1055 do array `propinas`. O aluno 1055 já pagou as propinas? Sim ou não?

Podemos também usar uma variável para o índice do array. Evidentemente, esta variável deve ser de um tipo inteiro, porque o ídice é alguma coisa contável; índice 3.4981 não faz sentido. Índice 3 faz, enderecerá o terceiro elemento do array. O código a seguir mostrará o conteúdo do array `account` (20 elementos):

```
for i := 1 to 20 do
  WriteLn(account[i]);
```

Arrays com mais de uma dimensão

Tal como em matemática, onde temos vectores (tensores de uma dimensão) e matrizes (tensores de duas

dimensões) em programação existem arrays de uma ou duas ou mesmo mais dimensões. Por exemplo, a declaração de um '*double array*' (um array de duas dimensões):

```
Var name: array[startindex1 .. endindex1, startindex2 .. endindex2] of type;
```

Na verdade, é também possível declarar este array assim:

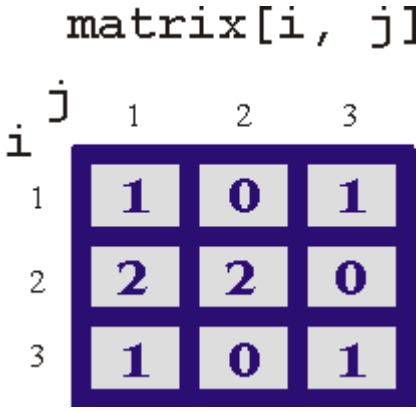
```
Var name: array[startindex1..endindex1] of array[startindex2..endindex2] of type;
```

o que mostra bem o que é um array de duas dimensões, nomeademente um array de arrays. Neste maneira o computador arranja os arrays. Por outro lado, o primeiro método de declarar um array de duas dimensões é mais lógico para as pessoas. Por isso, este modo de declaração é preferido.

O uso do um array de duas dimensões é parecido com o uso de um array de uma dimensão. Temos de seperar os índices com uma virgula ou com parênteses quadradras:

```
name[index1, index2]
name[index1][index2]
```

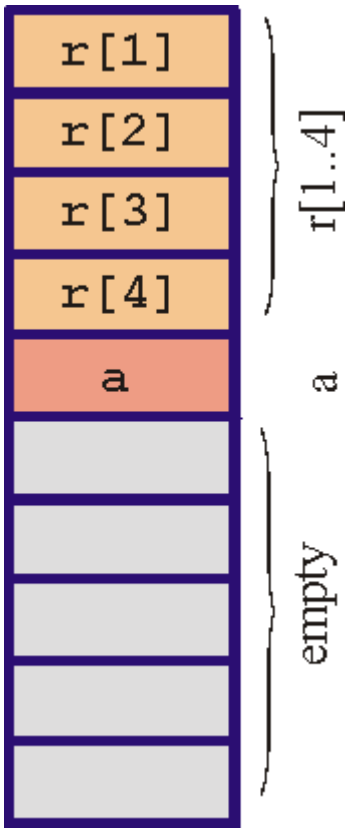
Um exemplo: para mostrar a matriz aqui ao lado esquerde o código a seguir pode ser usado. Nota que o array consiste de 9 (3x3) elementos do tipo integer.




```
PROGRAM ShowMatrix;

Var matrix: array[1..3, 1..3] of integer;

begin
  matrix[1, 1] := 1;
  matrix[1, 2] := 0;
  matrix[1, 3] := 1;
  matrix[2, 1] := 2;
  matrix[2, 2] := 2;
  matrix[2, 3] := 0;
  matrix[3, 1] := 1;
  matrix[3, 2] := 0;
  matrix[3, 3] := 1;
  for i := 1 to 3 do
    begin
       for j := 1 to 3 do
         Write(matrix[i, j],' ');
      WriteLn;
    end;
end.
```

Aviso

Com os arrays temos de haver sempre cuidade de usar índices válidos. Isto significa usar índices no gama especificado na altura de declaração. O computador só vai reservar espaço na memória suficiente para guardar esses variáveis. nada mais e nada menos. Se usamos um ídice 'out of bounds' (fora da gama) o resultado do programa pode tornar-se muito estranho.
Para esclarecer isto num exemplo: O programa a seguir declara um array de 4 integers `r[1..4]` e uma variável normal do mesmo tipo (integer) `a`. A figura ao lado mostra como a memória está organizada depois desta declaração. Agora, o que vai acontecer na linha onde vai atribuir um valor a `r[5]`? Se `r[5]`existeria, este `r[5]` ficaria no lugar que agora está ocupado por `a`. A maioria das linguagens de programação não preocupam-se com isso e vão escrever neste lugar, e **vamos perder o valor de `a`**.




```
PROGRAM Test;

Var r: array[1..4] of integer;
Var a: integer;

begin
  a := 0;
  WriteLn('a=',a);
  r[5] := 1;
  WriteLn('a=',a);
  ReadLn;
end.
```

O *output* do programa provavelmente será

```
a=0
a=1
```

Há lingaugens de programação que deixam nos especificar que nós queremos que o programa vai sempre verificar se os índices são válidos. Por exemplo o Turbo PASCAL. Mas, o Dev-PASCAL não tem este opção. Com este verificação o programa fica mais lente e ocupa mais espaço na memória e no disco, mas pode ajudar nos em eliminar erros do nosso programa. A verificação do índices chama-se *range-checking* (verificação do gama) e se o programa encontra um índice inválido vai gerar um '*range-check error*' (erro na verificação do gama).

"Hurray! Sei tudo dos arrays!"

Quick Test

To test your knowledge of what you have learned in this lesson, click here for an on-line test.

Peter Stallinga. Universidade do Algarve, 13 Abril 2002

Aula 18: Records

Record

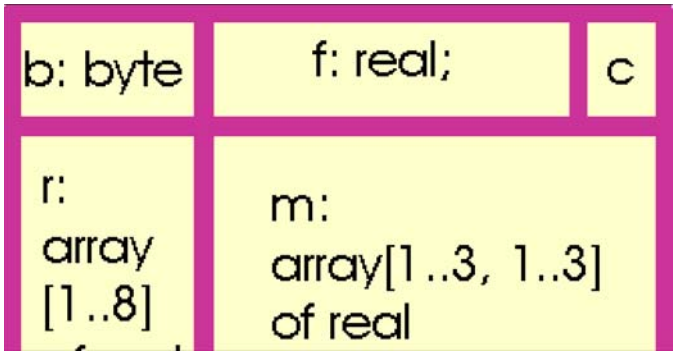
Na aula anterior aprendemos como armazenar variáveis do mesmo tipo num array. Bem organizado com um índice, tal como num armário de gavetas com na cada gaveta o mesmo tipo da informação. Se nós queremos juntar variáveis que não são do mesmo tipo, podemos fazer isto com um **record**.

Os três armários armazem coisas do mesmo tipo, tal como os arrays. Por exemplo, o armário esquerdo armazene "bytes", no meio é para "integers" e o armário direito é para "reals".

*O armário no centro está usado para armazenar coisas do tipo misto. Da mesma forma, um **record** é usado para armazenar variáveis dos tipos diferentes, integers, reals, or qualquer outro tipo, na mesma caixa.*

Um record é uma colecção de variáveis do tipo misto.

Declaração de um record



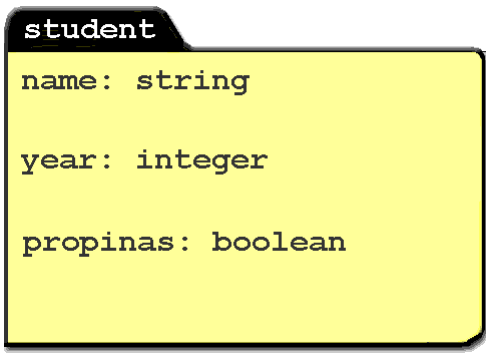
Uma visualização de um record, nonemademente uma colecção das variáveis. Cada variável dentro de um record chama-se um campo. Aqui temos 5 campos: um byte (b), um real (f), um boolean (c), um array simples de reals (r) e um array dobra de reals (m).

Para declarar um record usamos a seguinta

```
Var name:
   record
        item1: type1;
        item2: type2;
            |
        itemN: typeN;
   end;
```

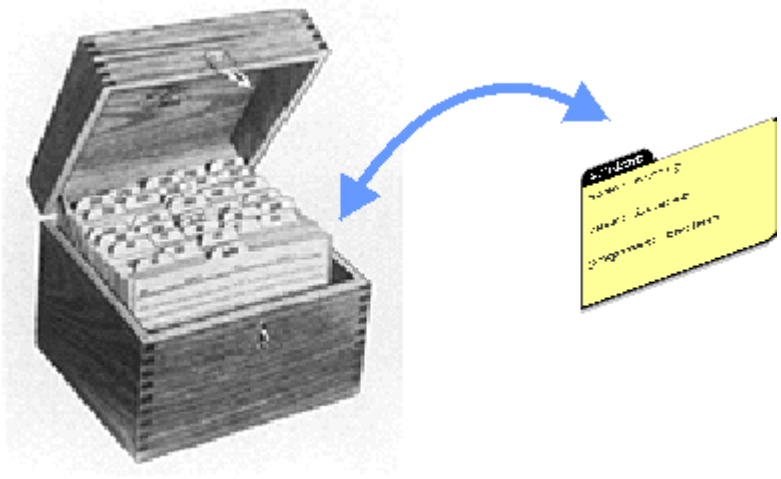
com
name: o nome para o record inteiro.
item1..itemN: os nomes dos campos no record. Estes nomes seguem as mesmas regras para identificadores como as variáveis, constantes, procedures, etc. Nota que podemos pôr tanto campos como nos queremos, com qualquer combinação dos tipos.
type1..typeN: os tipos dos campos do record.

Como exemplo, a declaração de um record para armazenar informação de um aluno pode conter os campos name, year, e propinas:




```
Var student:
  record
    name: string;
    year: integer;
    propinas: boolean;
  end;
```

Este record só consegue armazenar a informação de um aluno. No entanto, com o saber da aula anterior (aula 17), é possível construir um array capaz de armazenar records de muitos alunos. vamos construir um array de 1000 alunos:



```
Var students: array[1..1000] of
  record
    name: string;
    year: integer;
    propinas: boolean;
  end;
```

Mais tarde vamos ver que é muito mais fácil fazer isto através a definição de um novo tipo da variável (see aula 19).

Usar um record

Para ter acesso a um record usamos o formato

name.field

Por exemplo, para atribuir valores ao record `student` podemos fazer o seguinte

```
student.name := 'Peter Stallinga';
student.year := 2002;
student.propinas := TRUE;
```

ou no exemplo de um array de records `students`:

```
i := 1055;
students[i].name := 'Peter Stallinga';
students[i].year := 2002;
students[i].propinas := TRUE;
```

Nota a estrutura dos arrays de records. `students` é um array de records, portanto, `students[i]` é um dos elementos do array e, por isso, é um record. Se queremos atribuir um valor a um campo deste record usamos o ponto final e o nome do campo, então `students[i].name` é um string que conte o nome do aluno com número i.
Sintaxe errada seria `students.name[i]` (isto podemos usar se `students` for um único record que conte um campo `name` do tipo array)
Também errado: `students.[i]name`. Esta estrutura não faz sentido nenhum.

Outro exemplo:

```
Var coordinate:
  record
    x: real;
    y: real;
  end;

coordinate.x := 1.0;
coordinate.y := 0.0;
```

Os campos do record coordinate não são dos tipos mistos, então in princípio é possível fazer isto com um array:

```
Var coordinate: array[1..2] of real;

coordinate[1] := 1.0;
coordinate[2] := 0.0;
```

mas, a primeira versão tem preferência; com um record é mais legível.
Outro exemplo:

```
Var address:
  record
```

```
    street: string;
    housenumber: integer;
    andar: integer;
    porta: char;
  end;

address.street := 'Rua Santo Antonio';
address.housenumber := 34;
address.andar := 3;
address.porta := 'E';

writeln(address.street, '  ', address.housenumber);
writeln(address.andar, address.porta);
```

No exemplo acima precisava de escrever muitas vezes a palavra "address". Para poupar tempo, em PASCAL existe a combinação "with recordname do", o que significa que na instrução a seguir (ou tudo entre "begin" e "end") as variáveis levam o recordname em frente. Por isso, o código acima pode ser reescrivado na forma seguinta mais legível

```
with address do
   begin
     writeln(street, '  ', housenumber);
     writeln(andar, porta);
   end;
```

Quick Test

Para testar o seu conhecimento, sobre o que aprendeu nesta aula, clique aqui para um teste on-line.

Peter Stallinga. Universidade do Algarve, 16 Abril 2002

Aula 19: Definir novos tipos

Type

Às vezes é bom declarar um novo tipo de variável para usar mais tarde no programa. Para manter o código mais legível, ou para evitar escrever o mesmo código muitas vezes. Definir um novo tipo de variável é assim:

Type typename = description;

com `typename` o nome que queremos dar ao novo tipo e `decription` a descrição do novo tipo que pode incluir qualquer combinação (arrays e records) de tipos de variáveis ou tipos simples. Pode também ser um tipo apontador que nós vamos aprender na aula seguinta (aula 20).

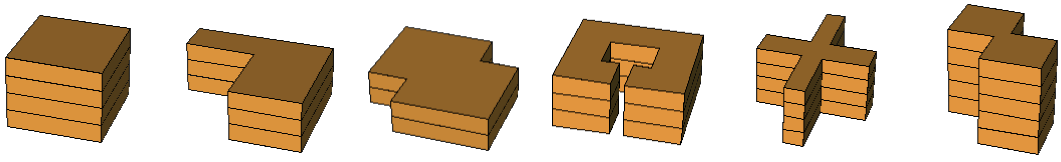
Exemplos:

```
Type float = real;
```

Isto é util para programadores que prefirem (ou costumam) programar na linguagem C. Depois este declaração, será possível a utilização das variáveis do tipo float, como em C.
Mais exemplos:

```
Type realarray = array[1..10] of real;

Type myrecord =
  record
    name: string;
    length: real;
    width: real;
    height: real;
  end;
```

Nota que definir um novo tipo **não vai criar uma nova variável!** Não vai reservar espaço na memória e não atribue um valor a uma variável. Só vai especificar ao compilador um novo tipo da variável que nós podemos usar mais tarde numa declaração de uma nova variável.



... definir novas caixas para as variáveis.

Usar o novo tipo

Após a especificação do novo tipo, podemos declarar uma nova variável deste tipo:

Var varname: typename;

com `varname` o nome da nova variável, e `typename` o tipo. Este declaração é exactamente igual à declaração das variáveis normais da aula 5. Só que agora podemos usar os nossos novos tipos.

Exemplos:

```
Var f: float;
```

Isto (agora) é muito parecido com a linguagem C (*em C seria "`float f`"*)

```
Var ra: realarray;
```

vai declarar uma variável do tipo realarray. E no código podemos usar este novo array, por exemplo

```
ra[1] := 2.68;
```

Isto é equivalente com

```
Var ra: array[1..10] of real;
ra[1] := 2.68;
```

O útlimo exemplo

```
Type myrecord =
  record
    name: string;
    length: real;
    width: real;
    height: real;
  end;
Var mydata: array[1..100] of myrecord;
mydata[23].length := 3.1;
```

Mais exemplos

```
PROGRAM WithTypeDefinition;

Type ra = array[1..6] of integer;
Var x: ra;
    y: array[1..7] of integer;

FUNCTION AreEqual(r: ra): boolean;
(* Nota que o novo tipo pode também ser usado para os parametros *)
begin
  if r[1]=r[2] then
    AreEqual := TRUE;
  else
    AreEqual := FALSE;
end;

begin
  x[1] := 1;
  x[2] := 0;
  WriteLn(AreEqual(x));
  y[1] := 1;
  y[2] := 0;
```

```
   (* A seguinta linha de código não é permitido, porque o tipo de y e o tipo que a
função 'AreEqual' está a esperar são diferente. Nota a diference no tamanho dos arrays.
*)
  WriteLn(AreEqual(y));
end.
```

```
PROGRAM WithTypeDefinition;

Type time =
  record
    hour, minute, second: integer;
  end;

PROCEDURE ShowTime(t: time);
  (* Vai mostrar o tempo no formato h:m.s *)
begin
  WriteLn(t.hour, ':', t.minute, '.', t.second);
end;

Var atime: time;

begin
  atime.hour := 23;
  atime.minute := 16;
  atime.second := 9;
  ShowTime(atime);
end.
```

---

um record de records:

```
Type date =
  record
    day, month, year: integer;
  end;
Type time =
  record
    hour, minute, second: integer;
  end;
Type dateandtime =
  record
    dattime: time;
    datdate: date;
  end;
Var x: datendtime;

x.dattime.hour := 1;
```

`x` é um record que contem dois campos. Um campo chama-se `dattime` o que é um record de três campos. Um destes campos é `hour`.

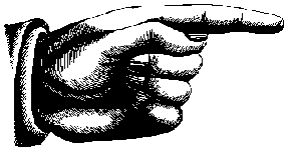
Mini Teste

Para testar o seu conhecimento, sobre o que aprendeu nesta aula, clique aqui para um teste on-line.

Peter Stallinga. Universidade do Algarve, 18 Abril 2002

Lecture 20: Apontadores

Apontador



Um apontador é um tipo de variável especial. O apontador não contem informação útil, mas só contem o endereço da informação.

Um apontador contem o endereço de um lugar na memória

Isto é parecido com guardar um endereço de um apartemento no meu livro de endereços. É só um endereço e nada mais.
Para declarar um apontador usamos a sintaxe seguinta:

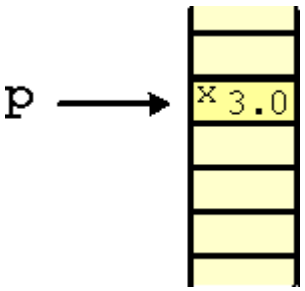
```
Var p: pointer;
```

Instruções para apontadores

Existem duas instruções básicas para apontadores em PASCAL

@x ou **Addr(x)** retorna o endereço do objecto **x**

p^ é o conteúdo do endereço **p**



No exemplo aqui ao lado, `x` é uma variável do tipo real com valor 3.0. Nós queremos que o apontador `p` aponta à esta variável. Por isso usamos a instrução

```
p := @x;
```

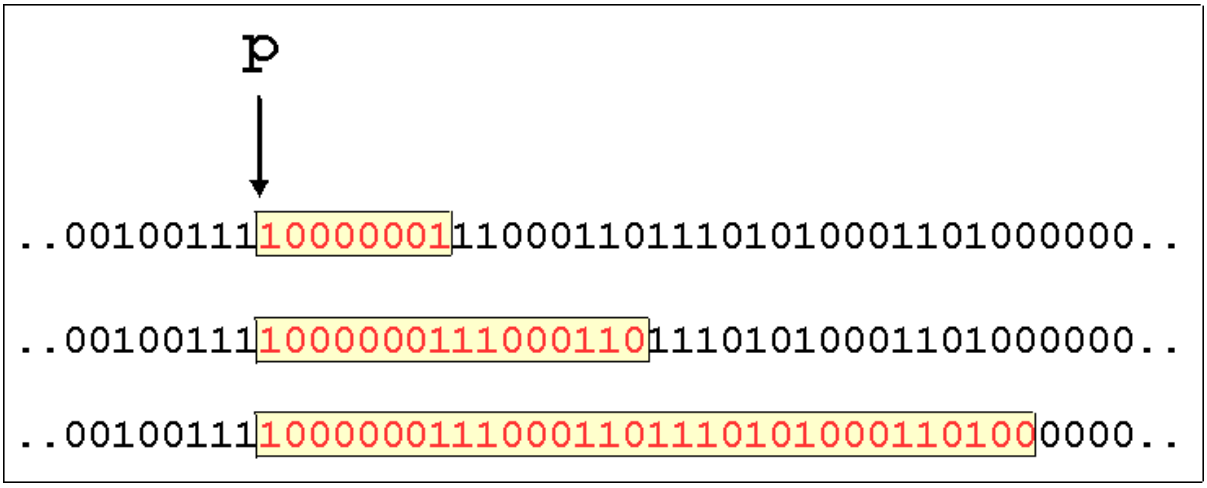
ou

```
p := Addr(x);
```

O valor de `p` é agora um endereço, nomeademente o endereço da variável `x`. Para mostrar o conteúdo deste endereço a idea poderia ser usar a instrução

```
WriteLn(p^);
```

Isto tem um problema. O compilador não sabe o que é que o `p` está a apontar. Qual o **tipo** de informação fica no endereço `p`. Quando o programa corre, o p apontará um ponto na memória sem saber o tipo de conteúdo deste endereço. Imagine a seguinta situação: O apontador `p` está a apontar um endereço. Se o conteúdo é um byte, o valor será diferente do que se o conteudo é um word. No exemplo abaixo, `p^` aponta um byte com valor 129 (binário: 10000001), se o mesmo `p^` apontarava um valor de tipo word o valor seria 25473 (binary: 0110001110000001), ou se `p^` apontarava um longint (4 bytes, 32 bits) o valor seria 743924609 (binário: 00101100010101110110001110000001). (Nota que nos computadores baseados nos precessadores Intel os números são armazenado com o bit menos significativo ao primeiro [LSB=least significant bit]).



Por isso temos de especificar o tipo de informação o apontador aponta. Para especifiacr isso temos de regressar à declaração do apontador. Em vez de só dizer que p é um apontador temos de especificar também o tipo:

Declaração de um apontador

A declaração de um apontador fazemos assim:

```
Var p: pointer;
Var p: ^type;
```

A primeira forma declara um apontador geral, sem especificar o tipo de informação no endereço `p`.
A segunda forma também especifica o tipo da informação no endereço `p`. O tipo (type) pode ser qualquer tipo que nós sabemos: tipos simples (integer, real, etc), tipos complicados (record, array), e mesmo combinações de tipos (arrays de records, records de arrays).
Exemplos:

```
Var byteptr:  ^byte;
    wordPtr:  ^word;
    real6arrayptr: ^array[1..6] of real;
```

Agora vamos verificar o nosso saber. Vamos declarar um apontador que aponta um tipo word e atribuimos o endereço de um word ao apontador:

```
Var wordptr: ^word;
    w: word;

begin
   (* atribuir um valor ao word w: *)
  w := 25473;
   (* atribuir o endereço ao apontador: *)
  wordptr := Addr(w);
   (* Mostrar o conteudo do endereco: *)
  WriteLn(wordptr^);
end.
```

output:

```
25473
```

Agora vamos complicar coisas. Vamos declarar um apontador do tipo 'apontar um **byte**', e atribuimos o endereço de um **word**. Vamos ver o que acontecerá:

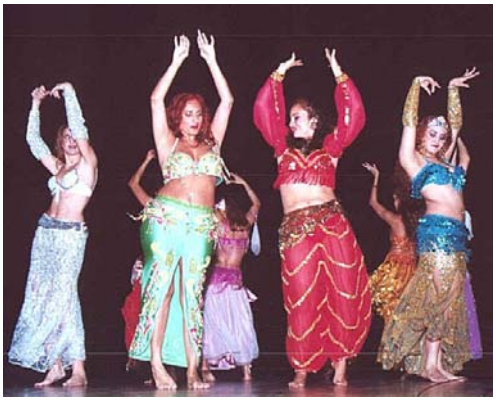
```
Var byteptr: ^byte;
    w: word;

begin
   (* assign a value to the word *)
  w := 25473;
   (* let a 'pointer to word' point to our word *)
  byteptr := Addr(w);
   (* show the contents of the memory wordptr points to *)
  WriteLn(byteptr^);
end.
```

output:
```
129
```

Este exemplo mostra que temos de haver cuidade com os apontadores. O valor de conteúdo de endereço `p` depende do tipo de `p`!

Pourquê?

Porquê usar apontadores? As coisas são mais complicados, mas há várias razões para usar apontadores. As mais importantes são



- Rapidez
- Flexibilidade

Rapidez: Imagine que quer escrever um procedimento com um parâmetro um array. Cada vez o programa chama este procedure, o array será copiado. Se, em vez disso, chamamos o procedimento com o **endereço** do array o programa correrá muito mais rápido. Isto implica só copiar uma única pequena variável (um apotador ocupa só 4 bytes [Intel computadores]). Análise os programas a seguir. O programa esquerde usa um array, o programa direito usa um apontador para chamar o procedure.

```
PROGRAM TestSpeed;

Type ra = array[1..1000] of real;
Var r: ra;
    i: longint;

PROCEDURE SlowProc(a: ra);
begin
  a[1] := 1.0;
end;

begin
  for i := 1 to 4000000 do
    SlowProc(r)
end.
```

tempo de execução (Pentium II 450 MHz, TURBO PASCAL 6): **115** s.

```
PROGRAM TestSpeed;

(* define an array and a pointer to
  that arrray: *)
Type ra = array[1..1000] of real;
     rap = ^ra;
Var r: ra;
    i: longint;

PROCEDURE FastProc(a: rap);
begin
  (* a is a pointer to an array *)
  (* a^ is what it points to, the *)
  (* the array. a^[1] is the first *)
  (* element of that array *)
  a^[1] := 1.0;
end;

begin
  for i := 1 to 4000000 do
    FastProc(@r)
   (* pass only a pointer to *)
   (* the procedure *)
end.
```

tempo de execução (Pentium II 450 MHz, Turbo PASCAL 6): **1** s.

(De facto o programa de lado direito é igual a um programa que usa a técnica de *passing by reference* (veja aula 15). `PROCEDURE SlowProc(Var a: ra);` também seria rápido. *Passing by reference* significa o apontador é dado ao procedimento)

Flexibilidade: Se, no começo do programa não sabemos quantas variáveis preciçamos, temos de declarar o máximo possível para garantir de ser capaz de correr o programa. Se nós queremos escrever um programa para calcular os primeiros N números primos, com N dado pelo utilizador, temos de declarar um array do máximo tamanho. Assim:

```
Var prime: array[1..10000000] of longint;
```

Este programa vai ocupar a memória inteira do computador, mesmo quando vamos só calcular 10 números. Não haverá mais lugar para os outros programas ou outras variáveis no mesmo programa. Mais bonito seria usar um array de tamnho flexível para declarar as variáveis dinamicamente; especificar as variáveis precisas, nada mais e nada menos. Com apontadores isto é possível.

Os apontadores e a idea de criação dinâmica é a base de object-oriented programming (programação orientada pelos objectos), o que é a forma de programação moderna. Este forma de programar fica fora do âmbito desta cadeira.

NIL

Um apontador que não aponta nada tem o valor NIL. NIL é uma contante predefinida em PASCAL.

Quick Test

Para testar o seu conhecimento, sobre o que aprendeu nesta aula, clique aqui para um teste on-line.

Peter Stallinga. Universidade do Algarve, 26 Abril 2002

Aula 21: Ficheiros

Comunicar com ficheiros

Hoje vamos aprender como escrever e ler ficheiros. Neste aula vamos só usar ficheiros do tipo texto. Isto são ficheiros com os dados no formato ASCII e legível pelas pessoas enquanto que os outros formatos são do tipo binário e só legível pelo computador. Todos os nossos ficheiros com os programas de PASCAL são do tipo ASCII.
Podemos ler e escrever ficheiros na disquete ou disco rigido ou mesmo do CD-ROM (neste caso - obviamente - é só possível ler os ficheiros).

Instruções

As seguintas instruções de PASCAL são relacionado ao acesso dos ficheiros:

```
Var text
Assign
Rewrite
Reset
Close
Read, ReadLn
Write, WriteLn
Eol, Eof
```

Declarar ficheiros:

Antes de abrir um ficheiro temos de especificar um variável que vai guardar a informação do ficheiro. Em PASCAL a declaração é assim:

```
Var filehandle: text;
```

Com `filehandle` o nome da variável que contem o estado do ficheiro. Isto não é igual ao nome do ficheiro, mas é só uma variável que vai guardar a informação do estado do mesmo (por exemplo: onde fica no disco?, está aberto?, a última unidade da informação).
O lugar de declarar este variável é com a declaração das outras variáveis.
Example:

```
Var f: text;
```

Atribuir um nome

Dentro do programa temos de especificar o nome (como está no disco) antes de abrir o ficheiro:

```
Assign(filehandle,filename);
```

O `filehandle` é igual à variável acima e `filename` é um string (constante ou variável) que contem o nome do ficheiro. Por exemplo:

```
Assign(f, 'MYFILE.TXT');

ReadLn(s);
Assign(f, s);
```

Input ou *output*?

Para abrir o ficheiro, existem duas formas. A escolha da forma depende do tipo da comunicação input (ler) ou output (escrever):

```
Reset(filehandle);
Rewrite(filehandle);
```

Por exemplo:

```
Reset(f);
Rewrite(f);
```

Ler e escrever

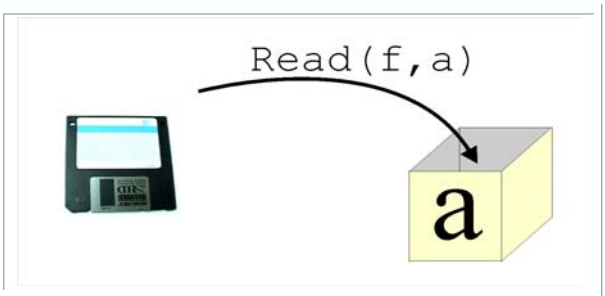
Ler e escrever é igual a ler e escrever através o teclado e o ecrã, respectivamente. Usamos as mesmas instruções (Read, ReadLn, Write, and WriteLn). A única diferença é que leve mais um parâmetro, nomeademente o ficheiro:

```
Read(filehandle,...);
ReadLn(filehandle,...);
Write(filehandle,...);
WriteLn(filehandle,...);
```

Nota que podemos só usar as instruções `Read` e `ReadLn` para ficheiros que foram abertos para *input* (por `Reset`). Da mesma forma as instruções `Write` e `WriteLn` é para escrever nos ficheiros que estão preperados para *output* (por `Rewrite`). Exemplos:

```
WriteLn(f, r:0:2);
ReadLn(f, opcao);
```

Ler uma variável com o teclado



Ler uma variável de um ficheiro

Fechar o ficheiro

Quando estamos pronto com o acesso do ficheiro temos de fechar-o. Isto é especialmente importante para ficheiros de *output*. Se esquecemos de fechar o ficheiro os dados no ficheiro não estarão completos. Fechar um ficheiro fazemos assim:

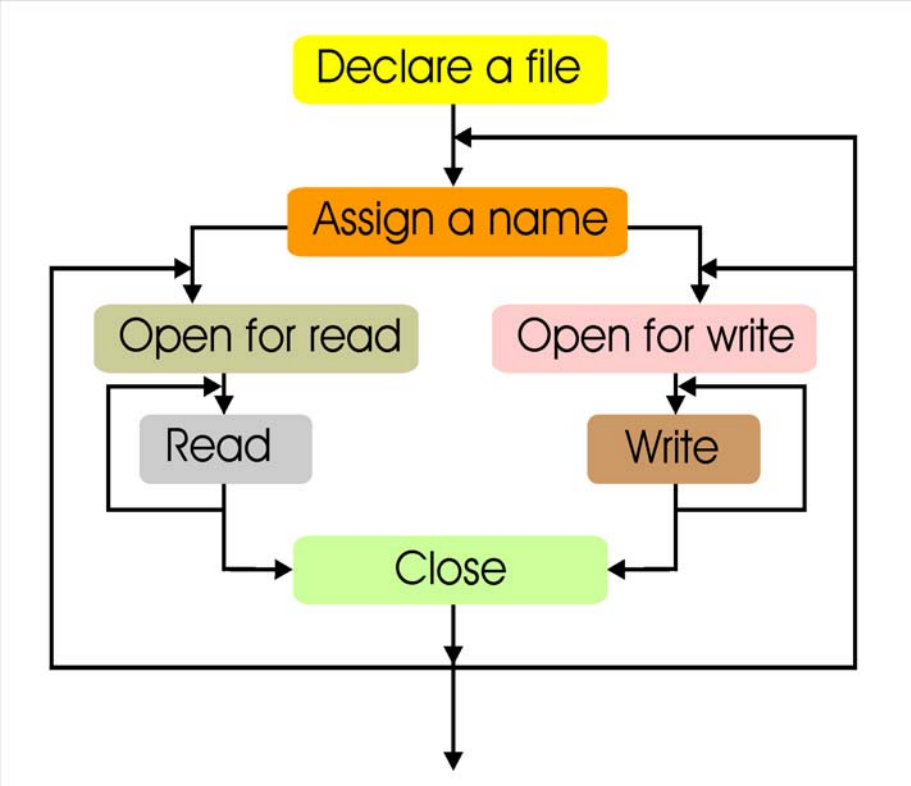
```
Close(filehandle);
```

por exemplo:

```
Close(f);
```

Resumo

Escrever e ler um ficheiro tem sempre os seguintes passos:



Eol, Eof

Duas instruções úteis sãol

Eol(filehandle): retorna TRUE se estamos a ler no fim da linha.
Eof(filehandle): retorna TRUE se estamos a ler no fim do ficheiro.

Exemplo:

```
While NOT Eof(f) do
  ReadLn(s);
```

o que vai ler até o fim do ficheiro.

Exemplos

código PASCAL	*ecrã*	*ficheiro TEST.TXT* ***após*** *de correr o programa*

`PROGRAM WithFileOutPut;`
 	
`Var f: text;`
`  s: string;`
`  i: integer;`
 	
`begin`
`  WriteLn('Nome do Ficheiro:');`
`  ReadLn(s);`
`  Assign(f, s);`
`  Rewrite(f);`
`  for i := 1 to 10 do`
`  WriteLn(f, i, ' Ola');`
`  Close(f);`
`end.`
	`Nome do Ficheiro:`
	`TEST.TXT`
		`1 Ola`
		`2 Ola`
		`3 Ola`
		`4 Ola`
		`5 Ola`
		`6 Ola`
		`7 Ola`
		`8 Ola`
		`9 Ola`
		`10 Ola`

código PASCAL	*ecrã*	*ficheiro TEST.TXT* ***antes*** *de correr o programa*

`PROGRAM WithFileInPut;`
 	
`Var f: text;`
`  c: char;`
`  s: string;`
`  i: integer;`
 	
`begin`
`  WriteLn('Nome do Ficheiro:');`
`  ReadLn(s);`
`  Assign(f, s);`
`  Reset(f);`
`  While NOT Eof(f) do`
`  begin`
`  (* le um caractere: *)`
`  Read(f, c);`
`  (* mostra no ecran: *)`
`  Write(c);`
`  end;`
`  Close(f);`
`end.`
	`Nome do Ficheiro:`
	`TEST.TXT`
	`1 Ola`
	`2 Ola`
	`3 Ola`
	`4 Ola`
	`5 Ola`
	`6 Ola`
	`7 Ola`
	`8 Ola`
	`9 Ola`
	`10 Ola`
		`1 Ola`
		`2 Ola`
		`3 Ola`
		`4 Ola`
		`5 Ola`
		`6 Ola`
		`7 Ola`
		`8 Ola`
		`9 Ola`
		`10 Ola`

Mini Teste

Para testar o seu conhecimento, sobre o que aprendeu nesta aula, clique aqui para um teste on-line.

Peter Stallinga. Universidade do Algarve, 30 Abril 2002

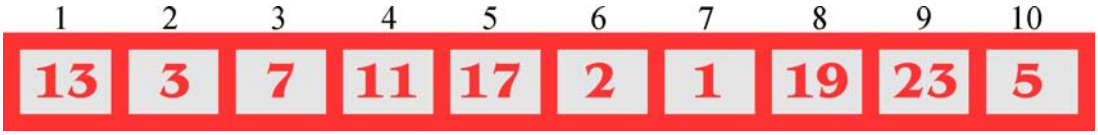
Aula 22: Algoritmos

Algoritmo

Um algoritmo é uma descrição de um método de resolver um problema. A palavra vem do nome de um matemático, Mohammed ibn-Musa Al-Khowarizmi, que viveu no Baghdad nos anos 780 - 850 dC.

Ordenar

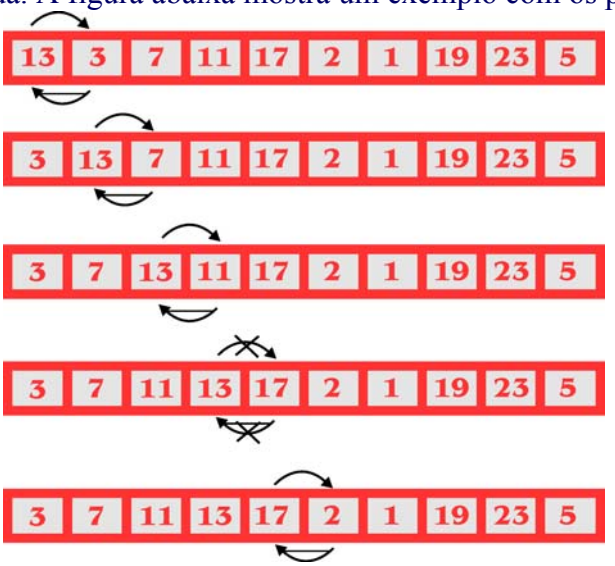
Para as nossas aulas isto significa que vamos conceber um método de atacar um problema mesmo sem mexer com o computador. Como exemplo vamos desenhar uma solução para ordenar um array de números. Imagine que temos um array de N integers. Como vamos atacar este problema?



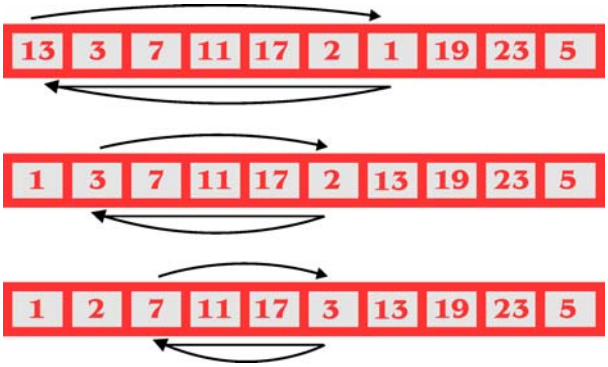
A figura mostra um array de 10 elements a ordenar.

Sem olhar nos detalhes (para já), podemos haver as seguintas soluções:

1. Olhar na lista. Se dois números consecutivos não estão ordenados corectos vamos trocar-os. Repete isto até a lista está ordenada. A figura abaixa mostra um exemplo com os primeiros passos.



2. Procura na lista o elemento mais pequeno. Põe este número no primeiro lugar. Depois procura o mínimo no resto dos números. Põe este número no segundo lugar. Repete isto N vezes e o array estará ordenado. A figura abaixa mostra os primeiros passos para o nosso array.



Bubble sort

Isto são dois algoritmos para ordenar um array. O primeiro chama-se "Bubble sort" (bubble = bolha, sort = ordenar), porque parece bolhas de ar na água que vão lentemente para o superfície. Vamos implementar esta idea em PASCAL. Ao primeiro]e útil desenhar um flow diagram (diagrama de fluxo).

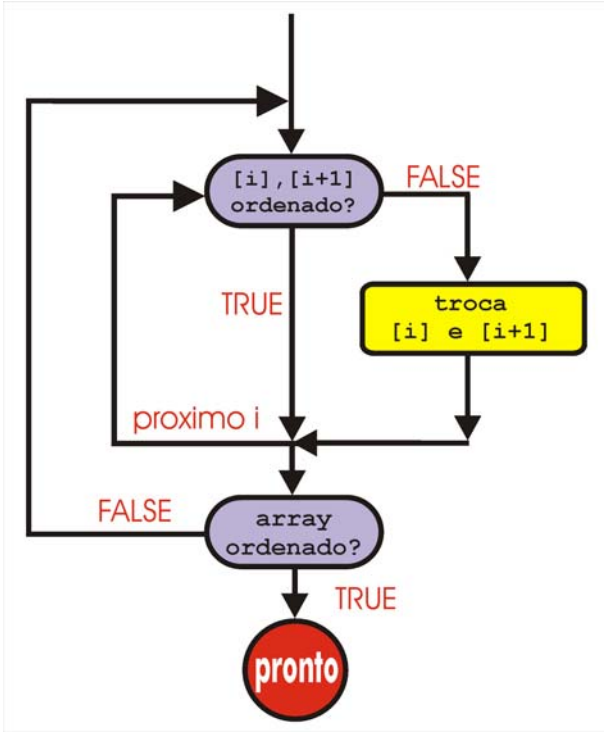


Diagrama de fluxo do algoritmo Bubble sort.
[i] significa elemento i do array.

Neste diagrama de fluxo já consegimos reconhecer a primeira idea de um programa PASCAL. Já é possível distinguir dois ciclos, um ciclo que verifica uma vez a ordenação de todos os pares de números e o outro ciclo que repete estes passos do array até TUDO está correcto. Além disso temos uma instrução para trocar dois números e uma instrução que verifica que dois números seguidos estão ordenados. Agorá estamos na altura de implementar os promenores. Vamos programar os três partes distintas da figura acima. Assume que existe um array com nome `n[1..N]`.

1) Para verificar se dois números seguidos estão ordenados correcto:

```
if n[i] < n[i+1] then ...
```

2) Trocar dois números também não é difícil. Por exemplo assim:

```
n[i] := n[i] + n[i+1];
n[i+1] := n[i] - n[i+1];
n[i] := n[i] - n[i+1];
```

Embora de funcionar (verifica isto!), este método é um pouco desajeitado. Normalmente trocar dois números fazemos com a ajuda de uma variável suplementar que vai guardar informação temporariamente:

```
temp := n[i];
n[i] := n[i+1];
n[i+1] := temp;
```

3) Para verificar que o array está ordenado completamente precisa-se uma variável que vai guardar a informação se houve uma mudança no último passo do array. Vamos declarar uma variável `change` que será `TRUE` se houve uma troca de dois números.

Com isto o procedimento inteiro é:

```
repeat
  change := FALSE;  (* até já ainda não houve trocas *)
  for i := 1 to N-1 do (* verifica todos os apres seguidos: *)
    if n[i]>n[i+1]  (* não ordenado? *)
      begin
          (* troca os dois números: *)
        temp := n[i];
        n[i] := n[i+1];
        n[i+1] := temp;
          (* faça um signal: *)
        change := TRUE;
      end;
until NOT change;
```

Velocidade / Eficiência

Na análise do nosso algoritmo é também possível dizer alguma coisa da eficiência do mesmo. Qual será o tempo de execução, no mínimo, no máximo, na média. Se assumimos que o tempo de execução é proporcional com o número de vezes o programa tem de ler um elemento do array podemos concludir o seguinte:

O algoritmo *Bubble sort* deve

- **no mínimo** fazer um passo do array, então ler 2*(N–1) números
- **no máximo** fazer N–1 passos do array, então ler $2*(N-1)^2$ números.
- a **média**: N^2–N (a média dos números acima)

com N o tamnho do array.
Em comparação, o outro algoritmo (Scan sort) deve

- **no mínimo** fazer N passos do array, mas cada vez pode ler menos elementos (a primeira vez tem de ler todos os N elementos, no segundo passo pode começar com elemento 2, depois com elemento 3, etc.), por isso deve ler N + (N–1) + (N–2) + (N–3) ... 1 = $N^2/2$ números

- o programa sempre deve ler o mesmo número de elementos, indepedente da distribução dos números. Por isso, **no máximo** também deve ler $N^2/2$ vezes.
- e a **média** fica assim $N^2/2$.

Qual algoritmo é mais eficiente? Para arrays com um grande número de elementos, por exemplo N=100, o segundo algoritmo seria cada vez mais eficiente (5000 vs. 9900). Também nota que no algoritmo *Bubble sort* temos de trocar mais números. Porquê? É fácil mostrar isto. Se um número está completamente no outro lado da lista temos de deslocar este número para o outro lado do array, isto torna o programa muito lento, com N−1 trocas. Com o outro algoritmo, o elemento seria trocada só uma vez. De facto, *Bubble sort* é provavelmente um dos piores algoritmos. O mais eficiente provavelmente é *Quick-sort* (fora do âmbito desta cadeira). A razão para mostrar o *Bubble sort* aqui é que este algoritmo é muito elegante e serve para mostrar ordenações e algoritmos em geral.

Quick Test

Para testar o seu conhecimento, sobre o que aprendeu nesta aula, clique aqui para um teste on-line.

Peter Stallinga. Universidade do Algarve, 6 Maio 2002

Palavras Chave

word	description	PASCAL	example
module	Sub-programa dentro dum outro programa para fazer uma tarefa específica.	`Procedure` `Function`	`Procedure Square(x:real); Function Square(x: real): real;`
declaração	reservar espace para uma variável e associar um nome a ela	`Var`	`Var x: real;`
constante	Define uma constante. Contrario a uma variável o seu valor não pode mudar	`Const`	`Const x = 3.0;`
atribuição	Operação para atribuir um valor a uma variável	`:=`	`x := 3.0;`
variável	Um lugar reservado na memória com o seu nome. Armazena informação dum tipo bem específico.		`X, a, abc455, ...`
condição	qualquer expressão que devolve um valor do tipo Verdadeiro/Falso. Normalmente utilizada para passar para uma outra parte do programa.		if `(x < 0)then`
expressão	Um cálculo que produz um certo valor		`3*a + 2*b + c`
instrução	Instrução ao computador para lhe dizer o que tem que fazer.		`a := 2*Sqrt(3.0);`
operador	Qualquer símbolo que opera sobre valores e produz outros valores.		`+, -, *, /`
operando	Aquilo sobre o qual trabalho um operador		`3.0`
operação	Um operador mais um operando		`3.0 + a`
programa	Conjunto completo de instruções com um fim bem determinado (*software*) Oposto ao computador onde o mesmo programa é executado (*hardware)*		Program Begin ... end.
Variáveis locais	variáveis que só podem ser utilizadas no interior dum procedimento ou função		
Variáveis globais	variáveis que podem ser utilizadas em qualquer lugar num programa		
Âmbito duma variável	O lugar do programa onde uma variável pode ser utilizada (local ou global)		
compilador	Um programa especial que traduz o código dos nossos programas e converte-os para linguagem de máquina.		
Linguagem de máquina	Um programa já traduzido que consiste de instruções para o processador (CPU)		
Saída(output)	resultados apresentados pelo computador (normalmente no monitor ou num ficheiro)		

Entrada(input)	Dados inseridos num programa (normalmente à partir do teclado ou ficheiro)		
argumento	parâmetro passado a uma função ou procedimento. Terá o mesmo tipo do parâmetro previamente definido.		
parâmetro	ver argumento		
tipo	tipo de variável, etc.		boolean, integer, word.
Passar ou saltar (branching)	decidir qual parte dum programa deve ser executada baseado numa condição		
memória	Lugar onde se armazena um programa e as suas variáveis		
Como saber mais: ver o dicionário de termos de computação FOLDOC na Internet.			

Dicas para programar correctamente

Seguem-se umas recomendações para programar correctamente:

- Utilizar procedimentos e funções cada vez que seja preciso repetir um mesmo grupo de instruções num mesmo programa. Faz com que um programa seja mais legível.

- Fazer estes procedimentos e funções independentes do resto do programa. O que quer dizer de não utilizar variáveis globais mas sim locais, em procedimentos ou funções.
- Não utilizar nomes para variáveis locais iguais aos nomes das variáveis globais.
- Utilizar nomes representativos para as variáveis, constantes, funções e procedimentos.
- Utilizar abundantes comentários para ilustrar o seu programa (* comentário *)
- Utilizar indentação. O programa fica mais legível e mais fácil para tirar erros.
- Não mude o valor das variáveis de controlo num ciclo "for". Se deseja fazer isto, então utilize um outro tipo de ciclo. ("*do-while*", ou "*repeat-until*")

Mini Teste 2: Computers

---

1. Quem desenhou o primeiro computador

- Bill Gates de Microsoft
- Blaise Pascal
- Charles Babbage
- IBM

---

2. O computador que a maioria de pessoas tem em casa é do tipo

- Supercomputer
- Mainframe
- Minicomputer
- Microcomputer
- Micro processor

---

3. Indique para cada peça de *hardware* a sua função

	input	output	armazenamento	processamento
Rato	○	○	○	○
Teclado	○	○	○	○
Memória	○	○	○	○
Monitor	○	○	○	○
Impressora	○	○	○	○
CPU	○	○	○	○

---

4. Para traduzir um programa de PASCAL numa linguagem que o computador percebe usamos

- Um compilador
- Um dicionário
- O Sistema Operativo
- O disco rígido

Mini Teste 3: Unidades de Informação / Memória

1. A unidade de informação mais pequena é

- ○ um bit
- ○ um byte
- ○ um nibble
- ○ um integer

2. A mais pequena unidade de informação ainda endereçavel é

- ○ o bit
- ○ o byte
- ○ o nibble
- ○ o integer

3. 1101 no sistema binário é (no sistema decimal) igual a

- ○ 1101
- ○ 15
- ○ 13
- ○ D

4. 2A no sistema hexadecimal é (no sistema decimal) igual a

- ○ 2A
- ○ 42
- ○ 20
- ○ 0

5. A maneira mais popular para representar texto é

- ○ binário
- ○ hexadecimal
- ○ decimal
- ○ ASCII

6. Quanto informação cabe aproximademente num standard disquete?

- ○ 1 byte: um caracter ASCII
- ○ 1 kilobyte (1 kB): um quarto duma página A4 em formato ASCII
- ○ 1 megabyte (1 MB): um livro em formato ASCII
- ○ 1 gigabyte (1 GB): uma pequena biblioteca em formato ASCII

Mini Test 4: Introdução a PASCAL

---

1. Quais dos seguintes *identifiers* são válidos

	válido	inválido	explicação
birthday8	○	○	
1shot	○	○	
hot?	○	○	
OLD_TIME	○	○	
down.to.earth	○	○	

---

2. Em PASCAL commentário escrevemos

- ○ depois "REM"
- ○ depois "//"
- ○ entre "{" e "}"
- ○ depois "comment"

Quick Test 5: Variables

1. A diferença entre `write` e `writeln` é

- `write` mostra no ecrãn, `writeln` é para a impressora.
- `write` é de C, `writeln` é de PASCAL.
- `writeln` põe o cursor na linha seguinte.
- `writeln` é para output formatado.

2. Para armazenar números inteiros usamos variáveis do tipo

- boolean
- byte, integer, ou word
- real ou double
- string

3. A gama de um integer é

- 0 .. 255
- 0 .. 65535
- -32768 .. 32767
- -2147483648 .. 2147483647

4. Declarar uma variável significa

- Reservar espaço na memória e associar um nome para aquele espaço.
- Atribuir um nome e um valor
- Inicilizar uma variável
- Mostrar o seu valor no ecrãn

5. Variáveis

- são inicializado com 0 no começo do programa
- são declaradas pelo seu primeiro uso
- devem ser atribuidos os valores na altura de declaração
- têm valores inprevisíveis no começo do programa

6. Para cálculos do tipo *floating point* com a mais alta precisão usamos variáveis do tipo

- boolean
- real
- longint
- extended

Teste rápido 6: Atribuição e Constantes

1. Se queremos atribuir o valor 8.3 à variável `r` fazêmo-lo:

- `r := 8.3;`
- `r = 8.3;`
- `r == 8.3;.`
- `8.3 -> r;`

2. Depois da atribuição da questão 1, qual das linhas seguintes de PASCAL produzirá

```
8.3000
```

- `writeln(8.3000);`
- `writeln(6*r,4);`
- `writeln('%6.4f', r);`
- `writeln(r:6:4);`

3. O que está errado no programa a seguir?

```
PROGRAM Error1;

VAR x: real;
CONST c = 1.0;

begin
  x*x := 2*c;
end.
```

- A constante não pode mudar o valor
- O lado esquerdo do `:=` pode apenas conter uma variável (única)
- A variável `x` não está definida
- O lado direito de `:=` não pode conter constantes

4. O que está errado no programa a seguir?

```
PROGRAM Error2;

VAR x: real;
CONST c = 2.0;

begin
  c := 2*x*c + 1;
end.
```

- A constante não pode mudar o valor
- O lado direito do `:=` pode apenas conter uma variável (única)
- A equação não tem solução
- As contantes têm que ser escritas com MAÍUSCULAS

5. Qual é a saída (*output*) do programa a seguir?

```
PROGRAM Variable;

VAR x: real;
CONST C = 1.0;

begin
  x := C + 1;
  x := 2;
  x := x + 3;
  writeln(x:4:1);
end.
```

- `7.0`
- `5.0`
- `3.0`
- `1.0`

Quick Test 7: Input and Math

1. A diferença entre `Read` e `ReadLn` é

- `ReadLn` é para ler ficheiros.
- `ReadLn` põe do lado o resto da linha
- `ReadLn` lê texto formatado
- `ReadLn` é para ler constantes, `Read` é para ler variáveis

2. Qual é o resultado de "`33 Mod 2`"?

- 1.5
- 16
- 16.5
- 1

3. Qual é o resultado de "`33 Div 2`"?

- 1.5
- 16
- 16.5
- 1

4. Qual é o resultado deste expressão?
"`1.0 + 2.0 * 3.0 - 6.0 / 2.0`"

- 4.0
- 1.5
- 14
- 9.0

Mini Teste 8: if ... then ... else

1. Qual comparação é inválido em PASCAL:

- `(a == b)`
- `(a <> b)`
- `(a = b)`
- `(a > b)`

2. O sintaxe para bifurcação simples é

- `if condition`
  `instruction;`
- `case condition`
  `instruction;`
- `if condition then`
  `instruction;`
- `case condition of`
  `instruction;`

3. Qual será o resultado deste programa?

```
PROGRAM IfThenElse;

Var a, b, c, d: integer;

begin
  a := 5; b:= 3; c := 99; d := 5;
  if a>6 then Write('A');
  if a>b then Write('B');
  if b=c then
    begin
      Write('C');
      Write('D');
    end;
  if b<>c then Write('E') else Write('F');
  if a>=c then Write('G') else Write('H');
  if a<=d then
    begin
      Write('I');
      Write('J');
    end;
end.
```

Verifique Não sei

4. Para declarar uma constante PI com valor 3.1415927 usamos

- Constant PI = 3.1415927;
- Const PI = 3.1415927;
- Const PI 3.1415927;
- Constant PI 3.1415927

Mini Teste 9: Álgebra Boolean / Case ... Of

1. Qual será o output do programa

```
PROGRAM Test;

Var a, b: real;
Const c = 10.0;

begin
  a := 9.0;  b := 2.0*c;
  if (a>0) XOR (b>0) then
    Write('Fixe!')
  else
    Write(' Uma pena');
end.
```

- `Fixe!`
- `Fixe! Uma pena`
- `Uma pena`
- o programa não tem *output*!

2. O que está mal no programa a seguir

```
PROGRAM Test;

Var a: real;
Const C = 2;

begin
  a := 3.0;
  Case a+1.0 Of
    1: Write('Fixe!');
    C: begin
         WriteLn('Cool!');
         WriteLn('Ingles');
       end;
    3: Write('Super!');
    else Write('Language?');
  end;
end.
```

- O `Case .. Of` não pode conter expressões (`a+1.0`)
- O `Case ... Of` não funciona com expressões do tipo real (`a+1.0`)
- Na estrutura `Case ... Of` não podemos usar constantes (`C`)
- Na estrutura `Case ... Of` não podemos usar `else`

3. O que é o resultado do cálculo `(43 AND 33)`?

Verifica Ajuda Nao sei

4.

`(3*4 + 12/6*i - j*2)`

é um exemplo de

- uma expressão
- uma condição
- uma atribuição
- uma operação

Mini Teste 10/11: Ciclos

1. No qual tipo de ciclos a instrução é executada no mínimo uma vez?

- For
- While-Do
- Repeat-Until
- não existe

2. Quero escrever um programa que vai pedir ao utilizador um número e o programa deve mostrar todos os números primos até este número. Melhor seria usar um ciclo do tipo

- For
- While-Do
- Repeat-Until
- Outra estrutura

3. Quais são as duas regras do *nesting* dos ciclos?

1:
2:

Ajuda Resposta corecta

4. A diferença entre os ciclos While-Do e Repeat-Until é

- While-Do é para ciclos com variáveis inteiros, Repeat-Until é para ciclos com variáveis do tipo floating point.
- Repeat-Until é para ciclos com variáveis inteiros, While-Do é para ciclos com variáveis do tipo floating point.
- No ciclo Repeat-Until a condição é verificada no início, no ciclo While -Do no fim.
- No ciclo While-Do a condição é verificada no início, no ciclo Repeat-Until no fim.

5. O que está mal no código a seguir?

```
x := 0.0;
while (x<10.0) do
  begin
    y := x*x;
    z := x*y;
    writeln('The square of ',x:0:2, ' is
', y:0:2);
    writeln('The cube of ',x:0:2, ' is ',
z:0:2);
  end;
```

- O ciclo nunca vai acabar
- Temos de usar um ciclo do tipo Repeat-Until.
- Temos de usar um ciclo do tipo For.
- A condição não pode conter variáveis do tipo floating point.

6. Queremos escrever um programa que vai pedir ao utilizador de escolher um tipo de cálculo ou sair do programa (1=addicionar, 2=subtrair, 0=sair). O programa deve continuar fazer isto até sempre (excepto, claremente, quando o utilizador escolhe 0). Neste caso, o melhor ciclo seria

- For
- While-Do
- Repeat-Until
- Outra estrutura

Mini Teste 12,13: Programação Modular

1. Quais são os tipos de módulos de PASCAL?

1:
2:

Resposta corecta

2. Quais são as vantagens de usar módulos?

1:
2:

Resposta corecta

3. Que será o resultado do programa seguinte?

```
PROGRAM Procs;
Var x: real;

PROCEDURE WriteFormatted(r: real; n:
integer);
begin
  WriteLn(r:0:n);
end;

begin
  x := 10.0;
end.
```

- 10.0
- r:0:n
- 0
- Este programa não gera *output*. Esquecemos de CHAMAR o procedimento!

4. Qual é a diferença entre um *Procedure* e uma *Function*?

- Uma Function aceita parâmetros (*input*), um Procedure não
- Uma Function retorna um valor (*output*), um Procedure não
- Um Procedure aceita parâmetros (*input*), uma Function não
- Um Procedure retorna um valor (*output*), uma Function não

Mini Teste 14: Funções Matemáticas

1. Que será o resultado da seguinta chamada da função?

```
Round(3.53)
```

- 3
- 3.0
- 4
- 4.0

2. Como implementar a função ArcCos(x)?

for x>0:

for x=0:

for x<0:

Correct Answer

3. Que será o resultado da seguinta chamada da função?

```
Random(100)
```

- 100
- 0, porque esquecemos de chamar Randomize;
- Imprevisível. Um número entre 0 e 99 (inclusive)
- Imprevisível. Um número entre 1 e 100 (inclusive)

4. Que será o valor de x depois a seguinta instrução?

```
x := Sqr(Sqr(Sqr(2.0)));
```

- Esta construção não é permitida!
- 4.0
- 16.0
- 256.0

Mini Teste 15: Âmbito das variáveis, passagem por valor e passagem por referência

1. Qual o âmbito de cada objecto no programa a seguir?

```
PROGRAM VariableTypes;
Var a: real;

PROCEDURE Proc1(b: real);
Var c: real;
Const d = 10.0;
begin
  c := b+d;
  Writeln(c);
end;


Function Proc2(Var e: real): real;
Const f = 20.0;
begin
  Proc2 := e+f;
end;

Var g: real;

begin
  a := 10.0;
  Proc1(a);
end.
```

	local	global	parâmetro	nenhum
a	○	○	○	○
b	○	○	○	○
c	○	○	○	○
d	○	○	○	○
e	○	○	○	○
f	○	○	○	○
g	○	○	○	○

2. Considere o programa abaixo

```
PROGRAM QuickTest15
Var x: integer;
PROCEDURE Show(Var a: integer);
begin
  write(a,' ');
  a := a + 1;
end;

begin
  x := 0;
  Write(x,' ');
  Show(x);
  Write(x);
end.
```

O procedimento usa a técnica de

- ○ Passagem por valor
- ○ Passagem por referência

e, por isso o resultado será

[ ]

Correct Answer

3. Qual será o *output* do programa a seguir?

```
PROGRAM DoubleNames;
Var x: integer;

PROCEDURE Show;
Var x: integer;
begin
  x := 1;
  x := x*x;
  Write(x,' ');
end;

begin
  x := 0;
  Show;
  Write(x);
end.
```

- ○ Não é permitido usar o mesmo nome para uma variável duas vezes!
- ○ 0 0
- ○ 1 0
- ○ 0 1
- ○ 1 1

Mini Teste 16: Programação recursiva

Considera o programa a seguir:

```
PROGRAM CalculateN;
Var a: real;

Function XfuncN(x: real; n: integer): real;
Var c: real;
begin
  c := 0.0;
  if n=0 then
    XfuncN := 1.0
  else
    XfuncN := x*XfuncN(x, n-1);
end;

begin
  WriteLn(XfuncN(3.0, 3):0:1);
end.
```

1. Qual será o *output* do programa?

Verifica Não sei

2. Quantas cópias da variável local c existem no máximo?

Verifica Não sei

Mini Teste 16 extra: Programação recursiva

Considera o programa a seguir:

```
PROGRAM CalculateN;
Var a: real;

Function XfuncN(x: real; n: integer): real;
Var c: real;
begin
  c := 0.0;
  if n=0 then
    XfuncN := 1.0
  else
    XfuncN := Exp(n*Ln(x)) + XfuncN(x, n-1);
      (* lembra: a^b = Exp(b*Ln(a)) *)
end;

begin
  WriteLn(XfuncN(2.0, 2):0:1);
end.
```

1. Qual será o *output* do programa?

Verifica Não sei

2. Quantas cópias da variável local c existem no máximo?

Verifica Não sei

Mini Teste 17, 18: Arrays e Records

1. Qual a diferença entre um array e um record?

- Um array é só para armazenar coisas contáveis, com records é possível armazenar tudo.
- Um record é só para armazenar coisas contáveis, com arrays é possível armazenar tudo.
- Arrays são para combinar variáveis de tipos diferentes, records são para armazenar variáveis do mesmo tipo.
- Records são para combinar variáveis de tipos diferentes, arrays são para armazenar variáveis do mesmo tipo.

```
FUNCTION Maximum(a, b: real): real;
var max: real;
begin
  if a>b then max := a else max := b;
 ......
end;
```

2. Agora, como deixar a função retornar o valor de max à instrução que chamou este função?

- Nada é automaticamente.
- `Maximum := max;`
- `return max;`
- Esta função não gera output e por isso não vai retornar nada!

```
Var a: array[1..10] of
  record
    x: record
         z: array[1..10] of
real;
         i: array[1..3] of
integer;
       end;
    y: record;
         r: real;
         p: double;
       end;
  end;
```

3. Como atribuir um valor de 0 ao (primeiro) `i` do array?

Verifica Não sei

4. Queremos construir um base de dados para armazenar a informação de 1000 alunos. Melhor seria fazer isto com uma variável

-
```
Var a: record
     number: integer;
     name: string;
     year: integer;
 end;
```
-
```
Var a: array[1..1000] of
     record
     name: string;
     year: integer;
 end;
```
-
```
Var a: record
     number: array[1..1000] of integer;
     name: array[1..1000] of string;
     year: array[1..1000] of integer;
end;
```
-
```
Var a: array[1..1000] of
     record
       name: array[1..1000] of string;
       year: array[1..1000] of integer;
     end;
```

Mini Teste 19: Type

1. O que é que o `Type` faz?

- Escreve texto no ecrã.
- Define um novo tipo da variável.
- Faz combinações de arrays e records.
- Declara variáveis dos tipos mistos.

```
Type b = real;
 ......
b := 3.1;
```

2. Porquê o código acima não funciona?

- Temos de usar '`Type b: real`' em vez.
- `real` já está definido.
- A sintaxe está mal; em vez temos de usar '`typedef`'.
- `Type` só faz uma especificação de um tipo de variável para declarar depois.

```
Type a = array[1..10] of
  record
    ri: record
         x: array[1..10] of real;
         y: array[0..3] of integer;
       end;
    rd: record;
         v: real;
         w: double;
       end;
  end;
Var b: a;
```

3. Como atribuir um valor de 0 ao ('primeiro') `y` do programa?

Verifica Não sei

4. Qual será o output do seguinte código?

```
Type floats = array[1..10] of real;

PROCEDURE WriteIt(r: floats);
begin
  WriteLn(r[1]);
end;

Var x: array[1..10] of integer;

begin
  x[1] := 3;
  WriteIt(x);
end.
```

- Inprevisível. Esquecemos de inicilizar o array `r`!
- `3.0`
- Nada; fizemos uma mistura de tipos em chamar o procedure.
- `3`

Mini Teste 20: Apontadores

1. Como declarar um apontador a um word?

- `Var a: ^word;`
- `Var a: word;`
- `Var a: @word;`
- `Var a: word^;`

2. Como atribuir o endereço da variável `x` ao apontador `p`?

- Depende do tipo de `x`.
- `p := ^x;`
- `p := @x;`
- `p := x^;`

```
Var b: array[1..20] of ^integer;
```

3. Como atribuir o valor 0 ao primeiro integer de `b`?

Verifica Não sei

4. O que significa `p := NIL`?

- 0.
- O conteúdo do endereço `p` será 0.
- O apontador `p` apontará nada.
- O apontador `p` apontará endereço 0.

Aula Prática 1-a

Sumário

Noções sobre o Windows

O Windows é um sistema operativo. Um sistema operativo é um conjunto de programas que gerem os recursos do computador, permitindo-nos trabalhar com ele.

O Windows é um sistema operativo multi-utilizador e multi-tarefa (significa que várias pessoas podem usar o mesmo computador ao mesmo tempo e correndo
programas diferentes).

O Windows tem um mecanismo de segurança que impede os utilizadores normais de danificarem ficheiros que são essenciais para o bom funcionamento do sistema.
Portanto, não tenham medo de experimentar. O computador "não morde", e o pior que pode acontecer é perderem os vossos ficheiros pessoais.

Cada utilizador tem um nome (login name) e uma password, que o identifica no sistema. Tem também uma área de trabalho só dele, aqui encontram-se todos os
ficheiros que lhe pertencem.

O Windows não é sensível às maiúsculas e minúsculas.

Para começar uma sessão em Windows temos de fazer login:

login - identifica os utilizadores que entram no sistema.

> Login: nome do utilizador
> Password: **********

Nota: É importante não esquecer a password.

Comandos Básicos do Windows

A partir de abrir um Command-Shell ("cmd" em "Start:Run", ou "MS-DOS Prompt") podemos entrar os nossos comandos. Grande parte dos comandos são abreviaturas de palavras inglesas.

Comandos de uso geral

exit - para terminar a sessão de trabalho no Command Shell

> exit

help - para pedir ajuda sobre um comando.

> help nome_comando

Comandos sobre directorias

As directorias servem para agrupar os ficheiros por temas, permitindo uma melhor organização da informação.

dir - para ver o conteúdo das directorias. vai também mostrar o tamanho e o data da ultima mudança

> dir nome_directoria - mostra os ficheiros e outras directorias
> no interior da directoria indicada.
> dir - mostra os ficheiros da directoria corrente

cd - para mover entre directorias (cd de change directory).

> cd \ - vai para a raíz.

> cd .. - sobe um nível, vai para a directoria a cima.

> cd . - a própria directoria.

> cd nome_directoria - vai para a directoria indicada.

> cd - para indicar a directoria corrente

mkdir - para criar novas directorias (**mkdir** de **m**a**k**e **dir**ectory).

> mkdir nome_directoria

rmdir - para apagar directorias (**rmdir** de **r**e**m**ove **dir**ectory).

> rmdir nome_directoria - apaga apenas se a directoria
> estiver vazia.

nome_directoria: Caminho desde a raíz (\) até á directoria desejada. Para cada novo nível na árvore coloca-se uma nova \.

Comandos sobre ficheiros

copy - para copiar ficheiros

> copy nome_directoria_origem\nome_ficheiros nome_directoria_destino

move - para mover ficheiros

> move nome_dirertoria_origem\nome_ficheiros nome_direrctoria_destino\nome_ficheiros

del - para apagar ficheiros (del de delete).

> del nome_directoria\nome_ficheiros - apaga os ficheiros da directoria indicada.

> del nome_ficheiros - apaga os ficheiros da directoria corrente.

type - para ver o conteúdo de ficheiros.

> type nome_directoria\nome_ficheiros

> type ficheiro | more

Mostra no écran o contéudo do(s) ficheiro(s).

Comandos essenciais do more:
Space - ir para a próxima página
b - ir para a página anterior
q - quit (sair)

Metacaracteres (* e ?)

Os metacaracteres podem ser utilizados com qualquer um dos comandos de ficheiros, e facilitam bastante quando se pretende fazer a mesma operação sobre
ficheiros que têm algo em comum no seu nome.

O * representa uma cadeia de caracteres.

O ? representa um caracter.

Permissões de Ficheiros

O Windows tem um mecanismo que nos permite proteger os nossos ficheiros dos outros utilizadores. A esse mecanismo chama-se attributes de ficheiros.

Existem 4 attributes de ficheiros
A: Archive (ficheiro)
R: Read-Only (só ler)
H: Hidden (invisível)
S: System file (do sistema operativo. Não mexe!)

para mudar

> attrib {+,-}{A,R,H,S} nome_ficheiros

por exemplo:

> attrib -r prim.pas

Para ver as especificações existentes:

> attrib

O Ambiente de Trabalho

Para criar um programa numa linguagem de programação qualquer, existem quatro passos que têm sempre que acontecer:

1. Pensar no problema, de preferência com papel e lápis.
2. Escrever o programa, para isso é necessário um editor de texto.
3. Verificar se o código que se escreveu não tem erros, para isso é necessário um compilador da linguagem de programação usada.
4. Quando o programa estiver livre de erros, criar o ficheiro executável e correr o programa.

Em Windows, como se podem realizar os três últimos passos?

2 - Editar ficheiros

Existem alguns editores de texto em Windows. O notepad é o mais importante.
Podes usar qualquer um deles, e até podes gravar os ficheiros numa disquete e continuar a trabalhar com eles em casa.
Para escrever os programas em Pascal é melhor usar o IDE (integrated development environment) de Turbo Pascal. Use

TP <nome_ficheiro>

na pasta onde pretende escrever o programa.

3 e 4 - Compilar e executar programas

Para compilar/executar um programa devem chamar o compilador gcc na linha de comandos:

Compilar e tornar executável:

> tpc nome_ficheiro.pas

Correr o programa:

> nome_ficheiro

Aula Prática 1-b

Sumário

Exemplos práticos dos comandos básicos do Windows

0. Login e abre uma janela "DOS Command"

em caso de erro de teclado: chcp 850

1. Comandos de uso geral

Comando **help**

```
> help dir
```

2. Comandos

Comando para mudar de disco

```
>  y:
```
(muda para o disco virtual que fica em Europa). O aluno sempre deve trabalhar em y:

Comando **cd**

```
> cd
```

Comando **dir**

```
> dir
```

Comando **cd**

```
> cd
```

Comando **mkdir**

```
> mkdir prog1
> cd prog1
```

Comando **edit** (em caso de erro de teclado: chcp 850)

```
> edit p1.pas
```

escreve lá texto, guarda o ficheiro e sai

Comando **type**

```
> dir
```

```
> type p1.pas
```

Comando **more**

```
> type p1.pas | more
```

Comando **copy**

```
> copy p1.pas p2.pas
> copy p1.pas p1a.pas
> dir
```

Comando **move**

```
> move p1.pas x.pas
> dir
```

Permissões de ficheiros. Comando **attrib**

```
> attrib
> attrib +r p1.pas
> attrib
```

Comando **del**

```
> dir
> del p*.pas
> dir
> attrib -r p1.pas
> del p*.pas
> dir
> del x.pas
> del *.*
> dir
```

Comando **rmdir**

```
> cd ..
> dir
> rmdir prog1
> dir
```

Ambiente de Trabalho

#Abrir o editor de texto.
#Escrever um texto livre.
#Guardar num ficheiro.
#Abrir um novo ficheiro.
#Escrever o programa Hello World.

```
PROGRAM HelloWorld;

begin
  writeln('Hello World');
end.
```

#Guardar num ficheiro hello.pas.

```
#Compilar e correr via linha de comandos.
```

Aula Prática 1-c

Sumário

A Internet

A Internet é uma rede mundial de computadores que estão espalhados pelo mundo inteiro. Cada computador ligado à Internet utiliza software próprio para poder disponibilizar e/ou aceder a informação. A Internet é um meio através do qual se pode aceder a informação disponível em documentos ou ficheiros que estão contidos noutros computadores. A Internet pode ser comparada com uma infra-estrutura para suportar uma espécie de biblioteca gigantesca.

Os computadores ligados à Internet podem aceder aos seguintes serviços:

Correio electrónico (e-mail).
Permite receber e enviar mensagens para outras pessoas em qualquer ponto do mundo.

Telnet ou login remoto.
Permite que utilizes o teu computador para te ligares a um computador remoto (possivelmente noutro país) e usá-lo como se estivesses lá.

FTP (File Transfer Protocol).
Permite que o teu computador possa transferir ficheiros de/para outros computadores.

World Wide Web (WWW ou "Web").
Quando te ligas à Internet utilizando o Netscape ou outro browser qualquer (ex: Windows Explorer), estás a ver documentos na WWW. A WWW está baseada na noção de Hipertexto. Este é um sistema que tem links, uma espécie de apontadores para outros documentos na Web. Cada documento é identificado por um URL (Uniform Resource Locator) que não é mais do que um endereço único para um documento da Web.

Exemplos de URLs:

http://www.cnn.com é o endereço do canal Americano CNN.
http://www.publico.pt é o endereço do jornal Publico.
http://www.ualg.pt é o endereço da Universidade do Algarve.
http://diana.uceh.ualg.pt/IC/ é o endereço da cadeira de Introdução a Computação.
http://w3.ualg.pt/~pjotr/ic/index.html é o "*mirror*" do endereço em cima
http://www.sosmath.com/index.html para ajudar em matemática
http://www.physicscentral.com/lou/index.html física

Sempre que quiseres encontrar coisas na Internet, aconselho a utilizares o Altavista, cujo URL é http://www.altavista.com ou Google (URL: http://www.google.com. O Google permite que escrevas palavras e devolve-te um conjunto de URLs que têm informação relacionada com essas palavras. Experimenta.

Os documentos na Web estão feitos utilizando uma linguagem chamada HTML (HyperText Markup Language). O HTML tem um conjunto de comandos (tags) que permitem formatar texto, incluir imagens, e especificar links para outros documentos. Se quiseres ver o HTML que gerou o documento que estás a ver neste preciso momento, vai à opção View e dentro dessa opção escolhe Page Source.

Aula prática 2a

Sumário

O compilador

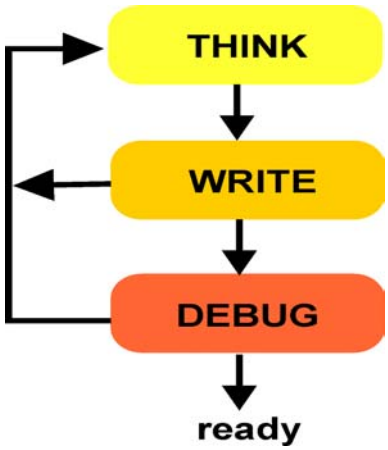
Nas nossas aulas vamos utilizar o compilador de linha de comando **tpc**. O **tpc** de Borland é um dos compiladores de PASCAL mais vendidos. Já não é o mais avançado (agora existe por exemplo Delphi ou Kylix) mas é muito bom para os nossos programas. Existem também compiladores de domínio livre (*public domain*), por exemplo o compilador de FreePascal (http://www.freepascal.org) . O que significa ser software de domínio livre? Significa que temos liberdade para:

1. Executar o software, qualquer que seja o nosso propósito
2. Estudar o modo como o software funciona e adaptá-lo às nossas necessidades
3. Distribuir cópias do software
4. Melhorar o software e distribuir esses melhoramentos para benefício da comunidade

O **tpc** suporta os standards modernos da linguagem PASCAL (como p/ex. o ANSI PASCAL) ao mesmo tempo que mantém a compatibilidade com os compiladores e estilos mais antigos.

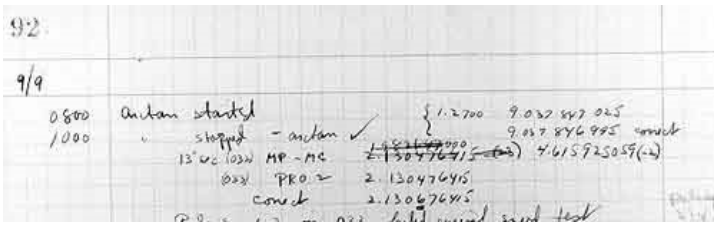
Software engineering

Crear programas consiste em vários etapas



1. **Pensar**. Estudar o problema.
2. **Escrever** um programa
3. Resolver os erros do programa. **Debugging**
4. Analisar o resultado
5. Em caso necesário, **Voltar** ao passo 3, 2 ou 1.

1. **Pensar**. Desenho um programa com papel e lápis. Ainda não toca o computador! Busca informação sobre o assunto. Chega a um algoritmo.



2. Escrever o programa. Use qualquer **editor**. Por exemplo notepad, edit, ou turbo.

3. **Debugging**. ("bug" = bicho, então debugging é debichomento). Eliminar erros do programa.
O compilador vai gerar "***compile-time errors***", por exemplo

```
type-mismatch error in line 34
```

Se o compilador nao encontre mais erros, vai gerar um ficheiro executável (`*.exe`) com o código binário.
Durante o correr do programa podem acontecer "***run-time errors***", por exemplo

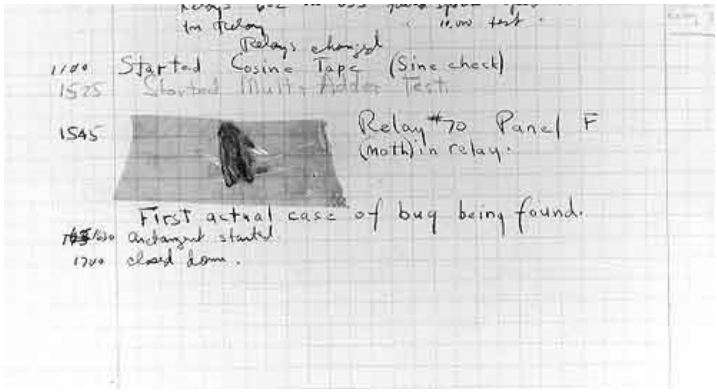
```
division by zero error
```

Ambas os tipos de erros devem ser eliminados!

4. Mesmo se o nosso programa passa o compilador sem gerar erros e corre sem gerar erros o resultado do programa pode estár mal. por exemplo, o "output" do programa pode ser

```
O raiz de 9 é 4
```

Temos de voltar ao passo 3, 2 ou 1.



The First Computer Bug

Grace Murray Hopper, working in a temporary World War I building at Harvard University on the Mark II computer, found the first computer bug beaten to death in the jaws of a relay. She glued it into the logbook of the computer and thereafter when the machine stops (frequently) they tell Howard Aiken that they are "debugging" the computer. The very first bug still exists in the National Museum of American History of the Smithsonian Institution. Edison had used the word bug and the concept of debugging previously but this was probably the first verification that the concept applied to computers.
(copied from http://www./firstcomputerbug.html)

Um exemplo passo a passo

Vamos então seguir, em quatro passos, um pequeno exemplo para criação do nosso primeiro programa.

1. Definir um espaço para o programa
2. Criar o programa
3. Compilar o programa
4. Executar o programa

Definir um espaço para o programa

É mais fácil mantermos o nosso espaço em disco organizado se criarmos uma directoria para cada programa no qual estejamos a trabalhar (excepção feita para programas ou projectos cujo código fonte esteja dividido em vários ficheiros). Neste caso vamos criar uma directoria chamada **prog1** para guardar o nosso primeiro programa.

mkdir prog1
cd prog1

Criar o programa

Um programa começa por ser um ficheiro de texto. Utiliza um editor de texto do teu agrado (p/ex. o **edit**, ou o **turbo (tp)**) para escrever esse texto a que é comum chamar-se código fonte. Consoante a escolha podes dar um dos seguintes comandos:

edit prog1.pas ou **tp prog1.pas**

e escrever o programa que se segue:

```
PROGRAM MyFirstProgram;

begin
  writeln('Hello World');
end.
```

Compilar o programa

O compilador pega no código fonte e converte-o num programa executável. Para compilares o teu código fonte usando o compilador **tpc** executa o comando:

tpc prog1.pas

O parâmetro **-o** diz ao compilador que o ficheiro executável deverá chamar-se prog1, enquanto o prog1.c no final do comando indica em que ficheiro se encontra o código fonte.

Executar o programa

Para correr o programa faz

prog1 ou **prog1.exe**

e a mensagem "Hello World" vai aparecer no ecrã.

Programas introdutórios em linguagem PASCAL

Agora podes fazer os programas que se seguem.

Aula prática 2 (continuação)

Agora que já fizeste o teu primeiro programa, utiliza um editor da tua preferência e um compilador para escrever e compilar os programas apresentados em baixo. Lembra-te de que cada programa fica num ficheiro separado, cada qual com o seu nome. Compara os programas, executa-os, observa e comenta os resultados.

Programa 1

Um programa com uma instrução:

```
PROGRAM SingleLine;

begin
  writeln('Este e o meu primeiro programa');
end.
```

Programa 2

Um programa com mais instruções:

```
PROGRAM MultiLine;

begin
  write('Este e ');
  write('o meu ');
  write('primeiro ');
  writeln('programa');
end.
```

Programa 3

Um programa mal estruturado, mas correrá sem erros. Experimente:

```
PROGRAM BadLayout; begin write('Este e '); write('o meu '); write('primeiro ');
writeln('programa'); end.
```

Programa 4

Encontre os erros no programa seguinte (tem 3). Use o compilador para eliminar os erros:

```
PROGRAM 3Erros;

begin
  write('Este e ');
  write('o meu ')
  write('primeiro ');
  writeln('programa');
end;
```

em caso de dúvidas, consulte os apontamentos da aula 4.

Programa 5

O seguinte program soma dois números inteiros, mas está mal estruturado. Faz um melhoramento do mesmo (indentação, rebaptizar os indentifiers, introduz comentário, etc).

```
PROGRAM program1;
var x10, a35, the_result_of_the_calculation_of_summing_two_variables; begin
write('Introduza um número: '); readln(x10');
write('Introduza um número: '); readln(a35);
the_result_of_the_calculation_of_summing_two_variables := x10 + a35;
writeln('A soma de ',x10,' e ',a35,' e
',the_result_of_the_calculation_of_summing_two_variables); end.
```

Programa 6

Elimina os erros do programa seguinte (*compile-time* erros e *run-time* erros!)

```
PROGRAM MyNameAndAge;

Var nome: string;
    start, end, idade: integer

begin
  write('O seu nome: ');
  readln(nome);
  write('o ano do seu nascimento: ')
  readln(start)
  write('O ano actual: ');
  readln(end);
  writelnn('Exmo. ',(end-start)/0,'  voce tem agora ',nome,' anos');
end.
```

Soluções

Introdução a Computação

Soluções da aula prática 2

Programa 4

Os erros

```
PROGRAM TresErros;              (* Indentifier não pode começar com um dígito *)

begin
  write('Este e ');
  write('o meu ');              (* faltava ; *)
  write('primeiro ');
  writeln('programa');
end.                            (* A última end do ficheiro é finalizada com . *)
```

Programa 5

```
PROGRAM program1;

var x10, a35, soma: real;

begin
  write('Introduza um número: ');
  readln(x10);
  write('Introduza um número: ');
  readln(a35);
  soma := x10 + a35;
  writeln('A soma de ',x10,' e ',a35,' e ',soma);
end.
```

Programa 6

```
PROGRAM MyNameAndAge;

Var nome: string;
    start, end, idade: integer              (* compile-time error: faltava ; *)

begin
  write('O seu nome: ');
  readln(nome);
  write('o ano do seu nascimento: ')        (* compile-time error: faltava ; *)
  readln(start)                             (* compile-time error: faltava ; *)
  write('O ano actual: ');
  readln(end);
  writelnn('Exmo. ',nome,'  voce tem agora ',(end-start),' anos');
      (* run-time error: division by zero *)
```

```
      (* erro no desenho do programa: trocar "(end-start)" com "nome" *)
end.
```

Aula prática 3

Sumário

- tamanho das variáveis
- inicializar
- texto formatado

Aviso geral

Save your programs! (guarda os seus programas).
Cada aluno tem uma conta na rede da universidade. Depois de fazer o *login*, o computador está ligado à rede e tem um disco rigido virtual (Y:) onde o aluno pode guardar os seus ficheiros. Ninguem pode mexer com os seus programas em Y:. Para trazer o trabalho a casa, pode também gravar os programas numa disquete (A:). Sem gravar os programas com regularidade, o aluno corre o risco de perder informação.

Jesus and Satan have an argument as to who is the better programmer. This goes on for a few hours until they agree to hold a contest with God as the judge. They set themselves before their computers and begin. They type furiously for several hours, lines of code streaming up the screen. Seconds before the end of the competition a bolt of lightning strikes, taking out the electricity. Moments later the power is restored and God announces that the contest is over. He asks Satan to show what he has come up with. Satan is visibly upset and cries I have nothing! I lost it all when the power went out. Very well then says God let us see if Jesus fared any better. Jesus enters a command and the screen comes to life in vivid display the voices of an angelic choir pour forth from the speakers. Satan is astonished. He stutters But how! I lost everything yet his program is intact! How did he do it?! God chuckles Jesus saves!

Tamanho das variáveis

Em PASCAL (e C também) existe uma instrução (`SizeOf`) que vai nos dizer o espaço uma variável ocupa na memória. Hoje vamos usar esta instrução para determinar os tamanhos de todos os variáveis que nos sabemos (veja aula 5).
A instrução `SizeOf` é um exemplo duma função. Ainde não sabemos o que é uma função, mas vamos dar aqui o uso geral de SizeOf:

```
SizeOf(nome)
```

retorne o tamanho da variável `nome`. Para mostrar o resultado, vamos usar `WriteLn` (aula 5).

```
WriteLn(SizeOf(nome));
```

1. Escreve um programa que

- declare variáveis de todos os tipos de variáveis nos sabemos.
- mostra o tamanho de cada variável no ecrãn, por exemplo:

```
integer: 2 bytes
real: 6 bytes
```

Inicialização

2. Muda o programa do trabalho 1 acima de forma que o programa vai mostrar o valor da cada variável, em vez de mostrar o seu tamanho. As variáveis têm todas valors igual a 0?
Agora faz uma inicialização de cada variável e experimenta outra vez.
Nunca assuma que o valor duma variável é 0.

Texto Formatado

O resultado do writeln(r) do programa acima fica feio. Por exemplo:

```
real: 6.93896007838148E003
```

Imagina receber a informação do saldo da sua conta no banco nesta forma!
Melhor seria

```
real: 6938.96
```

A instrução `WriteLn` pode gerar texto com formato predefinido. veja aula teórica 6.

3. Escreve um programa com variavéis `r`, `s`, e `i`, de tipo real, string e longint. Atribue valores: `r =` 63.9, `s =` o seu nome, e `i =` o numero do aluno. O programa deve mostrar esses tres variáveis no formato `s`: 30 lugares, `i`: 10 lugares, `r`: 10 lugares total e dois casos decimais depois o ponto, por exemplo:

```
Peter Stallinga     999999      63.90
```

Constantes

4. Escreve um programa que use variaveis do tipo *floating point* e uma constante igual a *e* (2.7....). O programa deve mostrar o valor de cada variável com 4 casas decimais.

Debugging

5. Elimina os erros do programa seguinte (compile-time erros, run-time erros e erros no desenho do programa). Não se preocupa com as instruções que não conhece; só elimina os erros que o compilador vai nos dizer:

```
FullOfErrors;

begin
  real: x,y,z;
  CONST PI = 1.6857;

  x := 3;
```

```
  y = 2*pi;
  1 := z;
  writelnn(x:0:3);
end.
```

---

soluções

Introdução a Computação

Soluções da aula prática 3

1.

```
PROGRAM TamanhoDasVariaveis;

Var bl: booelan;
    by: byte;
    i: integer;
    w: word;
    l: longint;
    r: real;
    d: double;
    e: extended;
    c: char;
    s: string;

begin
  WriteLn('boolean: ', SizeOf(bl), ' bytes');
  WriteLn('byte: ', SizeOf(bt), ' bytes');
  WriteLn('integer: ', SizeOf(i), ' bytes');
  WriteLn('word: ', SizeOf(w), ' bytes');
  WriteLn('longint: ', SizeOf(l), ' bytes');
  WriteLn('real: ', SizeOf(r), ' bytes');
  WriteLn('double: ', SizeOf(d), ' bytes');
  WriteLn('extended: ', SizeOf(e), ' bytes');
  WriteLn('char: ', SizeOf(c), ' bytes');
  WriteLn('string: ', SizeOf(s), ' bytes');
end.
```

resultado do programa:

```
boolean: 1 bytes
byte: 1 bytes
integer: 2 bytes
word: 2 bytes
longint: 4 bytes
real: 6 bytes
double: 8 bytes
extended: 10 bytes
char: 1 byte
string: 256 bytes
```

2a.

```
PROGRAM TamanhoDasVariaveis;

Var bl: booelan;
    by: byte;
    i: integer;
    w: word;
    l: longint;
    r: real;
    d: double;
    e: extended;
    c: char;
    s: string;

begin
  WriteLn('boolean: ', bl);
  WriteLn('byte: ', bt);
  WriteLn('integer: ',i);
  WriteLn('word: ', w);
  WriteLn('longint: ', l);
  WriteLn('real: ', r);
  WriteLn('double: ', d);
  WriteLn('extended: ', e);
  WriteLn('char: ', c);
  WriteLn('string: ', s);
end.
```

resultado do programa poderia ser:

```
boolean: TRUE
byte: 34
integer: -6589
word: 3256
longint: 0
real: 3.45927643E32
double: 5.111878091E103
extended: 9.9934512987E-204
char: %
string: BjHGgy^tTr4RFgjOI()0iOolooiGvTR4@#4Erd
BhHHGT&Yg%%90KJNhjJhHUY(7/&77676767767&77tyg6t
NBHGuyFIt675&%7687/8788hKJjho9YuhPUy
```

2b.

```
PROGRAM TamanhoDasVariaveis;

Var bl: booelan;
    by: byte;
    i: integer;
    w: word;
    l: longint;
    r: real;
    d: double;
    e: extended;
    c: char;
    s: string;

begin
  bl := FALSE;
  bt := 0;
  i := 0;
  w := 0;
  l := 0;
  r := 0.0;
  d := 0.0;
  e := 0.0;
  c := 'a';
  s := 'ola';
  WriteLn('boolean: ', bl);
  WriteLn('byte: ', bt);
  WriteLn('integer: ',i);
  WriteLn('word: ', w);
  WriteLn('longint: ', l);
  WriteLn('real: ', r);
  WriteLn('double: ', d);
  WriteLn('extended: ', e);
  WriteLn('char: ', c);
  WriteLn('string: ', s);
end.
```

resultado do programa será:

```
boolean: FALSE
byte: 0
integer: 0
word: 0
longint: 0
real: 0
double: 0
extended: 0
char: a
string: ola
```

3.

```
PROGRAM WriteFormatado;

Var r: real;
    i: longint;
    s: string;

begin
  r := 63.9;
  s := 'Peter Stallinga';
  i := 99999;
  WriteLn(s:30, i:10, r:10:2);
end.
```

resultado do programa:

```
               Peter Stallinga     99999      63.9
```

4.

```
PROGRAM Constante;

Var r1, r2: real;
Const E = 2.7;

begin
  r1 := 63.9*E;
  r2 := 1.0;
  WriteLn(r1:0:4);
  WriteLn(r2:0:4);
end.
```

5.

```
PROGRAM FullOfErrors;     (* faltava 'PROGRAM' *)

real: x,y,z;              (* o lugar de declarar as variaveis e antes 'begin' *)
                          (* a delaracao e: VAR nome: tipo; *)
CONST PI = 3.1415;        (* Pi nao e 1.6857 *)


begin
  x := 3.0;               (* lado esquerde: tipo real, lado direito deve ser igual *)
  y = 2.0*PI;             (* constantes com maiusculas *)
  z := 1;                 (* lado esquerdo: uma variavel, lado direito: expressao *)
  writeln(x:0:3);         (* writeln com um n *)
end.
```

Introdução a Computação

Aula prática 4

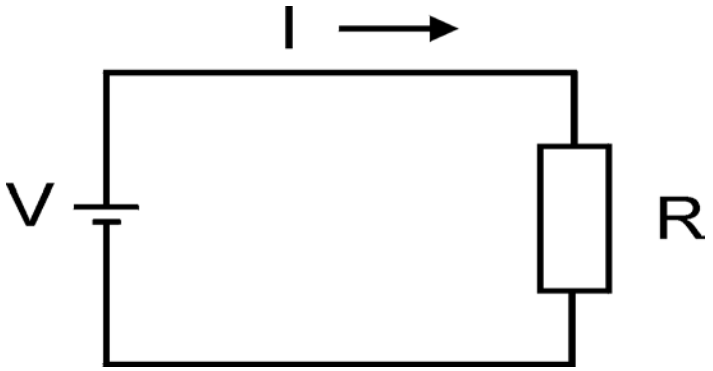
Sumário

- Calcular, atribuir
- estruturas `if ... then ... else`

---

1. Declarar variáveis, calcular valores e atribuição:

Como nos sabemos, o lei de Ohm é uma relação entre a corrente (*I*), a tensão (*V*) e a resistência (*R*), veja o circuito electrico abaixo:



Ohm's Law:

$$I = \frac{V}{R}$$

Escreve um programa que pede o utilizador valores de *V* e *R* e calcule a corrente *I*, por exemplo:

```
Qual o valor de V (volt)?
  12
Qual o valor de R (ohm)?
  1000
A corrente e 0.012
```

---

2. Muda o programa da pergunta 1 de forma que, quando o utilizador dá um valor da resistência negativo o programa vai gerar uma mensagem de erro, por exemplo:

```
Qual o valor de V (volt)?
  12
Qual o valor de R (ohm)?
  -1000
A resistencia nao pode ser negativa!
```

3.

Escreve um programa que calcule com números. O programa deve pedir o utilizador de escolher entre as opções 'multiplicar', 'addicionar', 'subtrair' e 'dividir', (Use a estrutura `if ... then ... else`)
Por exemplo:

```
numero 1: -1
numero 2: 3
Escolhe uma opcao:
1) addicionar
2) multiplicar
3) dividir
4) subtrair
 1
A soma e 2.0
```

4a. Números complexos

Números complexos são números especiais e muito usado no mundo de física, matemática e outras ciências. Os números são baseados na equação

$$i^2 = -1$$

Cada número complexo tem uma parte real e um parte imaginário. Então, um número complexo geral é de forma

$$z = a + b*i$$

onde a e b são números normais. Nota que o i não é uma variável, mas só um simbolo que represente um número imaginário.

Calcular com números complexos é fácil. Por exemplo, addicionar:

$$\begin{aligned} z_1 + z_2 &= (a_1 + b_1*i) + (a_2 + b_2*i) \\ &= (a_1 + a_2) + (b_1 + b_2)*i \end{aligned}$$

exemplo:

$$\begin{aligned} (1 - i) + (3 + 2i) &= (1 + 3) + (-1 + 2)i \\ &= 4 + i \end{aligned}$$

ou multiplicar:

$$\begin{aligned} z_1 * z_2 &= (a_1 + b_1*i) * (a_2 + b_2*i) \\ &= (a_1 * a_2) + (a_1* b_2)*i + (a_2 * b_1)*i + (b_1 * b_2)*i^2 \\ &= (a_1 * a_2 - b_1 * b_2) + (a_1* b_2 + a_2 * b_1)*\mathrm{i} \end{aligned}$$

exemplo:

$$\begin{aligned} (1 - i) * (3 + 2i) &= (1 * 3) + (1 * 2)*i + (-1 * 3)*i + (-1 * 2)i^2 \\ &= 3 + 2i - 3i + 2 \\ &= 5 - i \end{aligned}$$

Nota que o i não é uma variável, mas só um simbolo que represente um número imaginário.

Escreve um programa que pede o utilizador dois números complexos, multiplica os dois e mostra o resultado, por exemplo:

```
z1 = a1 + b1*i
Qual o valor de a1: 1
Qual o valor de b1: -1
z2 = a2 + b2*i
Qual o valor de a2: 3
Qual o valor de b2: 2
O resultado: 5.0 + -1.0i
```

4b.

Muda o programa da pergunta 4 da forma que o utilizador escolhe entre as opções 'mulitplicar' e 'addicionar' e o programa calcule um ou o outro.

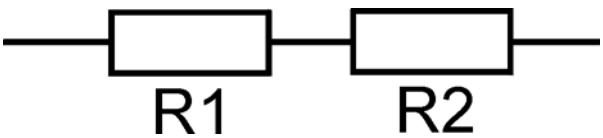
```
z1 = a1 + b1*i
Qual o valor de a1: 1
Qual o valor de b1: -1
z2 = a2 + b2*i
Qual o valor de a2: 3
Qual o valor de b2: 2
Escolhe uma opção:
1) addicionar
2) multiplicar
 1
A soma e 4.0 + 1.0i
```

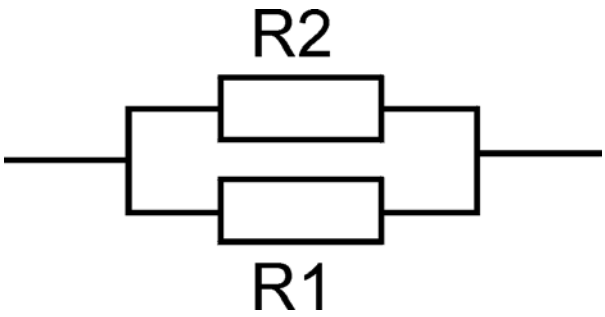
5.

Quem não gosta ou não percebe números complexos, pode escrever um programa que calcule a resistência equivalenta dum circuito com duas resistências em serie ou em paralel. O programa deve pedir o utilizador de escolher entre os dois

serial: $R = R_1 + R_2$

parallel: $R = R_1 * R_2 / (R_1 + R_2)$






```
Valor da resistencia 1: 1
valor da resistencia 2: 1
1) serie
2) paralel
 2
```

```
A resistencia equivalente e 0.500
```

---

soluções

Introdução a Computação

Soluções da aula prática 4

1.

```
PROGRAM Ohm;

Var v, r, i: real;

begin
  WriteLn('Qual o valor de V (volt)');
  ReadLn(v);
  WriteLn('Qual o valor de R (volt)');
  ReadLn(r);
  i := v/r;
  WriteLn('A corrente e ', i:0:3);
end.
```

resultado do programa:

```
Qual o valor de V (volt)?
  12
Qual o valor de R (ohm)?
  1000
A corrente e 0.012
```

2.

```
PROGRAM NegativeOhm;

Var v, r, i: real;

begin
  WriteLn('Qual o valor de V (volt)');
  ReadLn(v);
  WriteLn('Qual o valor de R (volt)');
  ReadLn(r);
  if (r<=0)
    writeln('A resistencia nao pode ser negativa!')
  else
    begin
      i := v/r;
      WriteLn('A corrente e ', i:0:3);
    end;
end.
```

resultado do programa:

```
Qual o valor de V (volt)?
  12
Qual o valor de R (ohm)?
  -1000
A resistencia nao pode ser negativa!
```

3.

```
PROGRAM Calculador;

Var a, b: real;
    opt: integer;

begin
  Write('numero 1: ');
  ReadLn(a);
  Write('numero 2: ');
  ReadLn(b);
  Writeln('Escolhe uma opcao: ');
  WriteLn('1) addicionar');
  WriteLn('2) multiplicar');
  WriteLn('3) dicidir');
  WriteLn('4) subtrair');
  readLn(opt);
  if (opt=1) then
    WriteLn('A soma e ', a + b:0:1)
  else
    if (opt=2) then
      WriteLn('O produto e ', a*b:0:1)
    else
      if (opt=3) then
        WriteLn('A divisao da ', a/b:0:1)
      else
        if (opt=4) then
          WriteLn('A diferenca e ', a-b:0:1)
        else
          WriteLn('Escolhe entre 1..4!');
end.
```

resultado do programa:

```
numero 1: -1
numero 2: 3
Escolhe uma opcao:
1) addicionar
2) multiplicar
3) dividir
4) subtrair
 1
A soma e 2.0
```

4a.

```
PROGRAM CalcularComplexo;

Var a1, b1, a2, b2, a3, b3: real;

begin
  WriteLn('z1 = a1 + b1*i');
  Write('Qual o valor de a1: ');
  ReadLn(a1);
  Write('Qual o valor de b1: ');
  ReadLn(b1);
  WriteLn('z2 = a2 + b2*i');
  Write('Qual o valor de a2: ');
  ReadLn(a2);
  Write('Qual o valor de b2: ');
  ReadLn(b2);
  a3 := a1 * a2 - b1 * b2;
  b3 := a1 * b2 + a2 * b1;
  WriteLn('O resultado: ',a3:0:1, ' + ', b3:0:1);
end.
```

```
z1 = a1 + b1*i
Qual o valor de a1: 1
Qual o valor de b1: -1
z2 = a2 + b2*i
Qual o valor de a2: 3
Qual o valor de b2: 2
O resultado: 5.0 + -1.0i
```

4b.

```
PROGRAM CalcularComplexoOpcao;

Var a1, b1, a2, b2, a3, b3: real;
    opt: integer;

begin
  WriteLn('z1 = a1 + b1*i');
  Write('Qual o valor de a1: ');
  ReadLn(a1);
  Write('Qual o valor de b1: ');
  ReadLn(b1);
  WriteLn('z2 = a2 + b2*i');
  Write('Qual o valor de a2: ');
  ReadLn(a2);
  Write('Qual o valor de b2: ');
  ReadLn(b2);
  Writeln('Escolhe uma opcao');
  Writeln('1) Addicionar');
  WriteLn('2) Multiplicar');
  ReadLn(opt);
  if (opt=1) then
```

```
    begin
      a3 := a1 + a2;
      b3 := b1 + b2;
      WriteLn('O resultado: ',a3:0:1, ' + ', b3:0:1);
    end
  else
    begin
      a3 := a1 * a2 - b1 * b2;
      b3 := a1 * b2 + a2 * b1;
      WriteLn('O resultado: ',a3:0:1, ' + ', b3:0:1);
    end;
end.
```

```
z1 = a1 + b1*i
Qual o valor de a1: 1
Qual o valor de b1: -1
z2 = a2 + b2*i
Qual o valor de a2: 3
Qual o valor de b2: 2
Escolhe uma opção:
1) addicionar
2) multiplicar
 2
A soma e 4.0 + 1.0i
```

5.

```
PROGRAM SerialParallel;

Var r, r1, r2: real;
    opt: integer;

begin
  Write('Valor da resistencia 1: ');
  ReadLn(r1);
  Write('Valor da resistencia 2: ');
  ReadLn(r2);
  Writeln('Escolhe uma opcao');
  Writeln('1) Serie');
  WriteLn('2) Paralel');
  ReadLn(opt);
  if (opt=1) then
    r := r1 + r2
  else
    r := r1*r2/(r1+r2);
  WriteLn('A resistencia equivalenta e ', r:0:3);
end.
```

Aula prática 5

Sumário

---

1. Reescreva o programa do exercicio 3 da aula prática 4 usando a estrutura **`Case ... Of`**.

Um programa de calculo. O programa deve pedir ao utilizador para escolher entre as opções 'multiplicar', 'addicionar', 'subtrair' e 'dividir', por exemplo:

```
numero 1: -1
numero 2: 3
Escolhe uma opcao:
1) addicionar
2) multiplicar
3) dividir
4) subtrair
 1
A soma e 2.0
```

---

2. Álgebra Booleana; combinar condições

Escreva um programa que pede ao utilizador dois números. O programa deve mostrar
1) O texto 'Ambos negativos' se os números forem ambos negativos
2) O texto 'No mínimo um é negativo' se pelo menos um for negativo
3) O texto 'Só um é negativo' se um for negativo mas não os dois.
Use operadores AND, OR e XOR
Por exemplo

```
numero 1: -1
numero 2: 3
No minimo um e negativo
So um e negativo
```

---

3. Álgebra Booleana com inteiros

Na aula 9 aprendemos álgebra Boolean com variáveis do tipo inteiro. Por exemplo, o exercicio do mini-teste 9

43 AND 33 dá 33

Agora, faça os cálculos com papel e lápis:

expressão	resultado
25 AND 49	

37 OR 11	
39 XOR 17	
7 OR 14	

Agora escreva um programa que calcule o resultado das mesmas expressões. O programa deve pedir ao utilizador dois números. Depois o utilizador pode escolher entre as opções 'AND', 'OR', 'XOR' e 'NOT', e o programa deve calcular o resultado da operação. Use variáveis do tipo byte. (Lembre-se, um byte tem 8 bit e só pode armazenar valores positivos, veja aula 3).
Verifique os seus resultados obtidos na tabela acima usando o seu programa.

---

4. Ciclos for.

Ciclos For são usado para repetir coisas numa maneira contável (veja aula 10)

Escreva um programa que escreve no ecrã 'Estou muito feliz' 1000 vezes.

4b: Mude o programa da forma a pedir ao utilizador

1) o texto a mostrar no ecran (use uma variável do tipo string)
2) o número de vezes que quer mostrar o texto
Por exemplo

```
texto a mostrar: Benfica o glorioso
numero de vezes: 3
Benfica o glorioso
Benfica o glorioso
Benfica o glorioso
```

---

5. Faça um pograma que escreve a tabuada de um determinado número (O número vem do utilizador).

Por exemplo

```
Um numero: 8
1 x 8 = 8
2 x 8 = 16
3 x 8 = 24
4 x 8 = 32
5 x 8 = 40
6 x 8 = 48
7 x 8 = 56
8 x 8 = 64
```

```
9 x 8 = 72
10 x 8 = 80
```

---

6. (difícil) Escreva um programa que pede ao utilizador um número (entre 0 e 255) e o programa mostrará o número em formato binário. (Sugestão: usa uma variável do tipo byte e usa as operações `x AND 128`, etc.) Exemplo

```
numero decimal: 33
binario: 00100001
```

---

soluções

Introdução a Computação

Soluções da aula prática 5

1.

```
PROGRAM Calculator;

Var a, b, c: real;
    opcao: integer;

begin
  Write('numero 1: ');
  ReadLn(a);
  Write('numero 2: ');
  ReadLn(b);
  WriteLn('Escolhe uma opcao:');
  WriteLn('1) addicionar');
  WriteLn('2) mulitplicar');
  WriteLn('3) dividir');
  WriteLn('4) subtrair');
  ReadLn(opcao);
  Case opcao of
    1: begin
         c := a + b;
         WriteLn('A soma e ',c:0:1);
       end;
    2: begin
         c := a * b;
         WriteLn('O produto e ',c:0:1);
       end;
    3: begin
         c := a / b;
         WriteLn('A divisao da ',c:0:1);
       end;
    4: begin
         c := a - b;
         WriteLn('A diferenca e ',c:0:1);
       end;
  end;
end.
```

```
numero 1: -1
numero 2: 3
Escolhe uma opcao:
1) addicionar
2) multiplicar
3) dividir
4) subtrair
```

```
 1
A soma e 2.0
```

2.

```
PROGRAM Combination;

Var a, b: integer;

begin
  Write('numero 1: ');
  ReadLn(a);
  Write('numero 2: ');
  ReadLn(b);
  if (a<0) AND (b<0) then
    WriteLn('Ambos negativos');
  if (a<0) OR (b<0) then
    WriteLn('No minimo um e negativo');
  if (a<0) XOR (b<0) then
    WriteLn('So um e negativo');
end.

numero 1: -1
numero 2: 3
No minimo um e negativo
So um e negativo
```

3.

expressão	resultado
25 AND 49	17
37 OR 11	47
39 XOR 17	54
7 OR 14	15

```
PROGRAM BooleanAlgebra;

Var a, b: byte;
    opcao: integer;

begin
  Write('numero a: ');
  ReadLn(a);
  Write('numero b: ');
  ReadLn(b);
  WriteLn('Escolhe uma opcao:');
  WriteLn('1) a AND b');
  WriteLn('2) a OR b');
  WriteLn('3) a XOR b');
  WriteLn('4) NOT a');
```

```
  ReadLn(opcao);
  Case opcao of
    1: WriteLn(a, ' AND ', b, ' = ', a AND b);
    2: WriteLn(a, ' OR ', b, ' = ', a OR b);
    3: WriteLn(a, ' XOR ', b, ' = ', a XOR b);
    4: WriteLn('NOT ', a, ' = ', NOT a);
  end;
end.
```

(Nota que o resultado do `NOT a` depende do tipo da variável)

4.

```
PROGRAM EstouFeliz;

Var i: integer;

begin
  for i := 1 to 1000 do
    WriteLn('Estou muito feliz');
end.
```

4b:

```
PROGRAM TextoAMostrar;

Var i, n: integer;
    s: string;

begin
  Write('texto a mostrar: ');
  ReadLn(s);
  Write('numero de vezes: ');
  ReadLn(n);
  for i := 1 to n do
    WriteLn(s);
end.

texto a mostrar: Benfica o glorioso
numero de vezes: 3
Benfica o glorioso
Benfica o glorioso
Benfica o glorioso
```

5.

```
PROGRAM Tabuada;

Var i, n: integer;

begin
  Write('Um numero: ');
  ReadLn(n);
```

```
  for i := 1 to 10 do
    WriteLn(i, ' x ', n, ' = ', i*n);
end.
```

```
Um numero: 8
1 x 8 = 8
2 x 8 = 16
3 x 8 = 24
4 x 8 = 32
5 x 8 = 40
6 x 8 = 48
7 x 8 = 56
8 x 8 = 64
9 x 8 = 72
10 x 8 = 80
```

6.

```
PROGRAM Binario;

Var a: byte;

begin
  Write('numero decimal: ');
  ReadLn(a);
  Write('binario: ');
  if (a AND 128)>0 then Write('1') else Write('0');
   (* 128 = 10000000 *)
  if (a AND 64)>0 then Write('1') else Write('0');
   (* 64 = 01000000 *)
  if (a AND 32)>0 then Write('1') else Write('0');
   (* 32 = 00100000 *)
  if (a AND 16)>0 then Write('1') else Write('0');
   (* 16 = 00010000 *)
  if (a AND 8)>0 then Write('1') else Write('0');
   (* 8 = 00001000 *)
  if (a AND 4)>0 then Write('1') else Write('0');
   (* 4 = 00000100 *)
  if (a AND 2)>0 then Write('1') else Write('0');
   (* 2 = 00000010 *)
  if (a AND 1)>0 then WriteLn('1') else WriteLn('0');
   (* 1 = 00000001 *)
end.
```

```
numero decimal: 33
binario: 00100001
```

ou uma solução com um ciclo Repeat-Until

```
PROGRAM Binario;

Var a, mask: byte;

begin
  Write('numero decimal: ');
```

```
  ReadLn(a);
  Write('binario: ');
  mask := 128     (* 128 = 10000000 *)
  repeat
    if (a AND mask)>0 then Write('1') else Write('0');
    mask := mask DIV 2;
      (* 128 -> 64 -> 32 -> 16 -> 8 -> 4 -> 2 -> 1 -> 0 *)
      (* 10000000 -> 01000000 -> 00100000 -> 00010000 -> *)
      (* 00001000 -> 00000100 -> 00000010 -> 00000001 -> *)
      (* 00000000 *)
  until mask < 1;
end.
```

agora com um ciclo For para mostrar todos os códigos binários entre 0 e 255:

```
PROGRAM Binario;

Var a, mask: byte;

begin
  For a := 0 To 255 Do
    begin
      Write('numero decimal: ');
      Write('binario: ');
        (* nao precisa pedir ao utilizador um numero *)
        (* 'a' vem do ciclo *)
      mask := 128
      repeat
        if (a AND mask)>0 then Write('1') else Write('0');
        mask := mask DIV 2;
      until mask < 1;
      WriteLn;           (* introduz nova linha *)
    end;
end.
```

Introdução a Computação
Aula prática 6

Sumário

- Ciclos For, Repeat-Until e While-Do
- Procedimentos sem *input* nem *output*

1.

Estatísticas simples. Você é dado(a) um a um uma série de números inteiros. Os ditos números podem ser positivos ou negativos. Quando o número introduzido seja -999, a série termina, SEM incluir o dito número. Para além de mostrar cada vez no ecrã o número inserido, **no fim da série** você deverá escrever no ecrã os resultados dos seguintes cálculos:
a) o valor médio dos números na série
b) o número menor e o número maior na serie
c) quantos dos números na série são pares e quantos são impares.
d) O número de valores introduzidos.

Dica: defina uma variável adequada para guardar os valores parciais para cada uma destas operações

---

2. Escreva um programa, onde cada uma das opções do exercício Nº 1 desta TP seja um procedimento.

Isto é, a leitura da série até o fim (quando o número -999 é introduzido) constitui um procedimento.
Cada uma das tarefas indicadas nas letras a, b, c, e, d, devem ser também procedimentos individuais.
Desta forma, a secção principal do programa executa só os procedimentos:
ler_dados, media, menor_maior , par_impar, total_numeros.
Por favor respeite estes mesmos nomes para os procedimentos.

---

3. Escreva um programa que simula uma calculadora simples que implementa as seguintes funções:

+, -, *, / , raiz quadrada, valor absoluto, seno e coseno dum ângulo.
O programa deve pedir primeiro ao utilizador(a) qual operação deseja efectuar, ou seja o operador. Logo deve pedir os operandos.
Pedir confirmação para cada operando. Se o utilizador assim decidir, deve poder mudar o valor previamente introduzido num operando.
Defina alguma variável que permita conhecer qual é a opção escolhida pelo utilizador(a).
Feito os cálculos os resultados devem ser amostrados no ecrã.
Este programa continua a oferecer o menú com as diferentes opções indefinidamente após amostrar os resultados, até que uma opção que indique o fim deste ciclo seja premida pelo utilizador(a).

---

4. Números primos: são aqueles divisíveis por 1 e por eles próprios.

Fazer um programa que verifica se um número introduzido pelo utilizador(a) é ou não um número primo.

---

5. Enquanto a soma total duma série de números introduzidos pelo utilizador(a) não seja superior a 1000, continue a executar um programa que adiciona os números introduzidos pelo utilizador, e que mostra no ecrã o valor da adição parcial deles.

---

soluções

Introdução à Computação
Aula prática 6 - Respostas

---

1 e 2 . Estatísticas simples utilizando procedimentos com variáveis globais.

```
program calculosm;
          {----programa que calcula estatisticas num ciclo ----}
uses crt;

var
num, total, numeros, maior, menor, pares, impares: integer;
{-------------------------------------------------------------------------}
procedure inicializacao;
begin
   total:= 0; pares:= 0; impares:= 0;
   maior:= -9999; menor:= 9999;
   writeln('Este programa vai calcular valores num ciclo até input=-999');
end;
{-------------------------------------------------------------------------}
procedure ler_dados;
begin
   writeln('Entrar um número: '); readln(num);
end;
{-------------------------------------------------------------------------}
procedure maior_menor;
begin
     if (num < menor) then
        menor:= num;
     if (num > maior) then
        maior:= num;
end;
{-------------------------------------------------------------------------}
procedure media;
begin
   writeln('M,dia dos números introduzidos: ', total / numeros:0:2);
end;
{-------------------------------------------------------------------------}
procedure par_impar;
begin
     if ((num mod 2) = 0) then
         pares:= pares + 1
     else
         impares:= impares + 1;
end;
{-------------------------------------------------------------------------}
procedure total_numeros;
begin
        total:= total + num;
end;
{-------------------------------------------------------------------------}
procedure estatisticas;
begin
   writeln(' Estatisticas ');
   writeln(' ------------ ');
   writeln('Total de números introduzidos: ', numeros);
   writeln('Números pares: ',  pares);
   writeln('Número impares: ', impares);
   writeln('Número menor: ',   menor);
   writeln('Número maior: ',   maior);
   media;
   readln;
end;
{-------------------------------------------------------------------------}
begin       {secção principal do programa }
```

```
clrscr;
inicializacao;
ler_dados;

while (num <> -999) do       {ciclo do programa }
   begin
        numeros:= numeros + 1;
        total_numeros;
        maior_menor;
        par_impar;
        ler_dados;
     end;
     estatisticas;    {mostrar resultados }
end.
```

3.

```
PROGRAM Calculadora;
uses crt;

var op1, op2, c: real;
    opcao: integer;
{-------------------------------------}
PROCEDURE Somar;
begin
  c := op1 + op2;
  WriteLn(' ', op1:0:2 , ' + ', op2:0:2 , ' = ',c:0:2);
end;
{-------------------------------------------------------------}
PROCEDURE Dividir;
begin
  c := op1 / op2;
  WriteLn(' ', op1:0:2 ,' / ', op2:0:2 , ' = ',c:0:2);
end;
{-------------------------------------------------------------}
PROCEDURE Multiplicar;
begin
  c := op1 * op2;
  WriteLn( ' ', op1:0:2 , ' * ', op2:0:2 , ' = ',c:0:2);
end;
{-------------------------------------------------------------}
PROCEDURE Subtrair;
begin
  c := op1 - op2;
  WriteLn(' ', op1:0:2 , ' - ', op2:0:2 , ' = ',c:0:2);
end;
{-----------------------------------------------------------------------}
FUNCTION graus_radians(angulo: real): real;
         {funcao que devolve o valor em radians dum angulo em graus }
begin
     graus_radians:= (angulo * PI) / 180;
end;
{-----------------------------------------------------------------------}
PROCEDURE seno(angulo: real); {funcao trabalho com radians e nao graus}
var r : real;
begin
    r:= graus_radians(angulo); {chama funcao que converte graus em radianes}
    writeln('Seno de ',angulo:0:2, ' = ', sin(r):0:4);
end;
{-----------------------------------------------------------------------}
PROCEDURE coseno(angulo: real); {funcao trabalha com radians }
var r : real;
begin
   r:= graus_radians(angulo);
   writeln('coseno de ',angulo:0:2, ' = ', cos(r):0:4);
end;
{-----------------------------------------------------------------------}
PROCEDURE valor_absoluto( valor: real);
begin
    writeln('Valor absoluto de ',valor:0:2, ' = ', (abs(valor)):0:2);
```

```
end;
{--------------------------------------------------------------------}
PROCEDURE raiz_quadrada(valor: real);
begin
    writeln('Raiz quadrada de ', valor:0:2, ' = ',sqrt(valor):0:2);
end;
{--------------------------------------------------------------------}
PROCEDURE ler_operandos;
var OK :char;
begin
   OK:= 'x';
   repeat
       if (opcao < 5) then
           begin
              Write('favor entrar operando 1: ');    readln(op1);
              Write('favor entrar operando 2: ');    readln(op2);
           end
       else
           begin
              write('Favor entrar valor a converter: '); readln(op1);
           end;
       write(' Esta seguro(a) dos seus operandos? '); readln(OK);
   until ((OK = 's') or (OK = 'S'));
end;


end;
{--------------------------------------------------------------------}
begin              { programa principal }
{--------------------------------------------------------------------}
  Repeat
    clrscr;
    WriteLn('            Escolhe uma opcao:');
    WriteLn('(1) adicionar             (2) multiplicar');
    WriteLn('(3) dividir               (4) subtrair');
    WriteLn('(5) valor absoluto        (6) raiz quadrada');
    WriteLn('(7) seno dum angulo       (8) coseno dum angulo');
    Writeln('(0) sair');
    write('              Qual , a sua op‡Æo?: ');
    ReadLn(opcao);
    if (opcao <> 0) then
      begin
        ler_operandos;
        Case opcao of
              1: Somar;
              2: Multiplicar;
              3: Dividir;
              4: Subtrair;
              5: valor_absoluto(op1);
              6: raiz_quadrada(op1);
              7: seno(op1);
              8: coseno(op1);
        else
              writeln('Erro no input. Repeta s.f.f.');
        end;
      readln;
      end;
  until (opcao = 0);

  WriteLn('Obrigado e bom dia');
  readln;
end.
```

---

4. Números primos: são aqueles divisíveis por 1 e por eles próprios.

Um numero é divisível pelo outro número se o resto da divisão é zero. (Na aula teórica 7 foram explicados os operadores para divisões, Div e Mod.) Um número n é primo se não for divisível por todos os números entre 2 e (n-1). Com estes duas ideas vamos resolver o problema. Sabemos exactamento quantas vezes temos de executar o ciclo: Então, utilizamos o ciclo **For**.

```
PROGRAM DeterminePrimo;
Var i, n: integer;
   primo: boolean;
begin
  Write('numero: ');
  ReadLn(n);
  primo := TRUE;
  For i := 2 To n-1 Do
      if (n Mod i) = 0 then  (* a Mod b = 0 significa que a variavel a e divisivel pela variavel b *)
      primo := FALSE;
     (* fim do ciclo For *)
  if  (primo) then
      WriteLn(n, ' e primo')
  else
    WriteLn(n, ' nao e primo');
end.
```

Para os especialistas: Existem algorítmos muito mais inteligentes para determinar se um número é primo. O programa acima é o mais simples. É relativamente fácil melhorar a eficiência do algoritmo: se encontramos uma vez que o resultado do (n Mod i)=0, não precisamos de continuar com o ciclo até verificar todos os números entre 0 e (n-1) Não é ?. Imagine vamos determinar se o número 10000 é primo. Só calculando 10000 Mod 2 sabemos que o número não é primo e podemos acabar com os cálculos. Vamos implementar esta idea.
Temos duas condições para sair do ciclo
1) chegamos ao fim com os números (i = (n-1)) ou
2) encontrámos um resultado do Mod que deu 0. Vamos combinar estas duas condições e temos de usar um outro tipo do ciclo (com ciclos For não é possivel combinar condições):

```
PROGRAM MelhorPrimo;
Var i, n: integer;     primo: boolean;
begin
Write('numero: ');   ReadLn(n);
primo := TRUE;   i := 2;

While ((i <= (n-1)) AND (primo = TRUE)) Do
   begin
      if  (n Mod i) = 0 then
         primo := FALSE;
      i := i + 1;     end;
      if (primo) then
         WriteLn(n, ' e primo')
     else
        WriteLn(n, ' nao e primo');
    end;
end.
```

A notar que na condição " if (primo) ...", em vez de primo=TRUE podemos escrever só primo, porque esta já uma variável do tipo boleana. Assim o código acima fica mais legível.

4a. Agora vamos por tudo num (outro) ciclo for:

```
PROGRAM DeterminePrimo;
Var i, n: integer;
   primo: boolean;
begin
  WriteLn('Numeros primos ate 10000:');
  for n := 3 to 10000 do
    begin
```

```
      primo := TRUE;
      for i := 2 to n-1 do
        if (n Mod i) = 0 then
          primo := FALSE;
      (* aqui acaba o ciclo For i *)
      if primo then
        Write(n,'   ');
    (* aqui acaba o ciclo For n *)
    end;
end.
```

---

5. Enquanto a soma total duma série de números introduzidos pelo utilizador(a) não seja superior a 1000, continue a executar um programa que adiciona os números introduzidos pelo utilizador, e que mostra no ecrã o valor da adição parcial deles.

```
program chegar_1000;
uses crt;

var
total, num: real;
n : integer;
begin
  clrscr;
  total:= 0; n:= 0;
  write('entrar um numero s.f.f: '); readln(num);
  if (num > 1000) then
    begin
      total:= num;   {situacao especial para primeira vez, caso o primeiro número > 1000 }
      n:= 1;
    end
  else
  repeat                  {também podia ser implementado com o ciclo while ...do }
     total:= total + num;
     if (total <= 1000) then
        begin
           n:= n + 1;
           writeln('---- Total parcial: ', total:0:2);
           if (total < 1000) then
             begin
               write('numero: '); readln(num);
             end;
        end;
  until (total >= 1000);

  writeln('Chegamos ao numero ', total:0:2);
  writeln('Numero de valores introduzidos: ', n);
  readln;
end.
```

Introdução a Computação

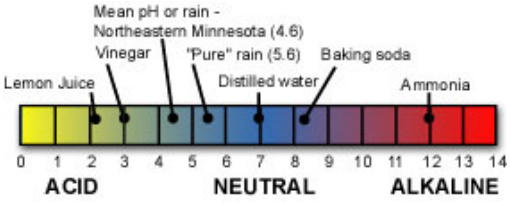
Aula prática 7

Sumário

- Funções e procedimentos sem e com parámetros
- Funções matemáticas implementadas em PASCAL
- números aleatórios

Nesta aula, usa funções e procedimentos onde é possível.

1a. Calcular o pH.



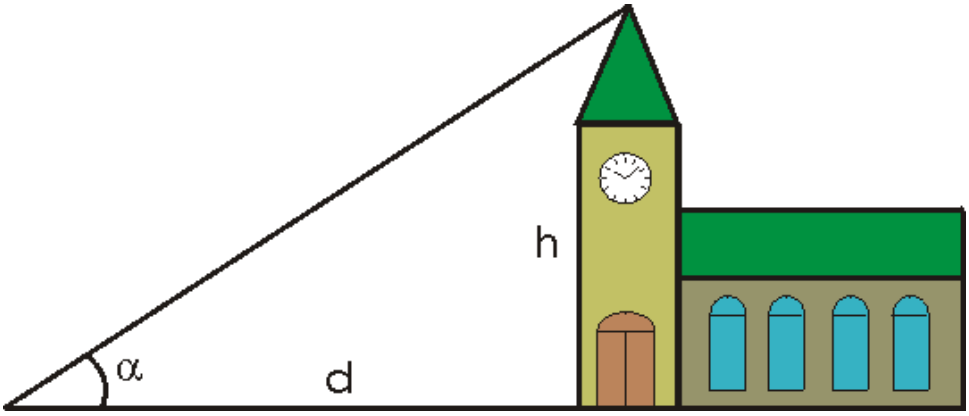
Como nos sabemos, o pH é um número que indica o acidez duma solução. Em mas pormenor, o pH diga a concentração dos iões de hidrogénio, $[H^+]$. Mais exacta, o pH é igual a $-^{10}\text{Log}\,[H^+]$. Com a unidade de concentração mol/l. Um pH muito baixo significa que a solução é muito ácido. Também, lembra a relação em agua: $[H^+][OH^-] = 10^{-14}\ \text{mol}^2\text{l}^{-2}$, então o $\text{pOH} = -^{10}\text{Log}([OH^-]) = -^{10}\text{Log}(10^{-14}/[H^+]) = 14 - \text{pH}$. Por isso, o pH da agua neutral, onde $[H^+] = [OH^-]$, é igual a 7.
Agora faça um programa que calcula o pH de uma solução. O utilizador pode escolher entre entrar a concentração dos iões de hidrogénio ou entrar a concentração dos iões de OH^-.
Por exemplo

```
Escolhe uma opcao:
1) [H+]
2) [OH-]
 1
Entra a concentracao de H+ em agua:
 1e-4
pH da solucao: 4.00
```

1b. Escreva um programa que faz o oposto. O utilizador entre o pH e o programa calculará as concentrações $[H^+]$ e $[OH^-]$. Por exemplo

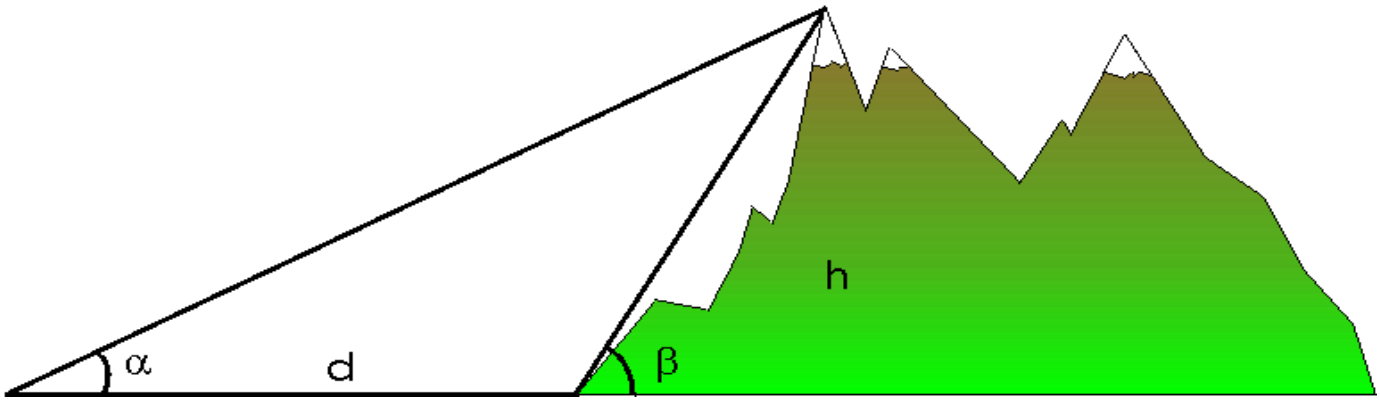
```
Diga o PH da solucao
 1.0
A concentracao [H+] = 1.0e-1 mol por litro
A concentracao [OH-] = 1.0e-13 mol por litro
```

2a. Calcular a altura de um edifício.



Determinar a altura de um objecto é nada fácil, mas com a ajuda de nosso computador será um pouco mais fácil. Imagine podemos só determinar a distância até o objecto e o ângulo de abertura do objecto. Com as regras de trigonometria é possível determinar a altura.
Escreva um programa que calcula a altura de um edifício (ou da serra) *h*. O utilizador deve entrar a distância até o objecto *d* e o ângulo de abertura do objecto, α.

```
Diga a distancia ate o objecto (m):
 100.0
Diga o angulo:
 15.0
A altura do objecto e 26.8 m.
```



2b. Programa de luxo: as vezes não é possível determinar a distância até o pé do objecto (imagine medir a altura de Mont Blanc). Ainde é possivel determinar a altura da serra se sabemos dois ângulos α e β e a distância entre os dois pontos de medição *d*.

```
Diga a distancia entre os dois pontos de medicao (m):
 8000.0
Diga o angulo 1:
 25.67
Diga o ângulo 2:
 67.41
A altura do objecto e 4807 m.
```

(a altura do Mont Blanc)

3. Mais um ciclo

Os números de Fibonacci são definidos da seguinta forma:

$f_1 = 1$

$f_2 = 1$

$f_n = f_{n-1} + f_{n-2}$

Faça um programa que mostra no ecrã os primeiros 30 números de Fibonacci. Usa um procedimento com nome `Fibonacci` que aceita um parámetro *n*, o número de números Fibonacci que o procedimento deve mostrar. O output deve ser igual a

```
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765 10946 17711 28657 46368 75025
121393 196418 317811 514229 832040
```

---

4. Random numbers.

a. Escreva um programa que gera a média de 100 números aleatórios. Usa uma função com nome `Media100` que gera os valores e retorna esse valor. Corre o programa duas vezes. Tem uma diferença? Muda o programa da forma que dá um valor diferente cada vez.

b. Escreva um programa que simule atirar uma moeda no ar 1000 vezes. O programa deve contar quantas vezes sai "cara" e quantas vezes sai "coroa".

---

soluções

Introdução a Computação

Soluções da aula prática 7

1a.

```
PROGRAM CalulatepH;

var conc, pH: real;
    opcao: integer;

PROCEDURE ConcH;
begin
  WriteLn('Entra a concentracao de H+ em agua (mol/l)');
  ReadLn(conc);
  pH := -Ln(conc)/Ln(10);
  WriteLn('pH da solucao: ', pH:0:2);
end;

PROCEDURE ConcOH;
begin
  WriteLn('Entra a concentracao de OH- em agua (mol/l)');
  ReadLn(conc);
  pH := 14.0-Ln(conc)/Ln(10);
  WriteLn('pH da solucao: ', pH:0:2);
end;

begin
  WriteLn('Escolha uma opcao:');
  WriteLn('1) [H+]');
  WriteLn('2) [OH-]');
   (* nota a forma de chamar o procedimento: *)
  ReadLn(opcao);
  Case opcao of
    1: ConcH;
    2: ConcOH;
  end;
end.
```

```
Escolhe uma opcao:
1) [H+]
2) [OH-]
 1
Entra a concentracao de H+ em agua:
 1e-4
pH da solucao: 4.00
```

1b.

```
PROGRAM CalulateConc;
```

```
var pH: real;

FUNCTION CalcH: real;
begin
   (* Nota a forma de retornar um valor, atraves o nome da funcao: *)
  CalcH := Exp(-pH*Ln(10));
end;

FUNCTION CalcOH: real;
begin
   (* Nota a forma de retornar um valor, atraves o nome da funcao: *)
  CalcOH := 1.0e-14/CalcH;
end;

begin
   (* O programa comeca aqui *)
  WriteLn('Diga o pH da solucao:');
  ReadLn(pH);
   (* nota a forma de usar a informacao que vem da funcao *)
  WriteLn('A concentracao [H+] = ', CalcH, ' mol por litro');
  WriteLn('A concentracao [OH-] = ', CalcOH, ' mol por litro');
end.
```

```
Diga o PH da solucao
 1.0
A concentracao [H+] = 1.0e-1 mol por litro
A concentracao [OH-] = 1.0e-13 mol por litro
```

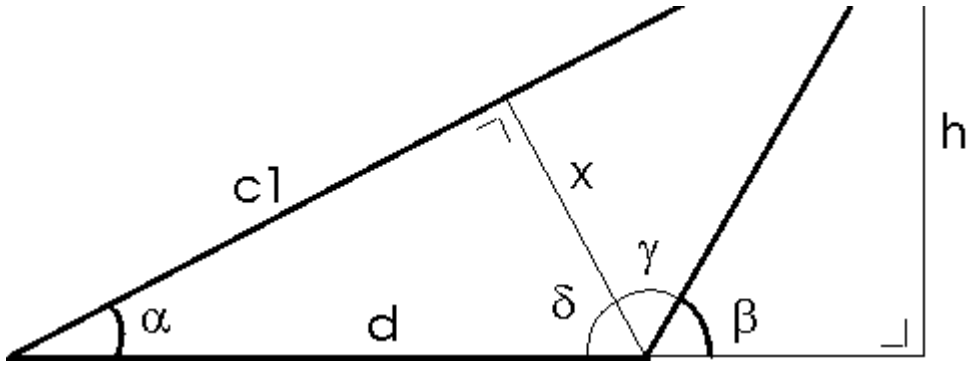
2a.

```
PROGRAM CalulateHeight;

var d, h, alfa: real;

begin
  WriteLn('Diga a distancia ate o objecto (m):');
  ReadLn(d);
  WriteLn('Diga o angulo:');
  ReadLn(alfa);
  h := d * Sin(Pi*alfa/180.0)/Cos(Pi*alfa/180.0);
  WriteLn('A altura do edificio e ', h ' m');
end.
```

```
Diga a distancia ate o objecto (m):
 100.0
Diga o angulo:
 15.0
A altura do objecto e 26.8 m.
```

2b. Tem *montes* de soluções. Aqui vem uma:

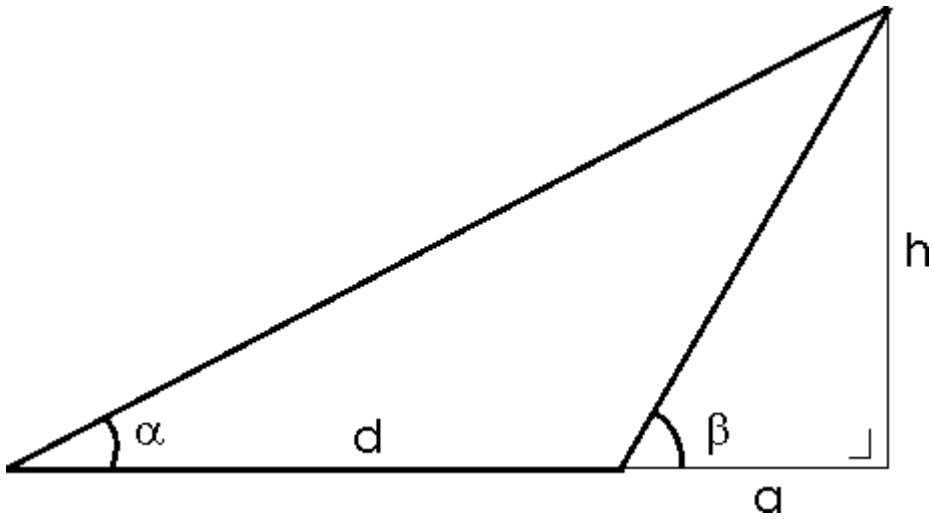
$$x = d \sin(\alpha)$$
$$c1 = d \cos(\alpha)$$
$$\gamma = 90^{o} + \alpha - \beta$$



$c2 = x \tan(\gamma)$
$h = (c1 + c2) \sin(\alpha)$

```
PROGRAM Mountains;

var alfa, beta, gama: real;
    d, x, c1, c2, h: real;
begin
  WriteLn('Diga a distancia entre os dois pontos de medicao (m)');
  ReadLn(d);
  WriteLn('Diga o angulo 1');
  ReadLn(alfa);
  WriteLn('Diga o angulo 2');
  ReadLn(beta);
  x := d * Sin(Pi*alfa/180.0);
  c1 := d * Cos(Pi*alfa/180.0);
  gama := 90 + alfa - beta;
  c2 := x * Sin(Pi*gama/180)/Cos(Pi*gama/180.0);
  h := (c1+c2) * Sin(Pi*alfa/180.0);
  WriteLn('A altura do objecto e ',h:0:0,' m.');
end.
```

Aqui vem mais uma:



$\tan(\alpha) = h/(d+a)$
$\tan(b) = h/a$

$h = d*\tan\alpha/(1-\tan\alpha/\tan\beta)$

```
PROGRAM Mountains;
var alfa, beta: real;
    d, a, h: real;
  FUNCTION Tan(ang: real): real;
  begin
    Tan := Sin(ang)/Cos(ang);
  end;
begin
  WriteLn('Diga a distancia entre os dois pontos de medicao (m)');
  ReadLn(d);
  WriteLn('Diga o angulo 1');
  ReadLn(alfa);
```

```
  WriteLn('Diga o angulo 2');
  ReadLn(beta);
  alfa := Pi*alfa/180.0;
  beta := Pi*beta/180.0;
  h := d * Tan(alfa)/(1-Tan(alfa)/Tan(beta));
  WriteLn('A altura do objecto e ',h:0:0,' m.');
end.
```

```
Diga a distancia entre os dois pontos de medicao (m):
 8000.0
Diga o angulo 1:
 25.67
Diga o ângulo 2:
 67.41
A altura do objecto e 4806 m.
```

3.

```
PROGRAM FibonacciSeries;

PROCEDURE Fibonacci(n: longint);
  (* o numero maximo sera 832040, entao so um longint chega: *)
Var f, fa, fb: longint;
begin
  fa := 1;
  fb := 1;
  WriteLn(fa);
  WriteLn(fb);
  for i := 3 to n do
    begin
      f := fa + fb;
      Write(f,' ');
      fa := fb;
      fb := f;
    end;
end;

begin
  Fibonacci(30);
end.
```

```
1 1 2 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765 10946 17711 28657
46368 75025 121393 196418 317811 514229 832040
```

4a.

```
PROGRAM Media;

FUNCTION Media100;
Var i: integer;
    sum: real;
begin
  sum := 0.0;
```

```
  for i := 1 to 100 do
    sum := sum+Random;
  Media100 := sum/100.0;
end;

begin
    (* com a seguinta instrucao o programa vai correr cada vez
       diferente: *)
  Randomize;
    (* chama a funcao e mostra o resultado: *)
  WriteLn('Media de 100 numeros aleatorios: ', Media100);
end.
```

4b.

```
PROGRAM Moeda;

Var i: integer;
    cara, coroa: integer;

begin
  Randomize;
    (* nunca esquece inicializar as variaveis: *)
  cara := 0;
  coroa := 0;
  for i := 1 to 1000 do
    if (Random(2) = 1) then
      cara := cara + 1
    else
      coroa := coroa + 1;
  WriteLn('Cara: ', cara, ' vezes');
  WriteLn('Coroa: ', cara, ' vezes');
  if (cara=coroa) then
    WriteLn('Empate!')
  else
    if cara>coroa then
      WriteLn('Cara ganhou!');
    else
      WriteLn('Coroa ganhou!');
end.
```

```
Cara: 459 vezes
Coroa: 541 vezes
Coroa ganhou!
```

Aula prática 8

1a. Escreva um programa que gera 1000 números aleatórios (inteiros) entre −10 e 20 (inclusive). Usa uma função com nome `Aleatorio` que retorna um valor neste intervalo.

1b. Muda o programa da questão 1a da forma que a função tem dois parâmetros, `a` e `b`, para especificar os limites do intervalo.

1c. Muda o programa da questão 1b: Determine o número mínimo e máximo da série.

---

2. Faça um programa que determine o factorial de um número

$$n! = n \times (n-1) \times (n-2) \ldots \times 2 \times 1$$

$$n! = n \times (n-1)!$$
$$1! = 1$$

2a: com um ciclo.
2b: com uma função recursiva.

3a. Prediza os resultados dos programas a seguir. Qual programa usa a técnica de passagem por referência e qual usa a técnica de passagem por valor?

pogram:

```
PROGRAM Program4a1;

Var m: integer;

PROCEDURE Operations(n: integer);
begin
  n := n+1;
  writeln(2*n);
end;

begin
  m := 1;
  Operations(m);
  WriteLn(m);
end.
```

pogram:

```
PROGRAM Program4a2;

Var m: integer;

PROCEDURE Operations(Var n: integer);
begin
  n := n+1;
  writeln(2*n);
end;

begin
  m := 1;
  Operations(m);
  WriteLn(m);
end.
```

output: *output*:

3b. Verifica as suas respostas.

---

4. Acaba os trabalhos da aula prática 7.

Para os especialistas:

5a. Introduz uma variável local na função `Factorial` do programa do trabalho 2b. Quando chamamos a função `Factorial(5)`, quantas cópias da variável local existem no máximo? (veja aula teórica 16).

resposta:

5b. Verifica a sua resposta (por exemplo com uma variável contadora global).

5c. Mais difícil: Faça a mesma coisa com a função `Fibonacci` da aula teórica 16. Quantas variáveis existirão no máximo depois a chamada `Fibonacci(5)`?

resposta:

---

soluções

Soluções da aula prática 8

1a.

```
PROGRAM HundredRandom;

Var i: integer;

FUNCTION Aleatorio: integer;
begin
  Aleatorio := Random(31)-10;
    (* Random(31) returns a number between 0 and 30 (incl.) *)
    (* Random(31)-10 returns between -10 and 20 (incl.) *)
end;

begin
  for i := 1 to 1000 do
    WriteLn(Aleatorio);
end.
```

1b.

```
PROGRAM HundredRandom;

Var i: integer;

FUNCTION Aleatorio(a, b: integer): integer;
begin
  Aleatorio := Random(b-a+1)+a;
   (* numero minimo: a, numero maximo: b-a + a = b *)
end;

begin
  for i := 1 to 1000 do
    WriteLn(Aleatorio(-10, 20));
end.
```

1c.

```
PROGRAM MinMaxRandom;
  (* determine o numero minimo e maximo de uma serie de numeros
     aleatorios *)

Var i, n, maxn, minn: integer;

FUNCTION Aleatorio(a, b: integer): integer;
begin
  Aleatorio := Random(b-a+1)+a;
end;

begin
```

```
   (* initiliazar com numeros grandes: *)
  minn := 32000;
  maxn := -32000;
  for i := 1 to 1000 do
    begin
      n := Aleatorio(-10, 20);
      if n>maxn then maxn := n;
      if n<minn then minn := n;
    end;
  WriteLn('Min: ', minn);
  WriteLn('Max: ', maxn);
end.
```

2a. cópia da aula teórica 16.

```
PROGRAM TestFactorial;

FUNCTION Factorial(n: integer): integer;
 (* returns n! *)
var result: integer;
    i: integer;
begin
  result := 1;         (* initialize the variable *)
  for i := 1 to n do
    result := i*result;
  Factorial := result;  (* return result *)
end;

begin
  WriteLn(Factorial(5));
end.
```

2b.

```
PROGRAM TestFactorial;

FUNCTION Factorial(n: integer): integer;
 (* returns n! *)
begin
  if n=1 then
    Factorial := 1
  else
    Factorial := n*Factorial(n-1);
end;

begin
  WriteLn(Factorial(5));
end.
```

3a.

PASSAGEM POR VALOR:

pogram:

```
PROGRAM Program4a1;

Var m: integer;

PROCEDURE Operations(n: integer);
begin
  n := n+1;
  writeln(2*n);
end;

begin
  m := 1;
  Operations(m);
  WriteLn(m);
end.
```

output:

```
4
1
```

PASSAGEM POR REFERÊNCIA

pogram:

```
PROGRAM Program4a2;

Var m: integer;

PROCEDURE Operations(Var n: integer);
begin
  n := n+1;
  writeln(2*n);
end;

begin
  m := 1;
  Operations(m);
  WriteLn(m);
end.
```

output:

```
4
2
```

5a.

resposta: 5

5b.

```
PROGRAM TestFactorial;

Var globalcounter: integer;

FUNCTION Factorial(n: integer): integer;
Var m: integer; (* local variable *)
begin
   (* uma copia da variavel local m sera criada *)
   (* vamos contar isso: *)
  globalcounter := globalcounter + 1;
  WriteLn('m existe ', globalcounter, ' vezes');
  if n=1 then
    Factorial := 1
  else
    Factorial := n*Factorial(n-1);
   (* a copia da variavel local m sera anihilada *)
   (* vamos contar isso: *)
  globalcounter := globalcounter - 1;
  WriteLn('m existe ', globalcounter, ' vezes');
end;

begin
  globalcounter := 0;
  WriteLn(Factorial(5));
end.
```

output:

```
m existe 1 vezes
m existe 2 vezes
m existe 3 vezes
m existe 4 vezes
m existe 5 vezes
m existe 4 vezes
m existe 3 vezes
m existe 2 vezes
m existe 1 vezes
m existe 0 vezes
120
```

5b.

```
PROGRAM TestFibonacci;

Var globalcounter: integer;

FUNCTION Fibonacci(n: integer): integer;
Var m: integer; (* local variable *)
begin
    (* uma copia da variavel local m sera criada *)
    (* vamos contar isso: *)
  globalcounter := globalcounter + 1;
  WriteLn('m existe ', globalcounter, ' vezes');
  if (n=1) OR (n=2) then
     Fibonacci := 1
  else
     Fibonacci := Fibonacci(n-1) + Fibonacci(n-2);
    (* a copia da variavel local m sera anihilada *)
    (* vamos contar isso: *)
  globalcounter := globalcounter - 1;
  WriteLn('m existe ', globalcounter, ' vezes');
end;

begin
  globalcounter := 0;
  WriteLn(Fibonacci(5));
end.
```

output:

```
m existe 1 vezes
m existe 2 vezes
m existe 3 vezes
m existe 2 vezes
m existe 3 vezes
m existe 2 vezes
m existe 1 vezes
m existe 2 vezes
m existe 3 vezes
m existe 2 vezes
m existe 3 vezes
m existe 4 vezes
```

```
m existe 3 vezes
m existe 4 vezes
m existe 3 vezes
m existe 2 vezes
m existe 1 vezes
m existe 0 vezes
5
```

Aula prática 9

---

1. Escreva um programa que calcule o saldo na conta bancária no fim de cada ano. O programa deve pedir ao utilizador
   1) O capital no início.
   2) A taxa de juros.
   3) O ano de começar e o ano de acabar a tabela.

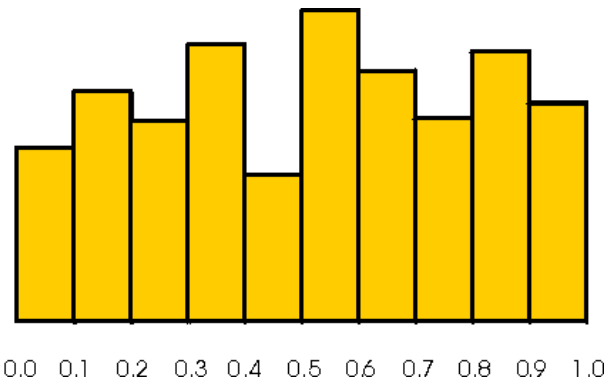
---

2. Declara num programa um record para guardar a informação de uma data. Tem os seguintes campos para guardar
1) dia da semana ('domingo' até 'sabado').
2) dia do mês.
3) mês.
4) ano.

Agora escreva um program que vai pedir ao utilizador os valores para cada campo e depois o programa deve calcular o número do dia do ano (por exemplo, dia 3 de fevereiro é dia 34 do ano).

---

3. Escreva um programa que vai gerar 1000 números aleatórios entre 0.0 e 1.0. O programa deve contar quantos números caiam nos dez intervalos 0..0.1, 0.1..0.2, 0.2..0.3, 0.3..0.4, 0.4..0.5, 0.5..0.6, 0.6..0.7, 0.7..0.8, 0.8..0.9, e 0.9..1.0.
Sugestões: Usa um array de dez elementos para contar. Usa a função `Trunc` (aula 14) para determinar no qual intervalo um número caiu.



---

4. Escreva um progama que declara um array de 100 números. Vamos guardar lá os primeiros 100 números primos.
Então vamos usar um array de tipo _____.
Escreva o resto do programa para calcular os números primos.

---

soluções

Soluções da aula prática 9

1.

```
PROGRAM Banco;

Var taxa: real;
    capital: real;
    i, ano1, ano2: integer;

begin
  Write('Capital : ');
  ReadLn(capital);
  Write('Taxa de juros: ');
  ReadLn(taxa);
  Write('ano comeco: ');
  ReadLn(ano1);
  Write('ano fim: ');
  ReadLn(ano2);
  WriteLn(i,'  ',capital:0:2);
  for i := ano1+1 to ano2 do
    begin
      capital := (1+taxa/100.0)*capital;
      WriteLn(i,'  ',capital:0:2);
    end;
end.
```

2.

```
Faz parte do trabalho pratico
```

3.

```
PROGRAM ContaAleat;

Var bins: array[1..10] of integer;
    i: integer;
    binnr: integer;
    aleat: real;

begin
   (* inicializar o array: *)
  for binnr := 1 to 10 do
    bins[binnr] := 0;
   (* gerar 1000 numeros: *)
```

```
  for i := 1 to 1000 do
    begin
        (* gerar um numero: *)
      aleat := Random;
        (* determinar o numero da caixa: *)
      binnr := Trunc(10*aleat)+1;
        (* aumentar o conteudo da caixa: *)
      bins[binnr] := bins[binnr] + 1;
    end;
   (* mostrar o resultado: *)
  for binnr := 1 to 10 do
    begin
      write((binnr-1)/10.0:0:1,'..',binnr/10.0:0:1,'   : ');
      writeln(bins[binnr]);
    end;
end.
```

possível output:

```
0.0..0.1   : 100
0.1..0.2   : 99
0.2..0.3   : 101
0.3..0.4   : 97
0.4..0.5   : 90
0.5..0.6   : 108
0.6..0.7   : 103
0.7..0.8   : 90
0.8..0.9   : 110
0.9..1.0   : 102
```

4.

```
PROGRAM Primos100;

Var primos: array[1..100] of integer;
    numero: integer;
    nprimos: integer;

FUNCTION IsPrimo(n: integer): boolean;
 (* retorna TRUE se n e numero primo *)
Var i: integer;
begin
  IsPrimo := TRUE;
  For i := 2 To n-1 Do
    if (n Mod i) = 0 then  (* a Mod b = 0 significa a e divisivel pelo b *)
      IsPrimo := FALSE;
end;

begin
  (* Nao sabemos quantos vezes temos de executar o ciclo *)
  (* no minimo uma vez: entao ciclo repeat-until *)
  (* mas tambem facilmente possivel com ciclo while-do *)
  (* Ao primeiro inicializar os parametros de controlo: *)
  nprimos := 0;    (* ainda nao temos nenhum primo *)
  numero := 1;     (* vamos comecar com determinar se 1 e primo *)
```

```
  repeat
    If IsPrimo(numero) then
      begin
         (* se numeor for primo, faz a administracao: *)
        nprimos := nprimos + 1;
        primos[nprimos] := numero;
      end;
    numero := numero + 1;
  until (nprimos=100);
    (* mostrar o resultado: *)
  for nprimos := 1 to 100 do
    Write('primo',nprimos,'=',primos[nprimos],' ');
end.
```

output (recortado):

```
primo1=1 primo2=2 primo3=3 primo4=5 primo5=7 primo6=11 primo7=13 ... primo100=523
```

Aula prática 10

---

1. Escreva um programa que pede ao utilizador 10 números. O programa deve ordenar os números e mostrar a lista ordenada.

2.

- Define um array (100 elementos) de records com campos `numero` (integer) e `div3` (boolean).
- Enche o array com números aleatórios entre 14 e 114. (guardar no campo `numero`)
- Determine quais os records contêm valores divisíveis por 3. Guarda isto no campo `div3` de cada record.
- Determine quantos valores foram divisível por 3.

Usa funções e procedimentos onde possível. Define novos tipos de variáveis!

---

3. Escreva um programa que guarda a informação dos seus amigos. As informações são nome e número telefónico. Usa um procedimento para inicializar o array com os dados (que não vêem do utilizador). O programa deve pedir um nome ao utilizador e como resposta dá o respectivo número telefónico. Exemplo do programa a funcionar:

```
Diz um nome:
alex
O numero do alex e 289123456
```

---

4. Declare

1 um word `w`
2 um integer `i`
3 um apontador ao um word `wp`
4 um apontador ao um integer `ip`

Faça uma atribuição –1 ao `i` e 1 ao `w`

Faça uma atribuição do endereço de `w` ao `wp` e o endereço de `i` ao `ip`.
Mostra o conteúdo dos endereços `wp` e `ip`.

Faça o contrário: uma atribuição do endereço de `i` ao `wp` e o endereço de `w` ao `ip`.
Mostra o conteúdo dos endereços `wp` e `ip`.

---

5. Determine quanto tempo leva executar os programas `TestSpeed` da aula teórica 20.

s e s

---

soluções

Soluções da aula prática 10

1.

```
PROGRAM Orden;

CONST N = 10;

Var ra: array[1..N] of real;
    i: integer;
    changed: boolean;
    temp: real;

begin
   (* pedir 10 numeros: *)
  for i := 1 to N do
    begin
      WriteLn('Numero ',i,':');
      ReadLn(ra[i]);
    end;
  changed := TRUE;
   (* continue ate nao houve mais mudancas: *)
  while changed do
    begin
      changed := FALSE;
      for i := 1 to N-1 do
          (* se ra[i] maior que ra[i+1] troca os dois *)
          (* e asinala que houve mudanca: *)
        if ra[i]>ra[i+1] then
          begin
            temp := ra[i];
            ra[i] := ra[i+1];
            ra[i+1] := temp;
            changed := TRUE;
          end;
    end;
  for i := 1 to N do
    Writeln(i,'  ',ra[i]:0:1);
end.
```

2.

```
PROGRAM Recs;

type rec =
  record
    numero: integer;
```

```
    div3: boolean;
  end;
Var ra: array[1..100] of rec;
    i: integer;
    totdiv3: integer;

begin
  for i := 1 to 100 do
    ra[i].numero := Random(101)+14;
  for i := 1 to 100 do
    if ((ra[i].numero MOD 3) = 0) then
      ra[i].div3 := TRUE
    else
      ra[i].div3 := FALSE;
  totdiv3 := 0;
  for i := 1 to 100 do
    if (ra[i].div3=TRUE)
      then totdiv3 := totdiv3+1;
  WriteLn(totdiv3);
end.
```

3.

```
PROGRAM Recs;

type rec =
  record
    nome: string;
    numero: longint;
  end;
Var ra: array[1..100] of rec;
    s: string;
    i: integer;

PROCEDURE Initialize;
begin
  ra[1].nome := 'Alex';
  ra[1].numero := 289123456;
  ra[2].nome := 'Sandra';
  ra[2].numero := 289234567;
  ra[3].nome := 'Joao';
  ra[3].numero := 289345678;
  ra[4].nome := 'Carla';
  ra[4].numero := 289456789;
  ra[5].nome := 'Hugo';
  ra[5].numero := 289567890;
end;

begin
  Initialize;
  Writeln('Diz um nome:');
  ReadLn(s);
  for i := 1 to 5 do
    if (ra[i].nome = s) then
```

```
      WriteLn('O numero do ', s, ' e ', ra[i].numero);
end.
```

4a.

```
PROGRAM Recs;

Var w: word;
    i: integer;
    wp: ^word;
    ip: ^integer;

begin
  i := -1;
  w := 1;
  wp := Addr(w);
  ip := Addr(i);
  WriteLn('ip: ', ip^);
  WriteLn('wp: ', wp^);
end.
```

output:

```
ip: -1
wp: 1
```

4b.

```
PROGRAM Recs;

Var w: word;
    i: integer;
    wp: ^word;
    ip: ^integer;

begin
  i := -1;
  w := 1;
  ip := Addr(w);
  wp := Addr(i);
  WriteLn('ip: ', ip^);
  WriteLn('wp: ', wp^);
end.
```

output:

```
ip: 1
wp: 65535
```

O programa do lado direito mostra que um integer com valor -1 é igual a 1111111111111111 binário, o que é igual a 65535 se for um word.
O conteúdo do um edereço depende do tipo de informação que fica lá.

5.

Veja aula teórica 20.

Aula prática 10

Esta aula consta de duas partes. A primeira está destinada ao alunos(as) que fazem esta disciplina pela primeira vez. A segunda parte está destinada aos repetentes que já fizeram os exercicios da primeira parte anteriormente. Contudo, os alunos(as) novos que desejem, podem fazer os exercicios da segunda parte em substituição da primeira.
Recomendação geral: *Fazer sempre o algoritmo de cada exercício com lápis, papel, bonecos e desenhos. Só quando tiver claro o que vai fazer, é que escreve as instrucções em Pascal.*

Parte A

---

1. Escreva um programa que pede ao utilizador 10 números. O programa deve ordenar os números e mostrar a lista ordenada.

2.

- Defina um array (capaz de conter 100 elementos) de *records* com os seguintes campos:
  a) `numero` de tipo inteior e `b) div3` de tipo boleano.
- Preencher este array com números aleatórios entre 14 e 114 e guardar no campo `numero`.
- Determine um a um todos registos cujos valores no campo *numero* são divisíveis por 3. Caso isto seja verdadeiro, armazenar esta resposta no campo `div3` do registo correspondente.
- No fim revise o array todo e calcule quantos valores são divisíveis por 3.
  Utilizar funções e procedimentos onde quer que seja possível. Defina novos tipos de variáveis!

---

3. Escreva um programa que guarda informações sobre os seus amigos. As informações são *nome* e *número telefónico*. Utilice um procedimento para inicializar o array com os dados (que não vêem do utilizador).
Posteriormente, o programa deve ser capaz de: pedido um nome ao utilizador e responder devolvendo o respectivo número telefónico. Exemplo do programa a funcionar:

```
Diga um nome: cheila
O numero de telefone da cheila é 289.693459
```

---

4. Declare

1 uma variável do tipo word `w`
2 uma variável do tipo integer `i`
3 um apontador ao um word `wp`
4 um apontador ao um integer `ip`

Atribuir os valores –1 ao `i` e 1 ao `w`

Agora faça uma atribuição do endereço de `w` ao `wp` e o endereço de `i` ao `ip`.
Mostra o conteúdo dos endereços `wp` e `ip`.

Seguidamente faça o contrário: atribuir o endereço de `i` ao `wp` e o endereço do `w` ao `ip`.
Mostrar o conteúdo dos endereços `em wp` e `ip` e aquele das variáveis apontadas pelos mesmos endereços.

5. Determine quanto tempo leva executar os programas `TestSpeed` da aula teórica 20.

s e s

B. Destinada aos alunos(as) repitentes.

---

6. Escreva um programa que faz o desenho dum triângulo de Pascal com oito (8) linhas. Como se sabe, este triângulo tem muitas propriedades interessantes para as matemáticas e as estatísticas. Nele encontramos números primos, números triangulares, a serie de Fibonacci, combinatória, etc.

Na imagem seguinte pode-se ver o exemplo do triângulo de Pascal que queremos construir:

							1							
						1		**1**						
					1		**2**		**1**					
				1		**3**		**3**		**1**				
			1		**4**		**6**		**4**		**1**			
		1		**5**		**10**		**10**		**5**		**1**		
	1		**6**		**15**		**20**		**15**		**6**		**1**	
1		**7**		**21**		**35**		**35**		**21**		**7**		**1**

O triângulo de Pascal é construído com um algoritmos simples. Será o mesmo utilizado por você para fazer na linguagem do mesmo nome do matemático francês:

1. A primeira linha tem um ‘1’ no centro. (o centro da maior linha, ou seja a última.)
2. Nas restantes linhas o valor de cada célula (ou coluna) corresponde **a soma dos valores nas diagonais da linha precedente** . Por exemplo, a linha 3 tem ‘1’ na coluna 6 e ‘2’ na coluna 8. Por tanto na coluna 7 da linha 4 corresponde a soma de 1 + 2 = 3.
   Isto é: linha[n+1, x] = linha[n, x-1] + linha[n, x+1].
3. Para calcular a soma das colunas na diagonal da linha precedente, imagine sempre que as posições em branco correspondem ao valor zero. De facto você pode trabalhar com uma matriz assim(zeros ou valores), e só no momento de imprimir pode converter os zeros em espaços em branco.
4. O número de colunas em cada linha está relacionado com o número de linhas que desejamos preencher. Você vai descobrir facilmente. Uma vez calculado isso, você sabe onde é que será inserido o primeiro ‘1’ da primeira linha.
5. O primeiro valor inserido em cada linha é sempre ‘1’ e é inserido na coluna anterior (na esquerda) respecto a posição onde começou a linha precedente, excepto a primeira linha claro. (0 + 1 = 1). Por isto, é fundamental saber sempre onde foi inserido o primeiro valor da linha anterior.

Mais dicas:
Criar uma função que recebe como argumentos una linha e uma coluna, e devolve a soma das diagonais da linha precedente. Pode ser muito útil.
Pode trabalhar só com duas linhas: actual e precedente e imprimir linha a linha. Porque depois de tudo, o único necessário saber para preencheer e imprimir uma linha são os valores que tive a linha precedente.

Alternativamente pode utilizar a matriz completa tal como no gráfico e imprimir no fim, após preenchida a matriz toda. Em quaisquer caso, não esqueça de inicializar os arrays utilizados.

Para saber mais sobre o triângulo de Pascal ver por exemplo em:
http://mathforum.org/dr.math/faq/faq.pascal.triangle.html

7.

Escreva um programa que guarda a informação sobre 10 elementos a sua escolha da **tabela periódica de elementos** e que permita a sua consulta. (actualmente esta tabela tem 103 elementos como sabe.)
Para informação e obtenção de dados para o seu exemplo, ver os sites:
http://www.cdcc.sc.usp.br/quimica/tabelaperiodica/tabelaperiodica1.htm
ou também:
http://www.brasil.terravista.pt/Albufeira/1895/

Para fazer, crie um array de registos ("*record")* em Pascal onde vai armazenar toda a informação interessante para cada elemento: Número, Nome, Símbolo, camadas, massa, ponto ebulição, ponto de fusão, nome do(s) descobridor(es), etc. Escolha pelo menos **cinco** informações distintas sobre cada elemento. Lembre-se que cada tipo de dado distinto merece também um **tipo** distinto em Pascal.
Uma vez carregada a tabela, faz uma função que **quando é dado o número do elemento,** a função **devolve a posição** (indice) na matriz onde está situado o elemento. Isto pela sua vez permite a um outro procedimento mostrar os dados armazenados no correspondente registo.

---

soluções

Soluções: Aula prática 10

Parte B.

---

6. Escreva um programa que faz o desenho dum triângulo de Pascal com oito (8) linhas. Como se sabe, este triângulo tem muitas propriedades interessantes para as matemáticas e as estatísticas. Nele encontramos números primos, números triangulares, a serie de Fibonacci, combinatória, etc.

```
{-----------------------------------------------------}
program Tpascal;
{programa que constroi o triângulo de Pascal com 8 linhas. Pode ser adaptado
a qualquer tamanho. P.Serendero }
uses crt;
const max_col = 15;
      max_lin = 8;
var centro: word;
   a : array[1..max_lin, 1..max_col] of word;
{-------------------------------------------------------------------}
procedure limpar_array;
{procedimento que inicializa todo o arranjo com zeros }
var i,j : word;
begin
   for  i:= 1 to max_lin do
        for j:= 1 to max_col do
            a[i, j]:= 0;
end;
{-----------------------------------------------------------------------}
function SomaDiag(lin, col: word):word;
{esta função calcula o valor a inserir numa posição, que corresponde a
soma dos valores existentes nas diagonais da linha precedente }
begin
   if (col+1 > max_col) then
      SomaDiag:= a[lin-1, col-1]
   else
      SomaDiag:= a[lin-1, col-1] + a[lin-1, col+1];
end;
{-------------------------------------------------------------------}
procedure imprimir;
{imprime o triangulo de Pascal. Cada posiçãoo que tem 0 , impressa como
um espaço livre. }
var lin, col, i : word;
begin
  clrscr;
  for i:= 1 to 3 do writeln;

  for lin:= 1 to max_lin do
     begin
```

```
        write('            ');
        for col:= 1 to max_col do
           if (a[lin, col] = 0) then
                write('  ')
           else write(a[lin, col]);
        writeln;
     end;
end;
{-----------------------------------------------------------------}
function centro_linha: word;
{inserir um 1 no centro do array correspondente a primeira linha }
begin
    centro_linha:= (max_col div 2) + 1;
end;
{-----------------------------------------------------------------}
procedure constroi_triangulo(centro: integer);
{sen optimiza‡äes. Verifica todas as posições para construção do triângulo}
var lin, col, inicio : word;
begin
    a[1][centro]:= 1;          {no centro da primeira linha so um 1 }
    for lin:= 2 to max_lin do   {algoritmo para as restantes linhas }
      begin
         centro:= centro - 1;
         for col:= centro to max_col do
            a[lin, col]:= SomaDiag(lin, col);
      end;

end;
{-----------------------------------------------------------------}
begin
     limpar_array;
     centro:= centro_linha;
     constroi_triangulo(centro);
     imprimir;
     readln;
end.
```

7.

Escreva um programa que guarda a informação sobre 10 elementos a sua escolha da **tabela periódica de elementos** e que permita a sua consulta. (actualmente esta tabela tem 103 elementos como sabe.)
Para fazer, crie um array de registos (“*record”)* em Pascal onde vai armazenar toda a informação interessante para cada elemento: Número, Nome, Símbolo, camadas, massa, ponto ebulição, ponto de fusão, nome do(s) descobridor(es), etc. Escolha pelo menos **cinco** informações distintas sobre cada elemento. Lembre-se que cada tipo de dado distinto merece também um **tipo** distinto em Pascal.
Uma vez carregada a tabela, faz uma função que **quando é dado o número do elemento,** a função **devolve a posição** (indice) na matriz onde está situado o elemento. Isto pela sua vez permite a um outro procedimento mostrar os dados armazenados no correspondente registo.

```
{-----------------------------------------------------------------------}
program tabelap;
Uses Crt;
```

```
type elemento = record
              numero     : integer;
              nome       : string[25];
              simbolo    : string[5];
              camadas    : real;
              massa      : real;
            end;

var  tabela:  array[1..10] of elemento;
    i : word; r: char;
{---------------------------------------}
procedure procura_mostra(n: integer);
begin
    writeln('      Elemento seleccionado: ');
    writeln('numero : ',tabela[n].numero);
    writeln('nome   : ',tabela[n].nome);
    writeln('simbolo: ',tabela[n].simbolo);
    writeln('camadas: ',tabela[n].camadas:0:4);
    writeln('massa  : ',tabela[n].massa:0:4);
    readln;
end;
{-------------------------------------------------------------}
procedure carrega_elemento(n:integer);
begin
    writeln('Favor entrar os dados dum elemento: ');
    write('numero : '); readln(tabela[n].numero);
    write('nome   : '); readln(tabela[n].nome);
    write('simbolo: '); readln(tabela[n].simbolo);
    write('camadas: '); readln(tabela[n].camadas);
    write('massa  : '); readln(tabela[n].massa);
end;
{-------------------------------------------------------------}
begin
   clrscr;
   for i:=1 to 10 do
      carrega_elemento(i);
   readln;
   clrscr;
   r:= 'x';

   while (r <> 'n') do
     begin
         write('Qual elemento quer ver? '); readln(i);
         procura_mostra(i);
         writeln; write('Deseja continuar? '); readln(r);
     end;
   writeln('vocˆ decidiu terminar. Adeus.');
   readln;
end.
```

Aula prática 11

Once upon a midnight dreary, while I pondered, weak and weary
Over many a quaint and curious volume of forgotten lore,
While I nodded, nearly napping, suddenly there came a tapping,
As of some one gently rapping, rapping at my chamber door.
" 'Tis some visitor," I muttered, "tapping at my chamber door -
Only this, and nothing more."

The Raven, Edgar Allan Poe, 1845.

1a. Neste aula vamos escrever um programa que vai ler um ficheiro. Como exemplo vamos ler um ficheiro que contem o poema *The Raven* de Edgar Allan Poe (um excerto, a primeira parte, fica no início desta página) e vamos contar quantas vezes cada letra aparece no texto.

- Só conta as letras normais ('A' .. 'Z')
- Não distingue entre 'A' e 'a', etc. Usa a função UpCase para converter 'a' para 'A', etc.
- Põe de lado todos os outros caracteres.

Funções e procedimentos úteis (além das funções da aula teórica 21):

`Ord(c)`
: Retorna o "número" do caracter c, ou seja o código ASCII do caracter. Por exemplo: `Ord('A')` é igual a `65`, `Ord('Z')` é igual a `90`. Veja tabela no fim da página.

`Chr(n)`
: É o inverso da função `Ord`. `Chr` retorna o caracter com código ASCII `n`. Por exemplo: `Chr(65)` é igual a `'A'`, `Chr(97)` é igual a `'a'`.

`EoF(f)`
: Retorna TRUE se estamos a ler no fim do ficheiro `f`. `f` é uma variável do tipo `text`. Veja aula 21.

`UpCase(c)`
: Retorna a letra maiúscula do caracter c. Por exemplo: `UpCase('f')` é igual a `'F'`. `UpCase('G')` é igual a `'G'`.

Aqui encontra-se o ficheiro `theraven.txt` com o texto inteiro do *The Raven* de Edgar Allan Poe. Guarda este ficheiro no seu disco (Y:). Carrega o botão direito do rato em cima da palavra azul 'aqui' e depois: Em Microsoft Internet Explorer "Save Target As...". Em Netscape Navigator "Save Link As...".

1b. Muda o programa da 1a de forma que vai guardar a informação no ficheiro 'contas.txt'.

1c. Qual letra aparece o mais frequente? (escreve um programa). Isto segue a regra 'ETAOIN SHRDLU' (em inglês)

ASCII table and description

ASCII stands for American Standard Code for Information Interchange. Computers can only understand numbers, so an ASCII code is the numerical representation of a character such as 'a' or '@' or an action of some sort. ASCII was developed a long time ago and now the non-printing characters are rarely used for their original purpose. Below is the ASCII character table and this includes descriptions of the first 32 non-printing characters. ASCII was actually designed for use with teletypes and so the descriptions are somewhat obscure. If someone says they want your CV however in ASCII format, all this means is they want 'plain' text with no formatting such as tabs, bold or underscoring - the raw format that any computer can understand. This is usually so they can easily import the file into their own applications without issues. Notepad.exe creates ASCII text, or in MS Word you can save a file as 'text only'

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.asciitable.com

---

soluções

Soluções da aula prática 11

1a.

```
PROGRAM EdgarAlanPoe;

Var f: text;
    c: char;
    ci: integer;
    cont: array[65..90] of integer;
    i: integer;

begin
  for i := 65 to 90 do cont[i] := 0;
  Assign(f, 'theraven.txt');
  Reset(f);
  While NOT Eof(f) do
    begin
      Read(f, c);
      c := UpCase(c);
      ci := Ord(c);
      if (ci>=65) AND (ci<=90) then
        cont[ci] := cont[ci] + 1;
    end;
  Close(f);
  for i := 65 to 90 do
    writeln(Chr(i),'  ',cont[i]);
end.
```

output (ecrã)

```
A  339
B  94
C  71
D  194
E  618
F  94
G  122
H  290
I  318
J  2
K  32
L  225
M  158
N  374
O  370
P  95
Q  9
R  336
```

```
S  278
T  437
U  121
V  66
W  79
```

1b.

```
PROGRAM EdgarAlanPoeF;

Var f: text;
    c: char;
    ci: integer;
    cont: array[65..90] of integer;
    i: integer;

begin
  for i := 65 to 90 do cont[i] := 0;
  Assign(f, 'theraven.txt');
  Reset(f);
  While NOT Eof(f) do
    begin
      Read(f, c);
      c := UpCase(c);
      ci := Ord(c);
      if (ci>=65) AND (ci<=90) then
        cont[ci] := cont[ci] + 1;
    end;
  Close(f);
  Assign(f,'contas.txt');
  Rewrite(f);
  for i := 65 to 90 do
    writeln(f, Chr(i), '  ', cont[i]);
  Close(f);
end.
```

1c.

```
PROGRAM EdgarAlanPoeC;

Var f: text;
    c: char;
    ci: integer;
    cont: array[65..90] of integer;
    i: integer;
    max: integer;
    maxi: integer;

begin
  for i := 65 to 90 do cont[i] := 0;
  Assign(f, 'theraven.txt');
```

```
  Reset(f);
  While NOT Eof(f) do
    begin
      Read(f, c);
      c := UpCase(c);
      ci := Ord(c);
      if (ci>=65) AND (ci<=90) then
        cont[ci] := cont[ci] + 1;
    end;
  Close(f);
  max := 0;
  for i := 65 to 90 do
    if cont[i]>max then
       begin
         max := cont[i];
         maxi := i;
       end;
  WriteLn('Max: ', Chr(maxi), '  ', cont[maxi]);
end.
```

output (ecrã)

```
Max: E  618
```

Aula prática 12

---

1. Faz um programa que determine quanto tempo leva dobrar o capital numa conta do banco. O utilizador deve dar os dados relevantes (por exemplo a taxa de juros).

---

2. Declare um array de 1000 elementos. Enche o array com números aleatórios e escreve o código para ordenar o array com o segundo algoritmo da aula teórica 22.

---

3. O programa que se segue deveria calcular o factorial e o somatório de um número introduzido pelo utilizador, no entanto tem alguns erros. Assinale e corrija os erros do programa para que realize o que é pretendido.

```
Program Factorial;

Var num, somatorio, factorial: integer;

begin
  Writeln('Indique um numero inteiro');
  ReadLn(num);
  while (num>0) do
    begin
      factorial := factorial * num;
      somatorio := somatorio + num;
    end;
  writeln('Factorial ', factorial, 'Somatorio ',
          factorial);
end.
```

---

4. Define um novo tipo de variável para guardar um coordenado ou vector (x, y, z). Depois escreve uma função que recebe um coordenado e devolve o comprimento do vector (a distância até o origem).

---

soluções

Soluções da aula prática 12

1.

```
PROGRAM DobrarCapital;

Var cap, juros: real;
    ano: integer;

begin
  Writeln('Taxa de juros (%): ');
  ReadLn(juros);
  ano := 0;
  repeat
    cap := cap + cap*(juros/100.0);
    ano := ano+1;
  until cap>=2.0;
  writeln('numero de anos: ', ano);
end.
```

2.

```
PROGRAM Ordenar1000;

Const N = 1000;
Var ra: array[1..N] of real;
    i, j: integer;
    min: real;
    jmin: integer;
    temp: real;

begin
  for i := 1 to N do
    ra[i] := Random;
  for i := 1 to N-1 do
    begin
      min := 2.0; { com certeza maior que numero maximo do array }
      jmin := i;
        { procura minimo no resto do array }
      for j := i to N do
        if ra[j]<min then
          begin
            min := ra[j];
            jmin := j;
          end;
        { troca o numero minimo com ra[i] }
      temp := ra[i];
```

```
      ra[i] := ra[jmin];
      ra[jmin] := temp;
    end;
   { mostrar resultado }
  for i := 1 to N do
    writeln(i:4,'  ',ra[i]:0:6);
  readln;
end.
```

3.

```
Program Factorial;

Var num, somatorio, factorial: integer;

begin
  Writeln('Indique um numero inteiro');
  ReadLn(num);
  factorial := 1;
  somatorio := 0;
  while (num>0) do
    begin
      factorial := factorial * num;
      somatorio := somatorio + num;
      num := num - 1;
    end;
  writeln('Factorial ', factorial, 'Somatorio ',
          somatorio);
end.
```

4.

```
PROGRAM Distance;

type coordenate = record
  x, y, z: real;
end;

FUNCTION VectorLength(co: coordenate): real;
begin
  VectorLength := Sqrt(Sqr(co.x) + Sqr(co.y) + Sqr(co.z));
end;

begin
  { codigo principal }
end.
```
