– Apresentação –

Funcionamento da Disciplina

- A avaliação é feita por um exame (16 valores):
    - Abrangendo toda a matéria lecionada;
    - Sem consulta;
    - Com uma parte teórica e outra (maior) de cariz prático,

  e por três testes (4 valores):
    - Sem mínimos e com um aviso de uma semana de antecedência;
    - Sem consulta;
    - 1º teste – condições (1 valor);
    - 2º teste – Ciclos e funções (1.5 valores);
    - 3º teste – Vetores e strings (1.5 valores),
    - A não realização de algum dos testes implicará a nota zero nesse teste.
- Têm que assistir a, no mínimo, 6 aulas práticas para poderem ter acesso a exame;
- As justificações de faltas às aulas deverão ser apresentadas ao docente da respectiva turma, no período máximo de 10 dias úteis após a ocorrência.

Objectivos da Disciplina

- Representação algorítmica da resolução de problemas;
- Implementação da estratégia de resolução em linguagem C.

Programa Prática

- Resolução de problemas;
- Representação algorítmica de estratégias de resolução;
- Tipos de dados, operadores, expressões;
- Instruções de selecção;
- Instruções de repetição;
- Funções;
- Arrays;
- Strings;
- Arrays bidimensionais.

Recursos e informações

- Na plataforma Moodle encontram as fichas de exercício (que devem trazer para as aulas práticas), os horários de atendimento, ficha da disciplina (onde estão as linhas gerais de funcionamento da disciplina) e apontamentos de apoio às aulas teóricas e práticas.

– Noções básicas de algoritmia –

Apresentar e explicar a matéria contida nos slides do ficheiro "IP_Algoritmia.pptx", cujo resumo é:

Resolução de Problemas

- **Antes de programar, deve-se PENSAR**;
- Todos os problemas podem enunciar-se em termos de um estado actual (aquilo conhecemos) e um estado final (aquilo que se quer obter);
- Resolver um problema implica:
  - Reconhecer aquilo que é relevante no estado actual;
  - Perceber qual é o estado final;
  - Encontrar a melhor estratégia que permita ir do estado inicial até ao final;
  - Implementar a estratégia escolhida (traduzi-la para um conjunto de instruções exactas passíveis de serem executadas sequencialmente);
  - Executar as instruções;
  - Avaliar a solução.

Tipos de Dados

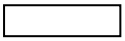
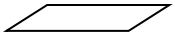
- Os problemas a resolver lidam com dados variáveis que podem ser:
  - Inteiros (positivos, negativos ou sem indicação);
  - Reais (positivos, negativos ou sem indicação);
  - Caracteres.
- Há ainda dados que são sempre constantes (têm definição especial na linguagem C);
- É sempre preciso definir os tipos de dados a usar.

Especificação ou análise do Problema

- **Dados de entrada**: Onde se define o estado inicial do problema (o que se conhece do problema ou o que se tem que pedir ao utilizador). Nesta fase deve-se atribuir nomes a cara informação a usar (chamadas de **variáveis**), definir o seu respectivo tipo e fazer uma breve descrição para cada uma delas (por exemplo, Nome variável (tipo) – Descrição);
- **Resultados pretendidos**: Onde se define o estado final do problema (quais os resultados que devem ser obtidos ou mostrados ao utilizador). Deve-se seguir o que é feito fase anterior para cada um dos resultados;

- **Processamento requerido**: Onde se define a estratégia, expressa em linguagem natural ou através de fórmulas.

Desenvolvimento da Solução

- **Fluxograma** (Flowchart): Usam-se determinados símbolos para tarefas específicas. Por exemplo:
  - Linhas de Ligação 
  - Início / Fim 
  - Acções/Processamento 
  - Entrada/Saída 
- **Pseudocódigo**: Usa-se linguagem natural para definir tarefas específicas. Por exemplo:
  - Definição de bloco de instruções:

    INÍCIO "nome do bloco"
    - Instrução 1;
    - Instrução 2;
    - …

    FIM " nome do bloco"
  - Entrada/Saída:

    OBTEM (<variável>)
    MOSTRA (<variável>)

– Processo de criação de uma aplicação –

Demonstração de todo o processo

Exemplo 1

Escreva "Ola mundo" no ecrã. Deverá fazer a especificação do problema, o pseudocódigo, o fluxograma e a codificação do exercício em linguagem C.

Especificação ou análise do problema:

Dados de entrada:

Não existem

Resultados pretendidos:

Mostrar a frase "Ola mundo" no ecrã

Processamento requerido:

Escrever no ecrã

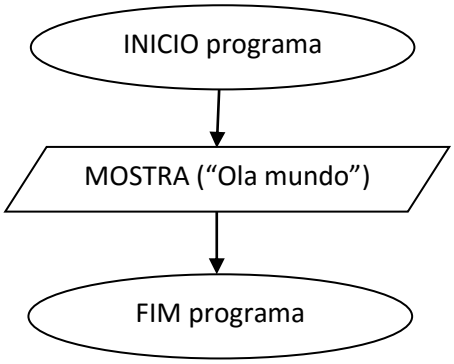
Pseudocodigo:

INÍCIO programa

MOSTRA ("Olá mundo")

FIM programa

Fluxograma:



Codificação em C:

```
#include <stdio.h>

void main()
{
      printf("Ola mundo\n");
}
```

Exemplo 2

Conhecendo as duas notas que um aluno teve nos testes, pretende-se calcular e mostrar no ecrã a média final a ter por ele na disciplina. Deverá fazer a especificação do problema, o pseudocódigo, o fluxograma e a codificação do exercício em linguagem C.

Especificação ou análise do problema:

Dados de entrada:

nota1 (real) – 1ª nota

nota2 (real) – 2ª nota

Resultados pretendidos:

media (real) – Média das duas notas

Processamento requerido:

Obter as duas notas

media = $\frac{nota1+nota2}{2}$

Mostrar a média final

Pseudocodigo:

INÍCIO programa

OBTEM (nota1)

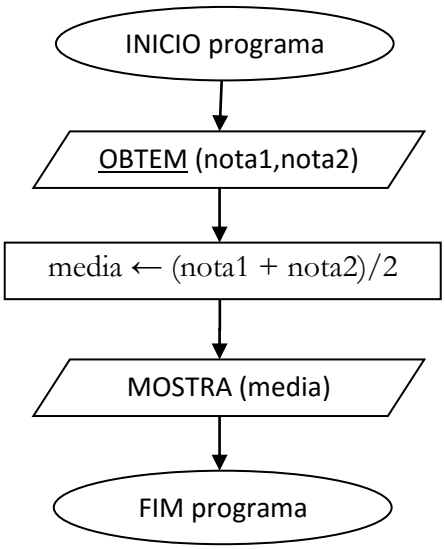
OBTEM (nota2)

media ← (nota1 + nota2)/2

MOSTRA (media)

FIM programa

Fluxograma:



Codificação em C:

```c
#include <stdio.h>

void main()
{
      float nota1,nota2,media;

      printf("Introduza a 1a nota\n");
      scanf("%f",&nota1);
      printf("Introduza a 2a nota\n");
      scanf("%f",&nota2);
      media=(nota1+nota2)/2;
      printf("Media: %.2f\n\n",media);
}
```

– Noções básicas de algoritmia –

Resolução de exercícios

Exercício 3 – Ficha 1

Elabore um algoritmo que calcule a área e o perímetro de um círculo. Deverá fazer a especificação do problema, o pseudo-código e o fluxograma.

Especificação ou análise do problema:

Dados de entrada:

R (real) – Raio do círculo

PI (real) – Valor de π

Resultados Pretendidos:

A (real) – Área do circulo

P (real) – Perímetro do circulo

Processamento requerido:

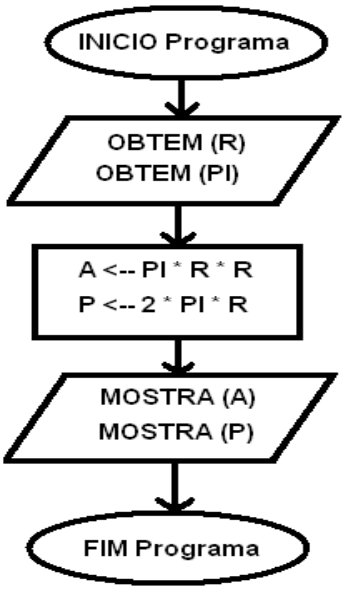
Obter R e PI

A = $PI \times R^2$

P = $2 \times PI \times R$

Mostrar A e P

Fluxograma:



Pseudocodigo:

INÍCIO Programa

OBTEM (R)

OBTEM (PI)

A ← PI . R . R

P ← 2 . PI . R

MOSTRA (A)

MOSTRA (P)

FIM Programa

Exercício 6 – Ficha 1

Escreva um algoritmo para calcular o número de eleitores (votantes) de um município, tendo por base o número de votos brancos, nulos e válidos. Deve também calcular e escrever a percentagem que cada tipo de voto representa em relação ao total de votantes.

Especificação ou análise do problema:

Dados de entrada:

nVotosB (inteiro) – Número de votos brancos

nVotosN (inteiro) – Número de votos nulos

nVotosV (inteiro) – Número de votos válidos

Resultados pretendidos:

nEleitores (inteiro) – Número de eleitores

pVotosB (real) – Percentagem de votos brancos

percentVotosN (real) – Percentagem de votos nulos

pVotosV (real) – Percentagem de votos válidos

Processamento requerido:

Obter nVotosB, nVotosN e nVotosV

nEleitores ← nVotosB + nVotosN + nVotosV

pVotosB ← $100 \times \frac{nVotosB}{nEleitores}$

percentVotosN ← $100 \times \frac{nVotosN}{nEleitores}$

pVotosV ← $100 \times \frac{nVotosV}{nEleitores}$

Mostrar pVotosB, percentVotosN e pVotosV

Pseudocodigo:

INÍCIO programa

OBTEM (nVotosB)

OBTEM (nVotosN)

OBTEM (nVotosV)

nEleitores ← nVotosB + nVotosN + nVotosV

pVotosB ← 100 . nVotosB/nEleitores

percentVotosN ← 100 . nVotosN/nEleitores

pVotosV ← 100 . nVotosV/nEleitores

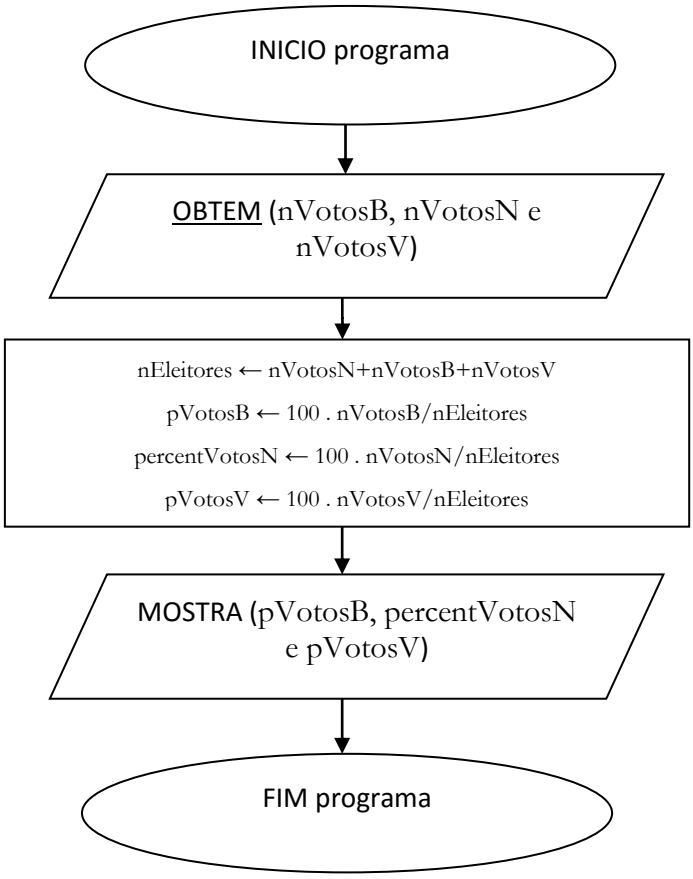
MOSTRA (pVotosB)

MOSTRA (percentVotosN)

MOSTRA (pVotosV)

FIM programa

Fluxograma:



Codificação em C:

```c
#include <stdio.h>

void main()
{
	int   nEleitores,nVotosB,nVotosN,nVotosV;
	float pVotosB,percentVotosN,pVotosV;

	printf("Introduza o numero de votos brancos: ");
	scanf("%d",&nVotosB);
	printf("Introduza o numero de votos nulos: ");
	scanf("%d",&nVotosN);
	printf("Introduza o numero de votos validos: ");
	scanf("%d",&nVotosV);
	nEleitores=nVotosB+nVotosN+nVotosV;
	pVotosB=100*nVotosB/nEleitores;
	percentVotosN=100*nVotosN/nEleitores;
	pVotosV=100*nVotosV/nEleitores;
	printf("\n\nPercentagem de votos brancos: %.2f%%\n",pVotosB);
	printf("Percentagem de votos nulos: %.2f%%\n",percentVotosN);
	printf("Percentagem de votos validos: %.2f%%\n\n",pVotosV);
}
```

Exercício 7 – Ficha 1

Um motorista de táxi deseja calcular o rendimento do seu carro. Considerando que o preço do combustível é de 1,3 €/litro, escreva um algoritmo para ler: a marcação do conta-quilómetros (Km) no início do dia, a marcação (Km) no final do dia, o número de litros de combustível gasto e o valor total (em €) recebido dos passageiros. Deve ainda calcular e escrever a média do consumo em Km/l e o lucro diário.

Especificação ou análise do problema:

Dados de entrada:

kmi (real) – Quilómetros marcados no início do dia (em km)

kmf (real) – Quilómetros marcados no final do dia (em km)

cg (real) – Combustível gasto (em litros)

valor (real) – Valor em dinheiro recebido dos passageiros (em €)

PC (CONSTANTE) – Preço do combustível (em €/litro)

Resultados Pretendidos:

c (real) – Média do consumo (em km/litro)

l (real) – Lucro líquido (em €)

r (real) – Rendimento do carro

Processamento requerido:

Obter kmi, kmf, cg e valor

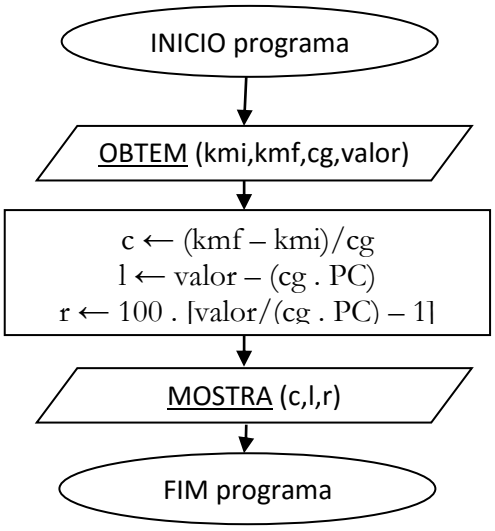
$c = \frac{(kmf - kmi)}{cg}$

$l = valor - (cg \times PC)$

$r = 100 \times \left[\frac{valor}{cg \times PC} - 1\right]$

Mostrar c, l e r

Fluxograma:



Pseudocodigo:

INÍCIO Programa

OBTEM (kmi, kmf, cg, valor)

c ← (kmf – kmi)/cg

l ← valor – (cg . PC)

r ← 100 . [valor/(cg . PC) – 1]

MOSTRA (c)

MOSTRA (l)

MOSTRA (r)

FIM Programa

– Declaração de variáveis na linguagem C –

A declaração das variáveis faz-se da seguinte maneira:

Tipo Nome valor a atribuir (nem sempre é necessário)

- O Nome deve ser descritivo. Costumam-se usar letras minúsculas (as maiúsculas são usadas para os nomes das constantes), mas isto não é obrigatório que seja assim;
- As variáveis a usar na linguagem C podem ser do Tipo:

Tipo	Observação (o que ocupa correndo o código abaixo no Code::Blocks)
Char	Um único carácter (ocupa 1 byte)
short int	Um inteiro curto (ocupa 2 byte)
unsigned short int	Um inteiro curto sem sinal (ocupa 2 byte)
Int	Um inteiro (ocupa 4 bytes)
unsigned int	Um inteiro sem sinal (ocupa 4 bytes)
long int	Um inteiro longo (ocupa 4 bytes)
Float	Um real (ocupa 4 bytes)
Double	Um real longo - igual a long float (ocupa 8 bytes)

A informação apresentada na coluna observação, na tabela acima, sobre os tamanhos de cada um dos diferentes tipos de dados não é definitiva, pois estes tamanhos podem variar conforme a implementação da máquina onde a aplicação é executa (ver http://www.lix.polytechnique.fr/~liberti/public/computing/prog/c/C/CONCEPT/data_types.html#int e http://en.wikipedia.org/wiki/C_data_types). Existe sempre a seguinte relação:

char (1 byte) <= short (2 bytes) <= int <= long (4 bytes) <= float < double

Exemplo:

```
int    x = 13;        x fica com 13
float  n;
```

Código de verificação do tamanho dos diferentes tipos de dados

```
#include <stdio.h>
#include <stdlib.h>

void main(void)
{
      char              c;
      short             s;
      unsigned short    us;
      int               i;
      unsigned          ui;
      long              l;
      float             f;
      double            d;

      printf("Tamanho de cada tipo de dados:\n");
      printf("\tCharacter (char) \t\t\t%d (bytes)\n", sizeof(c));
      printf("\tShort integer (short) \t\t\t%d (bytes)\n", sizeof(s));
      printf("\tUnsigned Short integer (Unsigned short) %d (bytes)\n",
sizeof(us));
      printf("\tInteger (int) \t\t\t\t%d (bytes)\n", sizeof(i));
      printf("\tUnsigned integer (Unsigned) \t\t%d (bytes)\n", sizeof(ui));
      printf("\tLong integer (long) \t\t\t%d (bytes)\n", sizeof(l));
      printf("\tFloat (float) \t\t\t\t%d (bytes)\n", sizeof(f));
      printf("\tDouble (double) \t\t\t%d (bytes)\n\n", sizeof(d));
}
```

– Operadores específicos da linguagem C –

Operadores aritméticos comuns (+, -, *)

Fazem as operações comuns de soma, subtracção e multiplicação.

Operadores aritméticos de resto e de divisão (%, /)

O operador de resto (%) só funciona com inteiros. O operador divisão, dependendo do tipo dos operandos, faz uma divisão inteira ou real (ver exemplo abaixo).

Exemplo:

```
int   n = 41 % 14;        n fica com 13
float x = 13;
float a = x / 2;          a fica com 6.5
float b = n / 2;          b fica com 6.0
```

Operadores de Incremento/Decremento unário (++i; i++; --i; i--)

Tornam a escrita sucinta. Deve-se ter atenção ao momento de execução do operador.

Exemplo:

```
int i=6;
int j=i++;          j fica com 6, i fica com 7
int k=++i;          k fica com 8, i fica com 8
```

Operadores de atribuição composta (+=; -=; *=; /=; %=)

Usados quando o destino é o primeiro operando. Também tornam a escrita mais sucinta.

Exemplo:

```
int N=12;
N/=2;               N fica com 6
N%= 3;              N fica com 0
```

Operadores de conversão de tipo (operador casting - ())

Altera o tipo de um dado valor. O resultado pode ser diferente consoante os tipos.

Exemplo:

```
int x = 3.5*2;      x fica com 7
int y = (int)3.5 * 2;  y fica com 6
```

Operadores lógicos (&&; | |; !)

Funcionam como a lógica booleana. Em C, o valor de zero é considerado como Falso e todos os outros valores diferentes de zero são considerados como verdadeiro. Como resultado da avaliação de uma expressão lógica apenas é devolvido 0 ou 1. O lado direito de uma expressão só é analisado se o resultado não puder ser antecipado.

Exemplo:

```
int a = 1;
int b = -1;
int c = 0;
int d = (a && b) || c;          d fica com 1
int d = (a && b) || ++c;    d fica com 1; nenhuma
                                variável muda de valor.
```

Operadores relacionais (==; !=; >; >=; <; <=)

É importante não confundir comparação == com a atribuição.

Precedência e associatividade de operadores em C

Nem sempre é evidente que operação se deve fazer primeiro. Assim, introduz-se os conceitos:

- **Precedência**
  Que tipo de operações são feitas primeiro.

  Exemplo:
  X=5 + 3 * 6; Dá 48 ou 23? Claro que dá 23.

- **Associatividade**
  Entre operações com a mesma precedência, "por que lado" se deve começar a fazer as operações? Para remover eventuais ambiguidades pode-se consultar a tabela de abaixo.

Precedence	*Name*	*Symbol(s)*	*Associativity*
1	array subscripting	`[]`	left
1	function call	`()`	left
1	structure and union member	`. ->`	left
1	increment (postfix)	`++`	left
1	decrement (postfix)	`--`	left
2	increment (prefix)	`++`	right
2	decrement (prefix)	`--`	right
2	address of	`&`	right
2	indirection	`*`	right
2	unary plus	`+`	right
2	unary minus	`-`	right
2	bitwise complement	`~`	right
2	logical negation	`!`	right
2	size	`sizeof`	right
3	cast	`()`	right
4	multiplicative	`* / %`	left
5	additive	`+ -`	left
6	bitwise shift	`<< >>`	left
7	relational	`< > <= >=`	left
8	equality	`== !=`	left
9	bitwise *and*	`&`	left
10	bitwise exclusive *or*	`^`	left
11	bitwise inclusive *or*	`|`	left
12	logical *and*	`&&`	left
13	logical *or*	`||`	left
14	conditional	`?:`	right

15	assignment	`= *= /= %=`
		`+= -= <<= >>=`
		`&= ^= |=`
			right

16	comma	`,`	left

Exercício 12 – Ficha 2

Indique o resultado da avaliação das seguintes expressões:

i) 4 > 8 || 3 <= 5 && 7 != 9

ii) 4 / 3 + 2 * 18 / 4

iii) (2 == 1) || (2 == 2)

iv) 2 == (1 || 2) == 2

v) 2 == 1 || 2 == 2

i) 4 > 8 || 3 <= 5 && 7 != 9

0 1 1

&& 1

|| 1

1

(4 > 8) || ((3<=5) && (7!=9)) → 0 || (1 && 1) → 0 || 1 → 1

ii) 4 / 3 + 2 * 18 / 4

1 + 36 / 4

\+ 9

10

(4 / 3) + ((2*18)/4) → 1 + (36/4) → 1+9 → 10

iii) (2 == 1) || (2 == 2)

0 || 1

1

(2==1) || (2==2) → 0 || 1 → 1

iv) 2 == (1 || 2) == 2

2 == 1

0 == 2

0

(2==(1 || 2))==2 → (2==1)==2 → 0 == 2 → 0

v) 2 == 1 || 2 == 2

0 || 1

1

(2==1) || (2==2) → 0 || 1 → 1

Exercício 13 – Ficha 2

Converta as expressões para instruções em linguagem C.

a) $x = a + \dfrac{b}{c + \dfrac{d+e}{f + \dfrac{g}{h}}}$ *b*) $x = \dfrac{\dfrac{a}{b} - 1}{c \times (\dfrac{c}{d} + 1)} + a$

a)

```
x = a + b/ (c + (d + e)/(f + g/h));
```

b)

```
x = ((a/b – 1) / (c * (c/d + 1)))  + a;
```

– Escrita formatada –

Função de escrita formatada printf(…)

- Permite ao programador controlar a aparência da saída de dados;
- Mostra no ecrã a informação que está entre aspas;
- Pode conter caracteres vulgares ou especificações de conversão (começam por **%**);
  - Esta especificação de conversão vai especificar como o valor vai ser convertido do seu formato interno (binário), para o formato de impressão (caracteres);
  - Por exemplo, **%d** indica que a função **printf(…)** vai converter um valor do tipo int num conjunto de caracteres correspondentes aos dígitos decimais.
- Os parâmetros desta função podem ser definidos da seguinte forma:
  int printf(const char *formato, arg1, arg2, …);
  - Número de caracteres mostrados.
  - Tipos dos objectos a escrever (especificações de conversão).
  - Valores que vão aparecer nos locais indicados pelas especificações de conversão.

Formatos de Leitura / Escrita (Resumo)

Tipo	Formato	Observações
char	%c	Um único caracter
int	%d ou %i	Um inteiro (base decimal)
int	%o	Um inteiro (base octal)
int	%x ou %X	Um inteiro (base hexadecimal)
short int	%hd	Um inteiro *short* (base decimal)
long int	%ld	Um inteiro *long* (base decimal)
unsigned short int	%hu	Um inteiro *short* positivo
unsigned int	%u	Um inteiro positivo
unsigned long int	%lu	Um inteiro *long* positivo
float	%f ou %e ou %E ou %G	
double	%lf ou %le ou ...	(double=long float)

Exemplo:	**Mostra:**

`int a=8;`
`int b=12;`
`printf("Numeros: %d %d",a,b);`
	Numeros: 8 12

`printf("%c %d %f %e %s\n",'a',2,3.14,56.,"oito");`	a 2 3.140000 5.600000e+001 oito

- **%–n.mFormato**
  - Caso o valor a mostrar use menos caracteres que os definidos como mínimo, justifica à esquerda. Sem o carácter – justifica à direita.
  - Número mínimo para o total de caracteres. Se necessitar de mais, expande-se
  - Se Formato for:
    - d → m é o número mínimo de dígitos a mostrar. São adicionados zeros ao que se mostra, caso necessário. Se m for omisso, assume-se como 1;
    - e/f → m é o número de dígitos a mostrar após o ponto decimal (por defeito usa 6). Se m for zero a parte decimal não é mostrada.

Exemplo: `printf("%8.4d\n%8.4d\n%8.4d\n",12,120,12000);`	0012 0120 12000
`printf("%8.4d\n%-8.4d\n%8.4d\n",12,120,12000);`	0012 0120 12000
`printf("%8.4f\n%8.4f\n%8.4f\n",12.0,120.0,12000.0);`	12.0000 120.0000 12000.0000

Escrita de um carácter

- Internamente os caracteres são representados como inteiros. O valor de um carácter como inteiro é o seu código ASCII. É possível ver o valor de um carácter imprimindo-o como inteiro.

Exemplo:
`char c='A';`
`int i=97;`
`printf("c=%c  i=%d\n",c,c);`
`printf("c=%c  i=%d\n",i,i);`
	c=A i=65
	c=a i=97

- A função **int putchar(int valor)** escreve um único caracter. Aceita o caracter entre pelicas ou o seu código. Devolve o código do que escreveu ou EOF em caso de erro.

Exemplo:
`int j;`
`putchar(1);`
`putchar('\n');`
`j=putchar('1');`
`printf("\n%d\n",j);`
	Mostra
	☺
	1
	49

Escrita de um conjunto de caracteres (string)

- A função **int puts(const char *buffer)** escreve o texto contido na variável buffer e muda de linha. Devolve EOF se ocorreu algum erro, ou então um número não negativo se escreveu bem;
- A função **int printf(…)** também escreve o texto, desde que o formato seja do tipo s. Este tipo designa-se por **string** e será apresentado numa matéria mais à frente.

Exemplo:
`int  j;`
`puts("Ola");`
`j=puts("Tudo bem?");`
`printf("%d",j);`
	Mostra:
	Ola
	Tudo bem?
	0

`char frase[10]= "Tudo bem!";`
`printf("%s","Escrita formatada");`
`printf("Frase: %s",frase);`
	Escrita formatada
	Frase: Tudo bem!

– Leitura Formatada –

Função de escrita formatada scanf(…)

- Esta função permite ao programador ler os dados de acordo com um determinado formato;
- Os parâmetros desta função podem ser definidos da seguinte forma:
  int scanf(const char *formato, arg1, arg2, …);
  - Número de especificações de formado lidas com sucesso ou EOF em caso de erro.
  - Tipos dos dados a ler. Em valores numéricos só conseguimos controlar o tipo de dados que se pretende ler, e não o número de dígitos ou caracteres.
  - Onde se guardam os valores lidos. Necessitam quase sempre de ser precedidas pelo caracter &.

Exemplo:	**Introduz:**	
`int a;`		a fica com 3
`float b;`	3	a fica com 8
`scanf("%d",&a);`	8 12.4	b fica com 12.400000
`scanf("%d %f",&a,&b);`		

- Colocar um ou mais espaços em branco na string de formato é indiferente, pois na obtenção dos dados esse(s) espaço(s) pode corresponder a um ou mais espaços em branco. É no entanto, caso exista, separar os dados com pelo menos um espaço;
- Quando existem caracteres vulgares dentro da string de formato, a função **scanf(…)** compara o caracter vulgar com que vem dos dados. Se os dois são iguais, descarta esse caracter dos dados e continua a processar a string de formato e os dados;

Exemplo:	**Introduz:**	
`int dia,mes;`		`dia fica com 8`
`float x,y;`		`mes fica com 12`
`scanf("%d/%d",&dia,&mes);`	`8/12`	`x fica com 0.000000`
`scanf("(%f,%f)",&x,&y);`	`(0, -1.3)`	`y fica com -1.300000`

- A função **scanf(…)** termina o processo de leitura (i.e. não guarda mais dados) quando encontra um dado que não pode pertencer ao formato escolhido;
- Quando a leitura falha permanece no *buffer* de entrada o que não foi "consumido". Usa-se a função **fflush(stdin)** para limpar esse *buffer*;

Exemplo:	**Introduz:**	
`int a=10,b=20;`		
`float x=5,y=15;`	Limpa *buffer*	a fica com 12
`fflush(stdin);`	12 .23 7.7 40	b fica com 20
`scanf("%d %d %f",&a,&b,&x);`	Falha para o b e pára	x fica com 5.000000
`scanf("%f",&y);`	Recupera valores do *buffer*	y fica com 0.230000

- Cabe ao programador verificar se o número de especificações de conversão coincide com o número de variáveis de entrada e se o formato está de acordo com a respectiva variável;

- Existem mais algumas formas de obter dados através da função **scanf(…)**, por exemplo, quando o tipo de dados dentro da string de formato for **c** ou **s** (obtenção de um ou mais caracteres);

Exemplo:	**Introduz:**	

`char c,str[60];`
`scanf("%c",&c);`
`fflush(stdin);`
`scanf("%s",str);`
`fflush(stdin);`
`scanf("%[aeiou]",str);`
`fflush(stdin);`
`scanf("%[aeiou]",str);`
	`W`
	
	`awiu`
	
	`awiu`
	
	`uae`
		`c fica W`
		
		`str fica com awiu\0`
		
		`str fica com a\0`
		
		`str fica com uae\0`

Leitura de um caracter

- A função **int getchar()** lê um único caracter da entrada standard;
- Tem sempre que se premir a tecla 'ENTER';
- Devolve o que leu ou EOF em caso de erro;
- Se for introduzido mais do que um caracter, lê apenas o primeiro;
- Guarda o resto do que foi introduzido para eventuais pedidos de entrada.

Exemplo:	**Mostra:**

`#include <stdio.h>`
`void main()`
`{`
`int c;`
`c=getchar();`
`putchar('c');`
`putchar('\n');`
`putchar(c);`
`putchar('\n');`
`}`
	X (se introduzir um X)
	c
	X

Leitura de uma string

A função **char *gets(char *buffer)** lê um conjunto de caracteres de uma só vez;

- Tem sempre que se premir a tecla 'ENTER';
- Pode dar erro se o número de caracteres que for introduzido for maior do que a capacidade da variável que os vai guardar;
- Devolve a variável com o que foi escrito se o processo foi bem-sucedido ou NULL se ocorreu algum erro.

Exemplo:	

`#include <stdio.h>`
`void main()`
`{`
`char string[10];`
 	
`gets(string);`
`}`
	OLA (enter)
	String fica com OLA\0

Exercício 15 – Ficha 2

Indique qual a saída produzida pelo seguinte programa:

```
#include <stdio.h>

void main(void)
{
   int     i = 10;
   float   f = 4.354;

   printf("%f\n", f);
   printf("%4.2f\n", f);
   printf("%0.4f\n", f);
   printf("%0.0f\n", f);
   printf("%4.2e\n", f);
   printf("%d\n", i);
   printf("%4d\n", i);
   printf("%-4d\n", i);
   printf("%4.2d\n", i);
   printf("%.4d\n", i);
}
```

```
4.354000
4.35
4.3540
4
4.35e+000
10
_ _10
10_ _
_ _ 10
0010
```

Exercício 17 – Ficha 2

Considere a seguinte instrução de entrada de dados:

```
scanf("%f%d%f%d", &x, &i, &y, &j);
```

Assumindo que estão declaradas as variáveis inteiras i,j e reais x,y, qual será o seu valor após a chamada desta instrução se o utilizador escrever a seguinte informação:

```
12 12.4 35.2 3
```

E se a instrução de leitura for a seguinte?

```
scanf("%d%d%f%f", &i, &j, &x, &y);
```

1º scanf: x=12.000000; i=12; y=0.400000; j=35;

2º scanf: i=12; j=12; x=0.400000; y=35.200000;

Ficheiro de código fonte e ficheiro executável

```
#include <stdio.h>

int main()
{
      printf("Ola\n");
      return 0;
}
```

```
01001010100101010010100
10010101010010010101010
00010101010100101010100
10101111010111001
```

Um programa em C é um ficheiro de texto normal. Normalmente o seu nome tem extensão .c. É conhecido como código fonte. É um ficheiro facilmente legível por humanos, e pode ser feito com um qualquer editor de texto.

O programa executado pelo computador é uma sequência de 0 e 1. Normalmente o nome tem extensão exe. É conhecido como ficheiro executável. É um ficheiro ilegível por humanos e é obtido através de um processo de compilação e linkagem.

Compilação

Processo de verificação da existência de erros de sintaxe no código fonte. Caso eles sejam encontrados, terá que se voltar ao processo de edição do código fonte para corrigir os erros. Também detecta situações que não são erro, mas que levantam suspeitas (warnings).

Linkagem

Processo de ligação dos códigos compilados de cada uma das instruções do código fonte. Esse código está nos ficheiros .LIB das respectivas bibliotecas. Só se não existirem erros no código fonte é que este processo pode ser feito.

Características de um programa desenvolvido em linguagem C

; – Marca o fim de uma instrução. Se não for colocado em cada uma das instruções, o compilador avisará que o programa tem erros de sintaxe.

`#include <nome.h>` – Indica ao pré-processador para substituir a linha em que se encontra esta instrução pelo conteúdo do ficheiro **`nome.h`**. Na prática permite o uso de novas funções (**`#include <stdio.h>`**, por exemplo, permite a leitura do teclado e a escrita no monitor.

`#define nome` – Permite definir constantes ou macros. Não precisa de levar **;**.

`void main()` – É a função principal do programa, por onde o programa começa a ser executado. Todas as aplicações C apenas podem ter uma função com este nome.

{ } – Marcam o inicio e o fim de um bloco de instruções.

return (vars ou valores) – usa-se quando se tem que retornar alguma coisa. Por exemplo, se em vez de **`void`** estivesse **`int`** a função principal retornaria um inteiro quando terminasse a sua execução.

// e /* Comentário */ – Permite comentar, respectivamente, uma ou várias linhas de código.

Exercício 11 – Ficha 2

Para cada um dos seguintes programas em linguagem C, indique, justificando, qual o resultado da sua execução. Deve confirmar o resultado no compilador.

a)

```c
#include <stdio.h>
void main()
{
        int x;
        x = 3 + 4 * 5 - 6;
        printf("%d\n",x);
        x = 3 * 4 % 5 - 6;
        printf("%d\n",x);
        x = (7 + 6) % 5 / 2;
        printf("%d\n",x);
}
```

17 → * tem precedência sobre +

-4 → * tem precedência sobre %

1 → faz-se da esquerda para a direita

b)

```c
#include <stdio.h>
void main()
{
        int x = 2, y, z;
        x *= 3 + 2;
        printf("%d\n",x);
        x *= y = 4;
        printf("%d\t%d\n",x,y);
        z = (x == y);
        printf("%d\t%d\t%d\n",x,y,z);
}
```

10 → equivalente a x = x * (3+2)

40 4 → faz-se da direita para a esquerda

40 4 0 → z fica com o resultado da pergunta

c)

```c
#include <stdio.h>
void main()
{
        int x, y = 1, z;
        x = 0;
        z = 1;
        x = x && y || z;
        printf("%d\n",x);
        printf("%d\n",x || !y && z);
        printf("%d\n",z >= y && y >= x);
}
```

1 → equivalente a x=((x && y) || z)

1 → equivalente a ((x || ((!y) && z))

1 → equivalente a ((z>=y) && (y>=x))

d)

```c
#include <stdio.h>
void main()
{
        int x, y, z;
        x = y = z = -1;
        ++x || ++y || ++z;
        printf("%d %d %d\n",x,y,z);
        x = y = z = -1;
        ++x && ++y && ++z;
        printf("%d %d %d\n",x,y,z);
}
```

→ faz as atribuições da direita para a esquerda (-1, -1, -1)
→ faz as atribuições da esquerda para a direita
0 0 0
→ faz as atribuições da direita para a esquerda (-1, -1, -1)
0 -1 -1 → só é feita a 1ª operação, visto o zero tornar qualquer AND falso

e)

```
#include <stdio.h>
void main()
{
	int i, j, k;
	i = j = k = 1;
	i -= j -= k;
	printf("%d\t%d\t%d\n",i,j,k);
	i = j = 1;
	printf("%d\n",i++ - ++j);
	printf("%d\t%d\n",i, j);
}
```

```
1    0    1

-1
2    2
```

Exercício 3 – Ficha 1

Codifique um algoritmo que calcule a área e o perímetro de um círculo.

Codificação em C:

```
#include <stdio.h>
#include <math.h>

void main()
{
	float	R, A, P;

	printf("Raio do circulo: ");
	scanf("%f",&R);
	A=M_PI*R*R;
	P=2*M_PI*R;
	printf("Area do circulo: %f\nPerimetro do circulo: %f\n ",A,P);
}
```

Exercício 7 – Ficha 1

Um motorista de táxi deseja calcular o rendimento do seu carro. Considerando que o preço do combustível é de 1,3 €/litro, escreva um algoritmo para ler: a marcação do conta-quilómetros (Km) no início do dia, a marcação (Km) no final do dia, o número de litros de combustível gasto e o valor total (em €) recebido dos passageiros. Deve ainda calcular e escrever a média do consumo em Km/l e o lucro diário.

Codificação em C:

```
#include <stdio.h>
#define	PC	1.3

void main()
{
	float	kmi, kmf, cg, valor, c, l, r;

	printf("Km iniciais: ");
	scanf("%f",&kmi);
	printf("Km finais: ");
	scanf("%f",&kmf);
```

```
        printf("Combustivel gasto: ");
        scanf("%f",&cg);
        printf("Dinheiro recebido: ");
        scanf("%f",&valor);
        c=(kmf-kmi)/cg;
        l=valor-(cg*PC);
        r=100*(valor/(cg*PC)-1);
        printf("\nMedia de consumo: %9.2f",c);
        printf("\nLucro liquido: %12.2f",l);
        printf("\nRendimento do carro: %4.2f\n\n",r);
}
```

Exercício 19 – Ficha 2

Desenvolva um programa que calcule a quantidade de dinheiro que um cliente tem no banco ao fim de um ano, dada a data de hoje, o montante inicial e a taxa de juro anual. O formato em que é apresentada a informação deve ser o seguinte:

Informação necessária:

Data Actual: 10/10/2008

Montante inicial: €10000

Taxa: 10%

Cálculo do montante:

No dia 10/10/2009 vai ter no banco €11000.

Especificação ou análise do problema:

Dados de entrada:

d (inteiro) – Dia da data actual

m (inteiro) – Mês da data actual

a (inteiro) – Ano da data actual

montante (real) – Montante de dinheiro depositado na data actual

taxa (real) – Taxa de juro anual

Resultados Pretendidos:

v (real) – Quantidade de dinheiro que o cliente terá ao fim de um ano

Processamento requerido:

Obter d, m, a, montante e taxa

$v = \frac{montante \times (1+taxa)}{100}$

Mostrar v

Pseudocodigo:

INÍCIO Programa

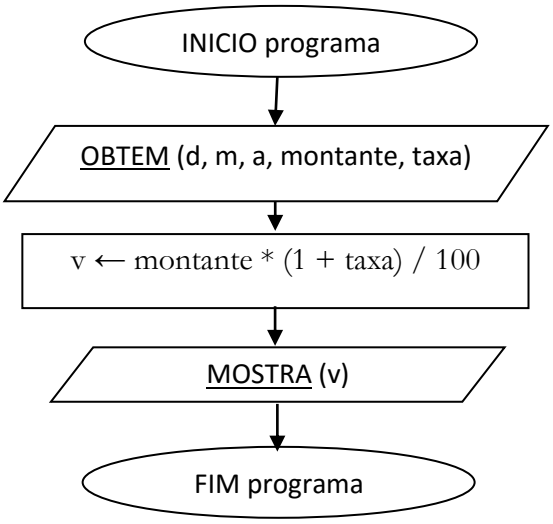
OBTEM (d, m, a, montante, taxa)

v ← montante * (1 + taxa)/100

MOSTRA (v)

FIM Programa

Fluxograma:



Codificação em C:

```
#include <stdio.h>

void main()
{
	int		d,m,a;
	float		montante,taxa,v;

	printf("Data actual: ");
	scanf("%d/%d/%d",&d,&m,&a);
	printf("\nMontante inicial: €");
	scanf("%f",&montante);
	printf("\nTaxa (%): ");
	scanf("%f",&taxa);
	v=montante*(1+taxa/100);
	printf("\nCalculo do montante:\n\nNo dia %d/%d/%d vai ter no
banco €%.2f\n\n",d,m,a+1,v);
}
```

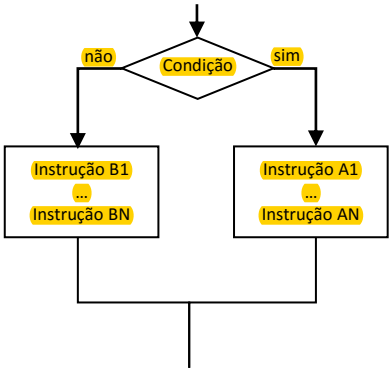
– Mecanismos de Selecção –

A instrução de controlo de fluxo de selecção:

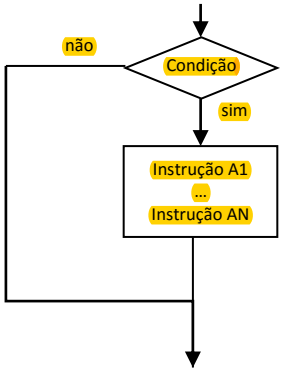
- Permite escolher uma entre várias hipóteses;
- Quando tem apenas uma hipótese pode escolhê-la ou evitá-la (o SE sem o SENÃO);
- Pode ser usado de forma encadeada.

Pseudocódigo: ***Fluxograma***

```
SE <condição> ENTÃO
    instrução A1
    …
    instrução AN
SENÃO
    instrução B1
    …
    instrução BN
FIM SE
```




```
SE <condição> ENTÃO
    instrução A1
    …
    instrução AN
FIMSE
```




```
CASO <variavel>
    QUANDO <valor1>
        instrução A1
        …
        instrução NA
    …
    QUANDO <valorN>
        instrução N1
        …
        instrução NN
FIM CASO
```

Uma das instruções de controlo de fluxo de selecção, em linguagem C, é:

```
if (condição)
{
        instrução 1
            ...
        instrução n
}
else
{
        instrução A
            ...
        instrução Z
}
```

- Pode-se omitir a parte do `else` se não for necessária;
- Podem-se omitir as chavetas se o bloco correspondente tiver só uma instrução;
- Quando existem selecções encadeadas cada `else` é associado ao `if` imediatamente anterior que se encontre no mesmo nível.

A outra instrução de controlo de fluxo de selecção, em linguagem, C é:

```
switch (variável)
{
      case constante1:
            instrução 11
            ...
            instrução 1n
            break;
      case constante2:
            instrução 21
            ...
            instrução 2n
            break;
      ...
      default:
            instrução d1
            ...
            instrução dn
}
```

- É usada quando o valor a testar pode assumir um de vários valores literais;
- Uma vez entrando num `case`, as instruções são executadas até que se encontre um `break`;
- A opção `default` é usada para especificar o que fazer caso a variável não tenha nenhum dos valores anteriores (e pode ser omitido).

Exercício 15 – Ficha 1

Escreva um algoritmo para ler o número de lados de um polígono regular e a medida do lado. Deve ainda calcular e imprimir a seguinte informação:

- Se o número de lados for igual a 3, escrever TRIÂNGULO e o valor do seu perímetro.
- Se o número de lados for igual a 4, escrever QUADRADO e o valor da sua área.
- Se o número de lados for igual a 5, escrever PENTÁGONO.

OBS: Considere que o utilizador só introduz os valores 3, 4 ou 5.

Especificação ou análise do problema:

Dados de entrada:

n (inteiro) – Número de lados do polígono

l (real) – Medida do lado do polígono

Resultados Pretendidos:

m (mensagem) – Identificação do tipo de polígono

val (real) – valor do perímetro ou área do polígono

Processamento requerido:

Obter n e l

Conforme o valor de n, escrever em m a identificação correspondente

Conforme o valor de n, calcular o val correspondente

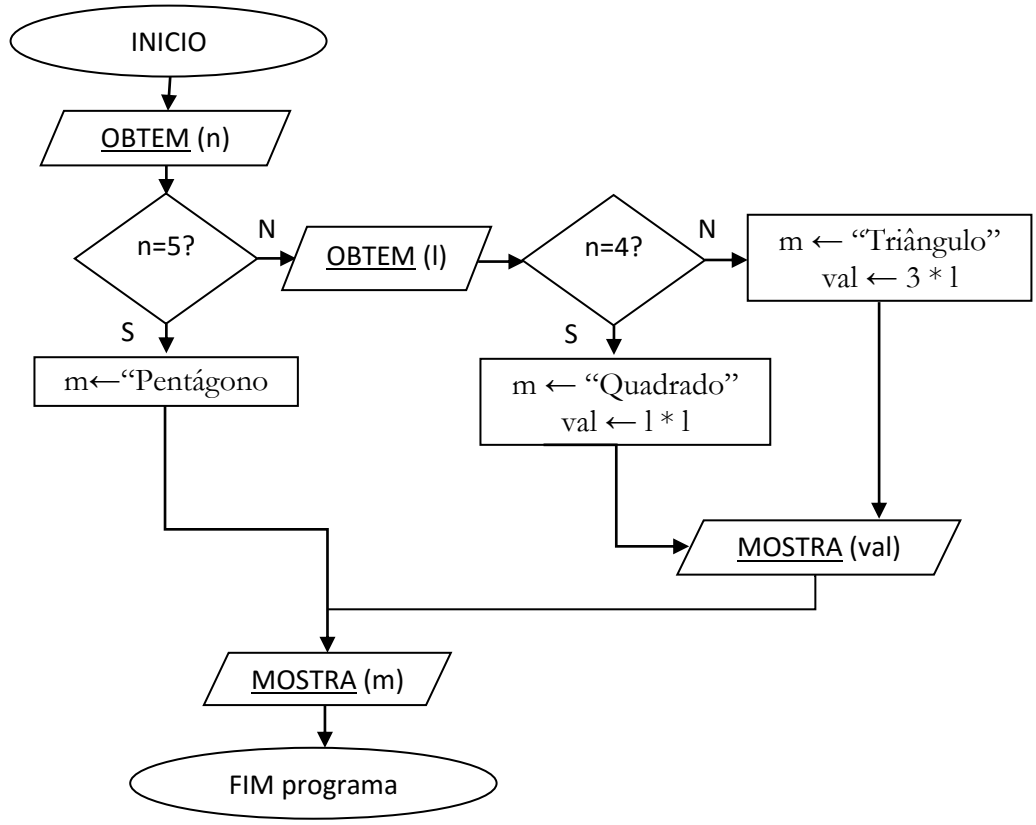
Mostrar m e val

Pseudocodigo:

```
INÍCIO Programa
    OBTEM (n)
    SE (n=5) ENTÃO
        m ← "Pentágono"
    SENÃO
        OBTEM (l)
        SE (n=4) ENTÃO
            m ← "Quadrado"
            val ← l*l
        SENÃO
            m ← "Triângulo"
            val ← 3*l
        FIM SE
        MOSTRA (val)
    FIM SE
    MOSTRA (m)
FIM Programa
```

```
INÍCIO Programa
    OBTEM (n)
    CASO (n)
        QUANDO 5
            m ← "Pentágono"
        QUANDO 4
            OBTEM (l)
            m ← "Quadrado"
            val ← l*l
            MOSTRA (val)
        QUANDO 3
            OBTEM (l)
            m ← "Triângulo"
            val ← 3*l
            MOSTRA (val)
    FIM CASO
    MOSTRA (m)
FIM Programa
```

Fluxograma:



Codificação em C (usando if…else):

```
#include <stdio.h>
void main()
{
	int   n;
	float l,val;
	printf("Introduza o numero de lados da figura: ");
	scanf("%d",&n);
	if (n==5)
		printf("Pentagono\n");
	else
	{
		printf("Introduza a medida do lado da figura: ");
		scanf("%f",&l);
		if (n==3)
		{
			val=n*l;
			printf("Triangulo de perimetro %f\n",val);
		}
		else
		{
			val=l*l;
			printf("Quadrado de area %f\n",val);
		}
	}
}
```

Codificação em C (usando switch…case):

```
#include <stdio.h>

void main()
{
      int   n;
      float l,val;

      printf("Introduza o numero de lados da figura: ");
      scanf("%d",&n);
      switch (n)
      {
          case 5:printf("Pentagono\n");
                 break;
          case 4:printf("Introduza a medida do lado da figura: ");
                 scanf("%f",&l);
                 val=l*l;
                 printf("Quadrado de area %f\n",val);
                 break;
          case 3:printf("Introduza a medida do lado da figura: ");
                 scanf("%f",&l);
                 val=n*l;
                 printf("Triangulo de perimetro %f\n",val);
                 break;
      }
}
```

Exercício 4 – Ficha 3

Desenvolva um programa que, após ler as coordenadas de um ponto do plano na forma de par ordenado (x, y), determine a que quadrante pertence.

Especificação ou análise do programa:

Dados de Entrada:

x e y (real) – Coordenada x e y do plano.

Resultados Pretendidos:

msg – Mensagem a dizer onde pertence o ponto introduzido.

Processamento requerido:

Obtém (x, y)

Se x=0 e y=0, msg ← "Origem"

Se x=0 e y≠0, msg ← "Eixo dos XX"

Se y=0 e x≠0, msg ← "Eixo dos YY"

Se x>0 e y>0, msg ← "1º Quadrante"

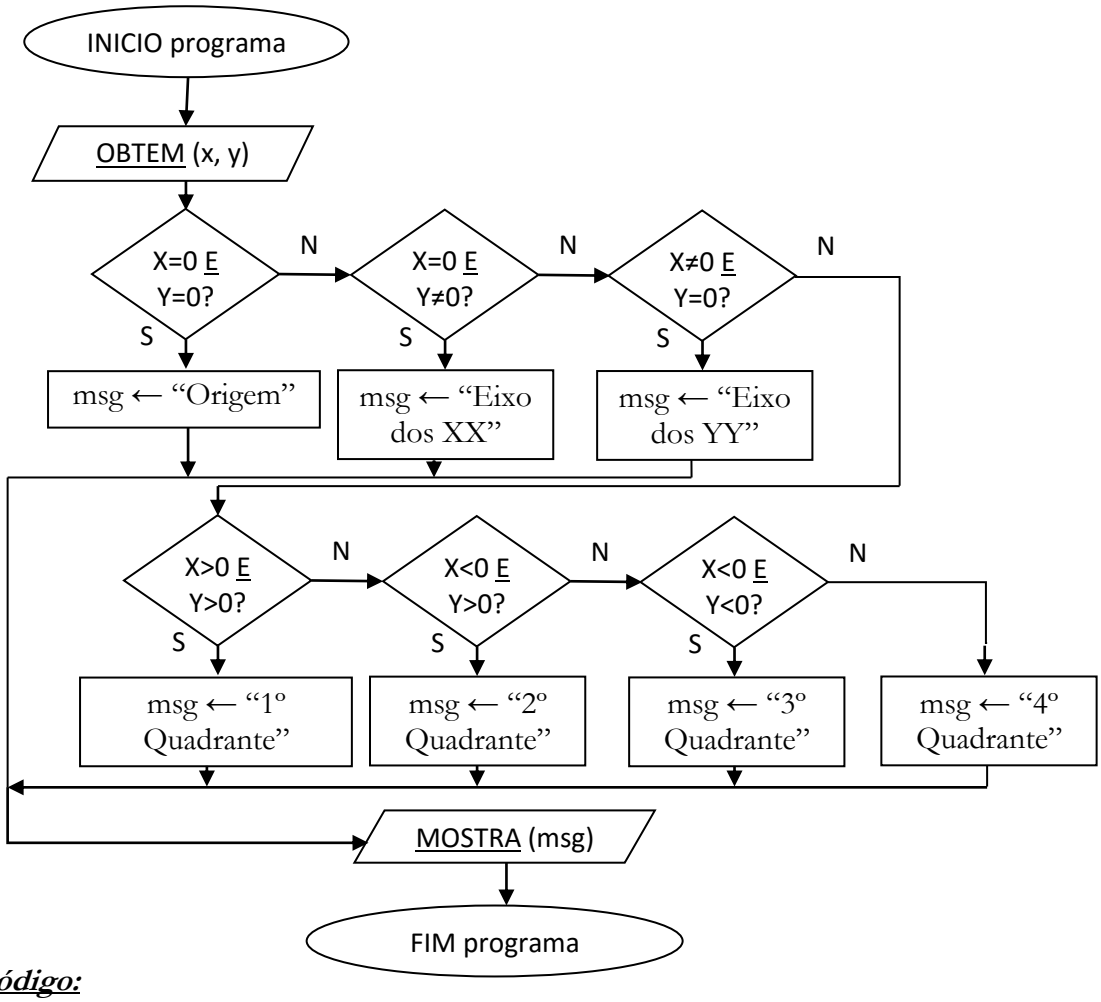
Se x<0 e y>0, msg ← "2º Quadrante"

Se x<0 e y<0, msg ← "3º Quadrante"

Se x>0 e y<0, msg ← "4º Quadrante"

Mostra (msg)

Fluxograma:



Pseudocódigo:

```
INÍCIO programa
        OBTEM (x, y)
        SE (x=0 E y=0) ENTÃO
                msg ← "Origem"
        SENAO
                SE (x=0 E y≠0) ENTÃO
                        msg ← "Eixo dos XX"
                SENAO
                        SE (y=0 E x≠0) ENTÃO
                                msg ← "Eixo dos YY"
                        SENAO
                                SE (x>0 E y>0) ENTÃO
                                        msg ← "1º Quadrante"
                                SENAO
                                        SE (x<0 E y>0) ENTÃO
                                                msg ← "2º Quadrante"
                                        SENAO
                                                SE (x<0 E y<0) ENTÃO
                                                        msg ← "3º Quadrante"
                                                SENAO
                                                        msg ← "4º Quadrante"
                                                FIM SE
```

FIM SE
FIM SE
FIM SE
FIM SE
FIM SE
MOSTRA (msg)
FIM programa

Codificação em C:

```
#include <stdio.h>
void main()
{
	float x,y;
	printf("Introduza as coordenadas (x,y): ");
	scanf("(%f,%f)",&x,&y);
	if (x==0)
	{
		if(y==0)
		{
			printf("Origem\n");
		}
		else
		{
			printf("Eixo dos XX\n");
		}
	}
	else
	{
		if (y==0)
		{
			printf("eixo dos YY\n");
		}
		else
		{
			if (x>0)
			{
				if (y>0)
					printf("1 Quadrante\n");
				else
				{
					printf("4 Quadrante\n");
				}
			}
			else
			{
				if (y>0)
					printf("2 Quadrante\n");
				else
					printf("3 Quadrante\n");
			}
		}
	}
}
```

Exercício 22 – Ficha 1 / Exercício 10 – Ficha 3

Escreva o algoritmo de um programa em que, dados três inteiros que representam o dia, mês e ano de uma determinada data, calcule e imprima o dia, o mês e o ano, relativos à data do dia seguinte. Não considere os anos bissextos. / Desenvolva um programa que dada a data atual no formato dd/mm/aaaa determine a data do dia seguinte. A leitura da data deve ser efetuada só com uma instrução de **scanf(…)**. Deverão ser codificadas duas versões deste o problema, uma primeira utilizando a instrução **if … else** e a segunda utilizando a instrução **switch … case**. Assuma que não existem anos bissextos.

Especificação ou análise do problema:

Dados de entrada:

d (inteiro) – Dia da data actual

m (inteiro) – Mês da data actual

a (inteiro) – Ano da data actual

Resultados Pretendidos:

d (inteiro) – Dia da data do dia seguinte

m (inteiro) – Mês da data do dia seguinte

a (inteiro) – Ano da data do dia seguinte

Processamento requerido:

Obter d, m e a

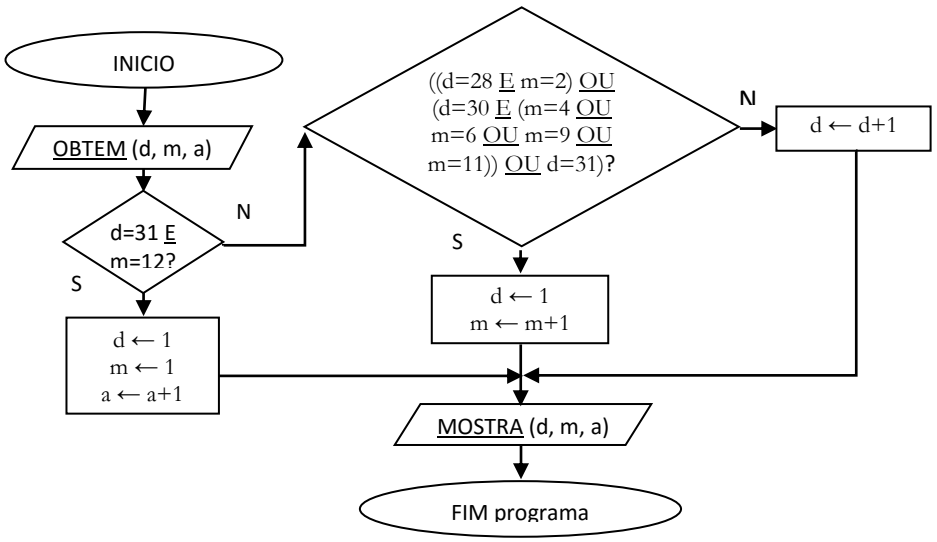
Se for o último dia do ano, passar para o primeiro dia do ano seguinte

Se for o último dia de qualquer um dos outros meses que não o de Dezembro, passar para o primeiro dia do mês seguinte

Se não for nenhuma das hipóteses anteriores, passar para o dia seguinte desse mês

Mostrar d, m e a

Fluxograma:



Pseudocodigo (com SE…ENTÃO…SENÃO…FIM SE):

```
INÍCIO Programa
        OBTEM (d, m, a)
        SE (d=31 E m=12) ENTÃO
                d ← 1
                m ← 1
                a ← a+1
        SENÃO
                SE ((d=28 E m=2) OU (d=30 E (m=4 OU m=6 OU m=9 OU m=11))
                OU d=31) ENTÃO
                        d ← 1
                        m ← m+1
                SENÃO
                        d ← d+1
                FIM SE
        FIM SE
        MOSTRA (d, m, a)
FIM Programa
```

Codificação em C:

```
#include <stdio.h>
void main()
{
      int   d,m,a;
      printf("Introduza a data de hoje (dd/mm/aa): ");
      scanf("%d/%d/%d",&d,&m,&a);
      if (d==31 && m==12)
      {
            d=1;
            m=1;
            a++;
      }
      else
            if ((d==28 && m==2) || (d==30 && (m==4 || m==6 || m==9 ||
m==11)) || d==31)
            {
                  d=1;
                  m++;
            }
            else
                  d++;
      printf("Data seguinte: %d/%d/%d\n",d,m,a);
}
```

Pseudocodigo (com CASO…QUANDO):

```
INÍCIO Programa
        OBTEM (d, m, a)
        CASO (d)
                QUANDO 28
                        CASO (m)
```

```
                        QUANDO (2)
                            d ← 1
                            m ← m+1
                        QUANDO (restantes meses)
                            d ← d+1
                FIM CASO
        QUANDO 30
                CASO (m)
                        QUANDO (4,6,9,11)
                            d ← 1
                            m ← m+1
                        QUANDO (restantes meses)
                            d ← d+1
                FIM CASO
        QUANDO 31
                d ← 1
                m ← m+1
                CASO (m)
                        QUANDO 13
                            m ← 1
                            a ← a+1
                FIM CASO
        QUANDO Restantes dias
                d ← d+1
    FIM CASO
    MOSTRA (d, m, a)
FIM Programa
```

Codificação em C:

```c
#include <stdio.h>

void main()
{
      int d,m,a;

      printf("Introduza a data de hoje (dd/mm/aaaa): ");
      scanf("%d/%d/%d", &d,&m,&a);
      switch (d)
      {
            case 28:
                  switch (m)
                  {
                        case 2:
                              d=1;
                              m=m+1;
                              break;
                        default:
                              d=d+1;
                  }
                  break;
            case 30:
                  switch (m)
                  {
                        case 4:
```

```
                        case 6:
                        case 9:
                        case 11:
                              d=1;
                              m=m+1;
                              break;
                        default:
                              d=d+1;
                  }
                  break;
            case 31:
                  d=1;
                  m=m+1;
                  switch(m)
                  {
                        case 13:
                              m=1;
                              a=a+1;
                              break;
                  }
                  break;
            default:
                  d=d+1;
      }
      printf("\n\nA data actualizada e %d%/%d%/%d\n\n",d,m,a);
}
```

Pseudocodigo (com misturas):

INÍCIO Programa
    OBTEM (d, m, a)
    CASO (m)
        QUANDO 2
            ultimodia ← 28
        QUANDO 4, 6, 9 ou 11
            ultimodia ← 30
        QUANDO 1, 3, 5, 7, 8, 10 ou 12
            ultimodia ← 31
    FIM CASO
    SE (d=ultimodia) ENTÃO
        d ← 1
        m ← m+1
        SE (m=13) ENTÃO
            m ← 1
            a ← a+1
        FIM SE
    SENÃO
        d ← d+1
    FIM SE
    MOSTRA (d, m, a)
FIM Programa

– Mecanismos de Repetição –

A instrução de controlo de fluxo de repetição:

- Permite repetir um conjunto de instruções até que determinada condição se torne falsa;
- É fundamental que alguma das instruções no corpo do ciclo permita transformar a condição numa falsidade (de forma a poder sair do ciclo).

Pseudocódigo

Enquanto (condição) Fazer

Bloco de Instruções

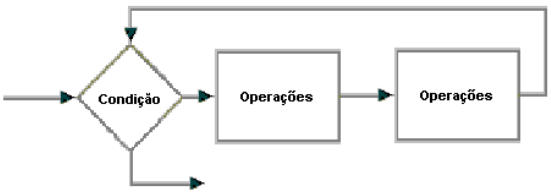
FimEnquanto

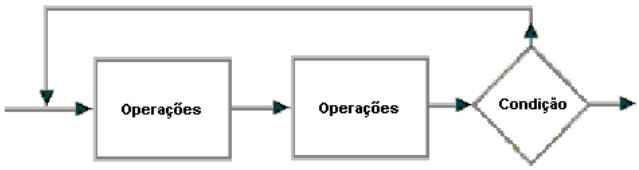
Fazer

Bloco de Instruções

Enquanto (condição)

Fluxograma



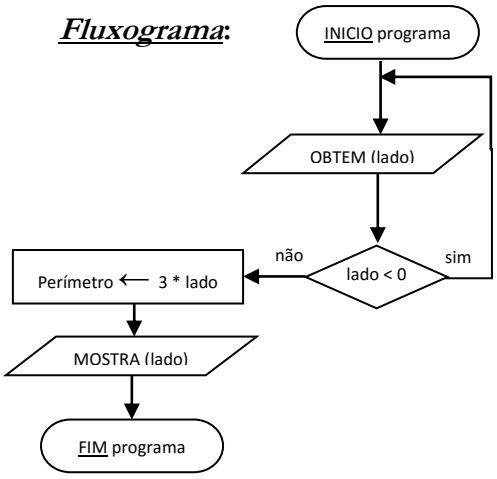


Exemplo: Calcule o perímetro de um triângulo regular. Enquanto o utilizador introduzir valores negativos para o comprimento do lado voltar a pedir o lado.

***Pseudocódigo*:**

INÍCIO programa
    FAZER
        OBTEM (lado)
    ENQUANTO (lado < 0)
    Perímetro ← 3 * lado
    MOSTRA (Perímetro)
FIM programa

***Fluxograma*:**



Uma das instruções de ciclo, em linguagem, C é:

- ```
  for (expr1;condição;expr3)
  {
      Bloco de instruções; /*corpo do ciclo */
  }
  ```
  onde
  - **expr1** é a expressão de inicialização executada uma única vez, à entrada no ciclo;
  - **condição** é a expressão que controla o fim do ciclo. Enquanto a expressão for verdadeira (diferente de zero), o bloco de instruções é executada. A avaliação da expressão é sempre efectuada antes da execução do corpo do ciclo;
  - **expr3** é a operação que é executada no fim de cada iteração do ciclo.
  ```

Exemplo - Tabuada

```
#include <stdio.h>

void main()
{
   int i,numero;

   printf("Numero:");
   scanf("%d",&numero);
   for (i=1;i<=10;i=i+1)
       printf("%2d*%2d=%2d\n",numero,i,numero*i);
}
```

Outra das instruções de ciclo, em linguagem, C é:

- `while (condição)`
  `{`
  `Bloco de instruções;` `/*corpo do ciclo */`
  `}`
  onde
  - A execução da instrução (ou bloco de instruções) é efectuada enquanto a avaliação da condição produzir um valor diferente de zero.
  - Esta avaliação é sempre feita antes da execução do corpo do ciclo (se logo à entrada do ciclo a condição tiver um valor falso, igual a zero, o corpo do ciclo não é executado).

Exemplo - Tabuada

```
#include <stdio.h>

void main()
{
     int i=1,numero;

     printf("Numero:");
     scanf("%d",&numero);
     while (i<=10)
     {
         printf("%2d*%2d=%2d\n",numero,i,numero*i);
         i=i+1;
     }
}
```

A última das instruções de ciclo, em linguagem, C é:

- `do`
  `{`
  `Bloco de instruções;` `/*corpo do ciclo */`
  `}`
  `while (condição);`
  onde
  - Neste tipo de ciclo o bloco de instruções é executado primeiro e só depois é que a condição é avaliada.
  - Se a avaliação da condição for verdadeira, a execução do ciclo continua. Se não, termina.
  - No ciclo `do-while` o corpo do ciclo é sempre executado pelo menos uma vez, ainda que a condição seja falsa (igual a zero).

Exemplo - Tabuada

```
#include <stdio.h>

void main()
{
    int i=1,numero;

    printf("Numero:");
    scanf("%d",&numero);
    do
    {
        printf("%2d*%2d = %2d\n",numero,i,numero*i);
        i++;
    }while(i<=10);
}
```

Exercício 26 – Ficha 1

Escreva um algoritmo que dado uma base e um expoente, calcule a respectiva potência através de multiplicações sucessivas.

***Especificação ou análise do problema*:**

Dados de Entrada:

base (real) – Número da base
expoente (inteiro) - Número do expoente

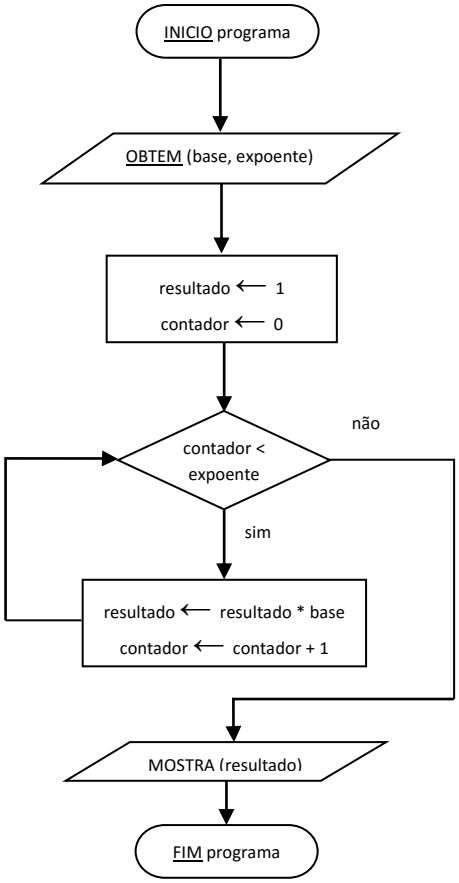
Resultados Pretendidos:

resultado (real) – resultado de base^{expoente}

Processamento requerido:

Obter (base, expoente)
Fazer expoente multiplicações de base
Mostrar (resultado)

***Fluxograma*:**



***Pseudocódigo*:**

INÍCIO programa
    OBTEM (base,expoente)
    resultado ⟵ 1
    contador ⟵ 0
    ENQUANTO (contador < expoente) FAZER
        resultado ⟵ resultado * base
        contador ⟵ contador +1
    FIM ENQUANTO
    MOSTRA (resultado)
FIM programa

Codificação em C:

Versão for

```
#include <stdio.h>
void main()
{
      float base,resultado=1;
      int   expoente,contador;
      printf("Introduza a base: ");
      scanf("%f",&base);
      printf("\nIntroduza o expoente: ");
      scanf("%d",&expoente);
      for (contador=0;contador<expoente;contador++)
            resultado=resultado*base;
      printf("Resultado: %f\n",resultado);
}
```

Versão while

```
#include <stdio.h>
void main()
{
      float base,resultado=1;
      int   expoente,contador=0;
      printf("Introduza a base: ");
      scanf("%f",&base);
      printf("\nIntroduza o expoente: ");
      scanf("%d",&expoente);
      while (contador<expoente)
      {
            resultado=resultado*base;
            contador=contador+1;
      }
      printf("Resultado: %f\n",resultado);
}
```

Versão do…while

```
#include <stdio.h>
void main()
{
      float base,resultado=1;
      int   expoente,contador=0;
      printf("Introduza a base: ");
      scanf("%f",&base);
      printf("\nIntroduza o expoente: ");
      scanf("%d",&expoente);
      if (expoente != 0)
      {
            do
            {
                  resultado=resultado*base;
                  contador=contador+1;
            }
            while (contador<expoente);
      }
      printf("Resultado: %f\n",resultado);
}
```

Exercício 29 – Ficha 1

Desenvolva um algoritmo para ler um número indeterminado de dados, contendo cada um o peso de um indivíduo. O último dado que não entrará nos cálculos, contém um valor negativo. O algoritmo deve calcular e imprimir:

- A média aritmética das pessoas que possuem mais de 60 kg.
- O peso da pessoa mais pesada que integra o grupo de indivíduos com menos de 60 kg.

Especificação ou análise do problema:

Dados de Entrada:

p1, p2,…, pn (real) - Peso das n pessoas

Resultados Pretendidos:

media (real) - Média dos pesos das pessoas com mais de 60 kg
mp (real) – Valor do peso maior entre todas as pessoas abaixo de 60 kg

Processamento requerido:

Obter (p)
Processar p enquanto for positivo, sendo que o processamento consistirá em:
Contar o número de pessoas com peso maior que 60 kg;
Somar todos os pesos maiores que 60 kg;
Guardar o valor do peso, sempre que encontrar algum que tenha peso menor que 60 kg, mas maior que o guardado.
Obter a média das pessoas que possuem mais de 60 kg
Mostrar (media, mp)

Pseudocódigo:

```
INÍCIO programa
    media ← 0
    contador ← 0
    mp ← 0
    FAZER
        OBTEM (p)
        SE (p>60) ENTÃO
                media ← media + p
                contador ← contador + 1
        SENÃO
                SE (p<60) ENTÃO
                        SE (mp<p) ENTÃO
                                mp ← p
                        FIM SE
                FIM SE
        FIM SE
    ENQUANTO (p>=0)
    media ← media / contador
    MOSTRA (media, mp)
 FIM  programa
```

Codificação em C:

```
#include <stdio.h>
void main()
```

```
{
	int		contador=0;
	float		p,mp=0.0,media=0.0;

	do
	{
		printf("Introduza o peso da pessoa %d : ",contador+1);
		scanf("%f",&p);
		if (p>60)
		{
			media=media+p;
			contador=contador+1;
		}
		else
			if (p<60)
				if (mp<p)
					mp=p;
	}
	while (p>=0);
	printf("Media dos pesos (+60kg) - %.2f\n\n",media/contador);
	printf("Pessoa mais pesada, com menos de 60kg - %.2f\n\n",mp);
}
```

Exercício 30 – Ficha 1

Supondo que a população do bairro A é 500 habitantes e que regista uma taxa anual de crescimento de 5,2% ao ano e que a população do bairro B é 1500 habitantes com uma taxa anual de crescimento de 1,8 %, desenvolva um algoritmo que calcule quantos anos serão necessários para que a população do bairro A ultrapasse a do bairro B. Considere que as taxas anuais de crescimento permanecem constantes.

Especificação ou análise do problema:

Constantes:

TCA (5.2%) - taxa de crescimento do bairro A
TCB (1.8%) - taxa de crescimento do bairro B

Dados de Entrada:

pa (real) - número de habitantes do bairro A, (500 inicialmente) – Não se OBTEM
pb (real) - número de habitantes do bairro B, (1500 inicialmente) – Não se OBTEM

Resultados Pretendidos:

n (inteiro) - Número de anos necessários até que a população do bairro A exceda a população do bairro B.

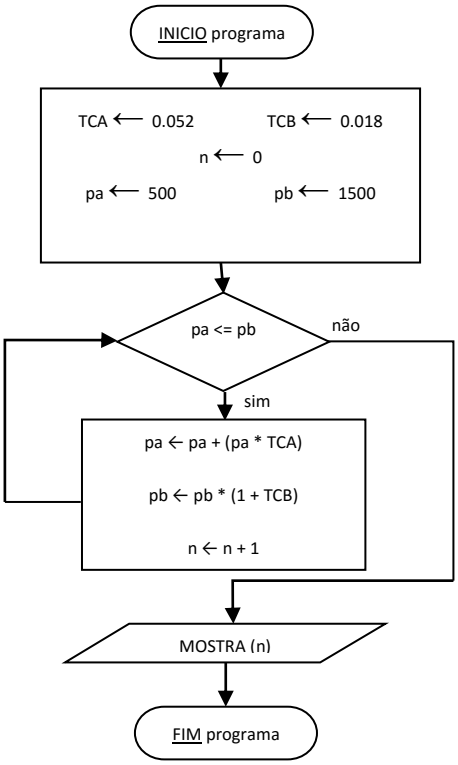
Processamento requerido:

Inicializar pa e pb com número inicial de habitantes dos respectivos bairros
Conta o número de vezes que é necessário fazer o cálculo
pa ← pa + (pa * TCA) e pb ← pb * (1 + TCB)
até que pa seja maior do que pb
Mostrar (n)

Pseudocódigo:

INÍCIO programa
    TCA ← 0.052
    TCB ← 0.018
    pa ← 500
    pb ← 1500
    n ← 0
    ENQUANTO (pa<=pb) FAZER
        pa ← pa + (pa * TCA)
        pb ← pb * (1 + TCB)
        n ← n + 1
    FIM ENQUANTO
    MOSTRA (n)
FIM programa

Fluxograma:



Codificação em C:

```
#include <stdio.h>

#define    TCA   0.052
#define    TCB   0.018

void main()
{
      int         n=0;
      float       pa=500,pb=1500;

      while (pa<=pb)
      {
            pa=pa+(pa*TCA);
            pb=pb*(1+TCB);
            n=n+1;
      }
      printf("Numero de anos: %d\n",n);
}
```

Exercício 32 – Ficha 1

Elabore um algoritmo que obtenha um número inteiro e que calcule a soma dos seus dígitos (Sugestão: os dígitos de um número inteiro são obtidos efectuando sucessivas divisões por 10).

Especificação ou análise do problema:

Dados de Entrada:
n (inteiro) - Número a processar

Resultados Pretendidos:
s (inteiro) - Soma dos dígitos

Processamento requerido:
Obter (n)
Separar os dígitos através de divisões inteiras sucessivas por 10 (resto fica com o digito menos significativo, quociente corresponde aos dígitos que faltam processar).
Somar os dígitos à medida que vão sendo obtidos.
Mostrar (s)

Pseudocódigo:

INÍCIO programa
    OBTEM (n)
    s ← 0
    ENQUANTO (n>0) FAZER
        d ← resto da divisão inteira entre n e 10
        n ← n/10
        s ← s + d
    FIM ENQUANTO
    MOSTRA (s)
FIM programa

Codificação em C:

```
#include <stdio.h>
void main()
{
      int   numero,digito,soma;
      soma=0;
      printf("Introduza um numero inteiro: ");
      scanf("%d",&numero);
      while (numero>0)
      {
            digito=numero%10;
            numero=numero/10;
            soma=soma+digito;
      }
      printf("soma dos digitos do numero - %d\n\n",soma);
}
```

Exercício 35 – Ficha 1

Desenvolva um algoritmo que obtenha os valores de a, b, c, d, e, f (garantindo que estes são positivos) e que calcule o seguinte somatório triplo.

$$\sum_{i=a}^{b}\sum_{j=c}^{d}\sum_{y=e}^{f} i \times j \times y$$

Especificação ou análise do problema:

Dados de Entrada:
a, c, e (inteiro) - Limites inferiores dos somatórios
b, d, f (inteiro) - Limites superiores dos somatórios

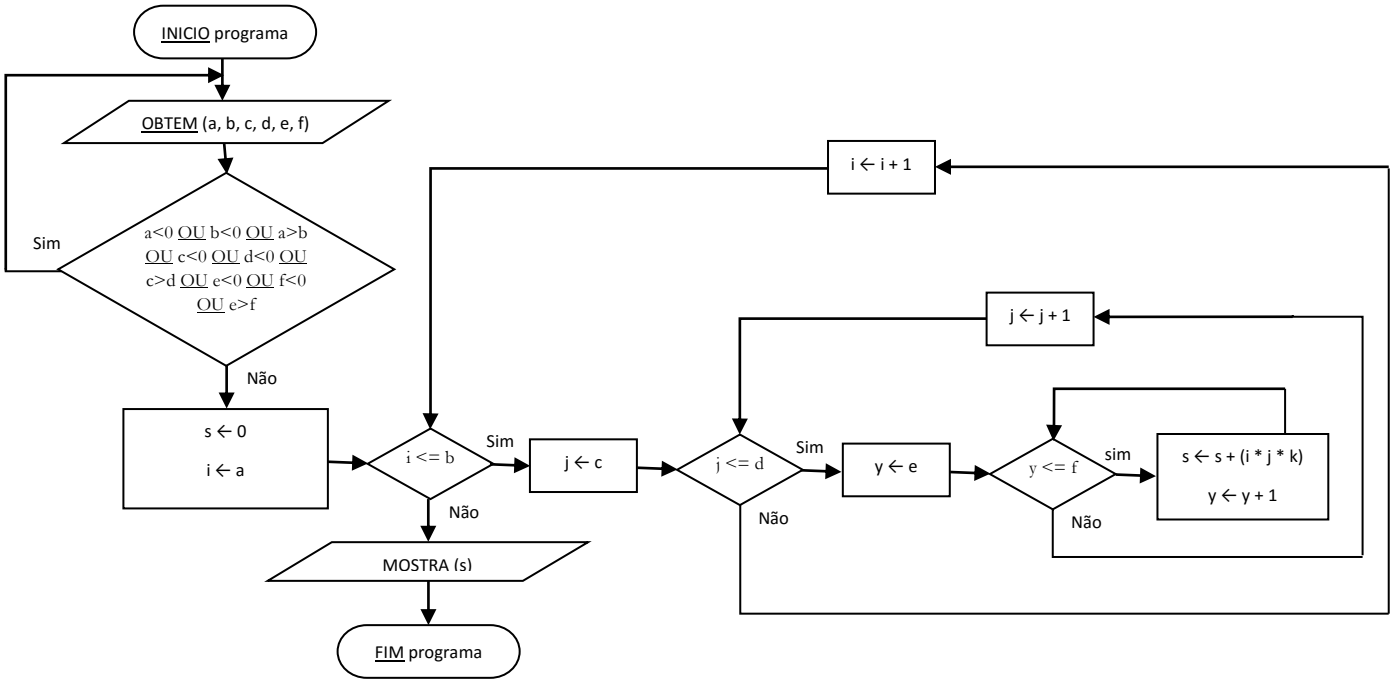
Resultados Pretendidos:
s (inteiro) - Valor dos somatórios

Processamento requerido:
Obter os valores positivos dos limites dos somatórios (a, b, c, d , e, f)
Fazer uma série de ciclos encadeados (um por cada somatório) e ir acumulando o valor da expressão.
Mostrar o valor calculado (s)

Pseudocódigo:

```
INÍCIO programa
    FAZER
        OBTEM (a,b,c,d,e,f)
    ENQUANTO (a<0 OU b<0 OU a>b OU c<0 OU d<0 OU c>d OU e<0 OU
    f<0 OU e>f)
    s ← 0
    i ← a
    ENQUANTO (i<=b) FAZER
        j ← c
        ENQUANTO (j<=d) FAZER
            y ← e
            ENQUANTO (y<=f) FAZER
                s ← s + (i * j * y)
                y ← y + 1
            FIM ENQUANTO
            j ← j + 1
        FIM ENQUANTO
        i ← i + 1
    FIM ENQUANTO
    MOSTRA (s)
FIM programa
```

Fluxograma:



Codificação em C:

```
#include <stdio.h>
void main()
{
      int   a,b,c,d,e,f,s,i,j,y;
```

```
        do
        {
              printf("Valor de A e de B: ");
              scanf("%d %d",&a,&b);
        }
        while (a<0 || b<0 || a>b);
        do
        {
              printf("Valor de C e de D: ");
              scanf("%d %d",&c,&d);
        }
        while (c<0 || d<0 || c>d);
        do
        {
              printf("Valor de E e de F: ");
              scanf("%d %d",&e,&f);
        }
        while (e<0 || f<0 || e>f);
        s=0;
        for (i=a;i<=b;i++)
              for (j=c;j<=d;j++)
                    for (y=e;y<=f;y++)
                          s=s+(i*j*y);
        printf("Valor do somatorio: %d\n",s);
}
```

Exercício 2 – Ficha 4

Desenvolva um programa que leia uma sequência de salários do pessoal de uma empresa e calcule o salário médio.

Implemente duas versões: na primeira o utilizador informa, no início da execução, quantos empregados vão ser considerados. Na segunda versão, a introdução termina quando surgir um ordenado nulo ou negativo.

Versão 1

Especificação ou análise do programa:

Dados de Entrada:

n (inteiro) – numero de salários a considerar.
s1, s2,…, sn (reais) – Os n salários.

Resultados Pretendidos:

media (real) – Valor médio dos salários.

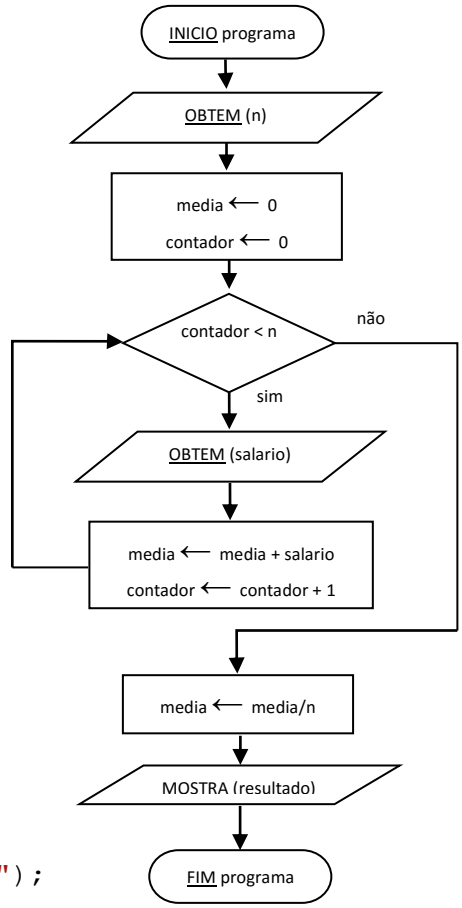
Processamento requerido:

Obtém (n)
Obtém (s1, s2, …, sn)
media ← $\sum_1^n si/n$
Mostra (media)

Pseudocódigo:

INÍCIO programa
    OBTEM (n)
    contador ← 0
    media ← 0
    ENQUANTO (contador<n) FAZER
        OBTEM (salario)
        media ← media +salário
        contador ← contador+1
    FIM ENQUANTO
    media ← media/n
    MOSTRA (media)
FIM programa

Fluxograma:



Codificação em C:

```
#include <stdio.h>
void main()
{
      int         n,contador;
      float       salario,media;
      printf("Introduza o numero de empregados: ");
      scanf("%d",&n);
      for (media=0,contador=0;contador<n;contador=contador+1)
      {
            printf("Salario do %d empregado: ",contador+1);
            scanf("%f",&salario);
            media= media+salario;
      }
      if (n!=0)
            printf("Media salarios - %.2f\n",media/n);
}
```

Versão 2

Especificação ou análise do programa:

Dados de Entrada:
s1, s2,…, sn (reais) – Os n salários.

Resultados Pretendidos:
media (real) – Valor médio dos salários.

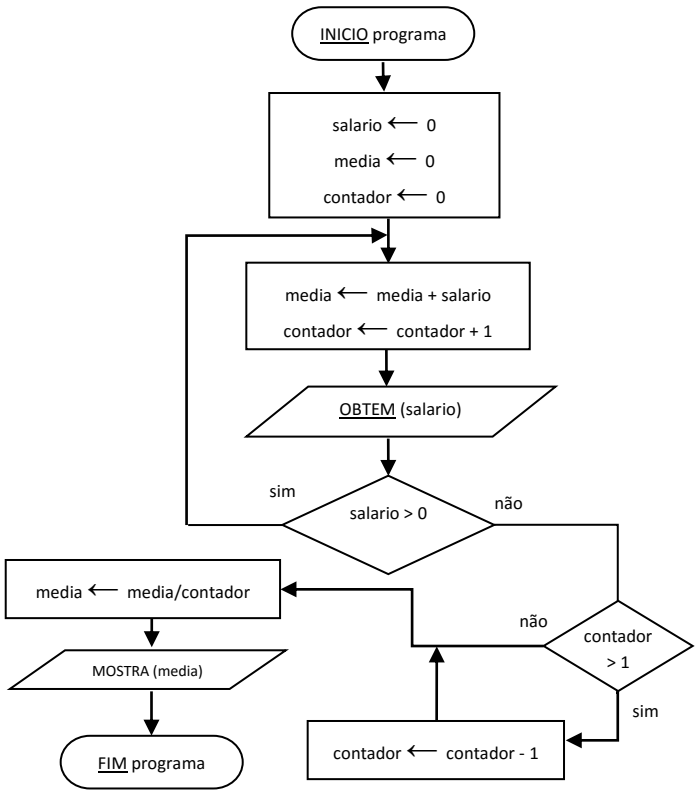
Processamento requerido:
Obtém (s1, s2, …, sn)
media ← $\sum_1^n s/n$
Mostra (media)

Pseudocódigo:

INÍCIO programa
    salario ← 0

media ← 0
contador ← 0
FAZER
media ← media +salario
contador ← contador + 1
OBTEM (salario)
ENQUANTO (salario>0)
SE (contador > 1)
contador ← contador - 1
FIM SE
media ← media / contador
MOSTRA (media)
FIM programa

Fluxograma:



Codificação em C:

```
#include <stdio.h>

void main()
{
      int         contador=0;
      float       salario=0,media=0;

      do
      {
            media=media+salario;
            contador=contador+1;
            printf("Introduza o salario do %d empregado: ",contador);
            scanf("%f",&salario);
      }
      while (salario>0);
      if (contador>1)
            contador=contador-1;
      printf("Media dos salarios - %.2f\n\n",media/contador);
}
```

Exercício 3 – Ficha 4

Desenvolva um programa que simule o funcionamento de uma caixa registadora de um supermercado. Considere que existem produtos de dois tipos: alimentares e não alimentares. O programa deve receber como entrada o preço e o tipo de um conjunto de produtos. A introdução termina quando surgir um preço nulo ou negativo. Os preços introduzidos estão sujeitos a uma taxa de IVA de 6% para os produtos alimentares e de 23% para os produtos não alimentares. Para cada cliente deve ser emitido um talão com a seguinte informação: número de produtos alimentares, número de produtos não alimentares, número total de produtos, preço sem IVA e preço com IVA (com duas casas decimais).

Especificação ou análise do programa:

Dados de Entrada:

tp (caracter) – Tipo de produto (alimentar – 'A' ou 'a' e não alimentares – 'O' ou 'o').
preco (real) – O preço do item.

Resultados Pretendidos:

numAlimentos (inteiro) – Número de produtos alimentares
numOutros (inteiro) – Número de produtos não alimentares
nprods (inteiro) – Soma do número total de produtos
precoTotal (real) – Preço total sem IVA
precoTotalIVA (real) – Preço total com IVA

Processamento requerido:

Fazer o processo abaixo enquanto preço positivo:
Obter (preço, tp)
Contar o número de produtos alimentares
Contar o número de produtos não alimentares
Somar os preços sem IVA de todos os produtos processados
Calcular o preço do produto com IVA
Somar os preços com IVA calculado de todos os produtos processados
Mostrar (numAlimentos, numOutros, nprods, precoTotal, precoTotalIVA)

Pseudocódigo:

```
INÍCIO programa
    numAlimentos ← 0
    numOutros ← 0
    precoTotal ← 0
    precoTotalIVA ← 0
    FAZER
            OBTEM (preco)
            SE (preço>0)
                FAZER
                    OBTEM (tp)
                ENQUANTO (tp≠'a' E tp≠'A' E tp≠'o' E tp≠'O')
                CASO (tp)
                    QUANDO 'a' OU 'A'
                        numAlimentos ← numAlimentos +1
                        taxa ← 0.06
                    QUANDO 'o' OU 'O'
                        numOutros ← numOutros +1
                        taxa ← 0.23
                FIM CASO
                precoTotal ← precoTotal + preco
                precoTotalIVA ← precoTotalIVA + preço * (1 + taxa)
            FIM SE
    ENQUANTO (preco>0)
    MOSTRA (numAlimentos, numOutros, precoTotal, precoTotalIVA)
FIM programa
```

Codificação em C:

```c
#include <stdio.h>

void main()
{
      float preco,precoTotalIVA=0,precoTotal=0,taxa;
      int   numAlimentos=0,numOutros=0;
      char  tp;

      do
      {
```

```
            fflush(stdin);
            printf("Introduza o preco unitario do produto: ");
            scanf("%f",&preco);
            if (preco>0)
            {
                  tp=' ';
                  while (tp!='A' && tp!='a' && tp!='O' && tp!='o')
                  {
                        fflush(stdin);
                        printf("Tipo (A- Alimentares, O- Outros): ");
                        scanf("%c",&tp);
                        if (tp!='A' && tp!='a' && tp!='O' && tp!='o')
                              printf("Seleccao de tipo invalida\n");
                  }
                  switch (tp)
                  {
                        case 'A':
                        case 'a':
                              numAlimentos++;
                              taxa=0.06;
                              break;
                        case 'O':
                        case 'o':
                              numOutros++;
                              taxa=0.23;
                              break;
                  }
                  precoTotal+=preco;
                  precoTotalIVA+=preco*(1+taxa);
            }
      }
      while (preco>0);
      printf("\n\nProdutos alimentares - %d \n",numAlimentos);
      printf("Produtos nao alimentares - %d \n",numOutros);
      printf("Total de produtos - %d\n",numAlimentos+numOutros);
      printf("Preco total sem IVA - %.2f\n",precoTotal);
      printf("Preco total com IVA - %.2f\n",precoTotalIVA);
}
```

Exercício 4 – Ficha 4

Desenvolva um programa que escreva no monitor todos os números de três algarismos entre m e n, que verifiquem a propriedade: numero = soma do cubo dos algarismos.

Exemplo: 371 = 3 x 3 x 3 + 7 x 7 x 7 + 1 x 1 x 1.

Os valores m e n são indicados pelo utilizador e devem obedecer às seguintes condições:

$m < n$ $m \geq 100$ $n \leq 999$

Especificação ou análise do programa:

Dados de Entrada:

m (inteiro) – limite inferior
n (inteiro) – limite superior

Resultados Pretendidos:

Mostrar todos os algarismos entre m e n que sejam iguais à soma dos cubos dos seus dígitos

Processamento requerido:

Obter (m, n)
Extrair os dígitos dos vários números, fazendo divisões sucessivas por 10
Somar os cubos desses algarismos
Comparar essa soma com o número inicial
Mostrar todos os números que tornem a comparação verdadeira.

Pseudocódigo:

```
INÍCIO programa
        FAZER
            OBTEM (m)
        ENQUANTO (m<100 OU m>999)
        FAZER
            OBTEM (n)
        ENQUANTO (n>999 OU n<=m)
        i ← m
        ENQUANTO (i<=n) FAZER
            x ← i
            d1 ← resto da divisão entre x e 10
            x ← x / 10
            d2 ← resto da divisão entre x e 10
            d3 ← x / 10
            SE (d1*d1*d1 + d2*d2*d2 + d3*d3*d3 = i)
                MOSTRA (i)
            FIMSE
            i ← i+1
        FIM ENQUANTO
FIM programa
```

Codificação em C:

```
#include <stdio.h>

#define MIN     100
#define MAX     999

void main()
{
      int   m,n,i,d1,d2,d3,x;

      do
      {
            printf("Introduza o valor de m: ");
            scanf("%d",&m);
      }
      While (m<MIN || m>MAX);
      do
      {
            printf("Introduza o valor de n: ");
            scanf("%d",&n);
      }
      While (n>MAX || n<=m);
      for (i=m;i<=n;i++)
      {
            x=i;
            d1=x%10;
            x=i/10;
```

```
            d2=x%10;
            d3=x/10;
            if (d1*d1*d1+d2*d2*d2+d3*d3*d3==i)
                  printf("Numero %d\n",i);
      }
}
```

Exercício 9 – Ficha 4

Desenvolva um programa que determine qual o máximo de uma sequência de números inteiros não negativos introduzidos pelo utilizador (utilize o valor zero para assinalar o fim da sequência). Além de indicar o número máximo, o programa deve especificar em que posição da sequência o máximo apareceu e quantos números tinha a sequência.

Exemplo: para a sequência de inteiros 2 5 3 6 8 1 2 0, o programa deverá escrever:

Maximo: 8 Surgiu na posicao 5 Tamanho da sequencia: 7

Especificação ou análise do programa:

Dados de Entrada:

n1, n2, …, n (inteiro) – Sequencia de números inteiros não negativos terminada por 0

Resultados Pretendidos:

maior (inteiro) – Maior dos valores introduzidos
posicao (inteiro) – Posição do maior dos valores na sequência
tam (inteiro) – Tamanho da sequencia

Processamento requerido:

Fazer o procedimento abaixo enquanto n foi positivo:
Obter o próximo número da sequência
Se o primeiro número da sequência for positivo inicializar processo (tam ← 1, maior ← n, posicao ← tam)
Comparar maior com os seguintes números da sequência, guardar o maior e a posição (que será o tamanho actual da sequência)
Incrementar tam à medida que os valores forem sendo introduzidos

Pseudocódigo:

```
INÍCIO programa
      tam ← 0
      FAZER
            OBTEM (n)
            SE (n>0) ENTAO
                  tam ← tam+1
                  SE (tam=1) ENTAO
                        maior ← n
                        posicao ← tam
                  SENAO
                        SE (n>maior) ENTAO
                              maior ← n
                              posicao ← tam
                        FIM SE
```

FIM SE
FIM SE
ENQUANTO (n≠0)
MOSTRA (maior, posicao, tam)
FIM programa

Codificação em C:

```
#include <stdio.h>
void main()
{
	int	n,tam=0,posicao,maior;
	do
	{
		fflush(stdin);
		printf("Introduza o numero:");
		scanf("%d",&n);
		if (n>0)
		{
			tam++;
			if (tam==1)
			{
				maior=n;
				posicao=tam;
			}
			else
				if (n>maior)
				{
					maior=n;
					posicao=tam;
				}
		}
	}
	while(n!=0);
	printf("\nMax: %d. Posic %d. Tamanho: %d\n",maior,posicao,tam);
}
```

Exercício 10 – Ficha 4

Desenvolva um programa que leia um conjunto de números reais introduzidos pelo utilizador. Espera-se que o utilizador introduza os números por ordem crescente. A leitura deve terminar quando esta regra for violada. No final da introdução, o programa deve indicar a soma e a média dos números correctamente introduzidos.

Exemplo: para a sequência de números -2.4 -1.5 3 7.8 12.2 8.9 (repare que o número 8.9 não deverá ser considerado, uma vez que não está na ordem correcta), o programa deverá escrever:

Soma: 19.10
Media: 3.82

Especificação ou análise do programa:

Dados de Entrada:

n (real) – Sequência de números reais em ordem crescente

Resultados Pretendidos:

soma (real) – Soma dos números correctamente introduzidos
media (real) – Média dos números correctamente introduzidos

Processamento Requerido:

Obter os números da sequência, ir somando-os e contando-os, à medida que são obtidos, enquanto o último número introduzido for menor que o anterior
Calcular média dos números introduzidos na ordem correcta
Mostrar (soma, media)

Pseudocódigo:

```
INÍCIO programa
    OBTEM (num)
    soma ← num
    n ← 1
    FAZER
        ult ← num
        OBTEM (num)
        SE (num > ult) ENTAO
            soma ← soma+num
            n ← n + 1
        FIM SE
    ENQUANTO (num > ult)
    media ← soma/n
    MOSTRA (soma, media)
FIM programa
```

Codificação em C:

```c
#include <stdio.h>

void main()
{
	float	num,ult,soma,media;
	int	n=1;

	printf("Introduza numero: ");
	scanf("%f",&num);
	soma=num;
	do
	{
		fflush(stdin);
		ult=num;
		printf("Introduza numero: ");
		scanf("%f",&num);
		if (num>ult)
		{
			soma+=num;
			n++;
		}
	} while(num>ult);
	media=soma/n;
	printf("\nSoma - %.2f  Media - %.2f\n\n",soma,media);
}
```

Ou, de outra forma,

```
#include <stdio.h>
#include <limits.h>

void main()
{
	float	num,ult,soma=0,media;
	int	n=0;

	ult=-(float)INT_MAX;
	while (1)
	{
		fflush(stdin);
		printf("Introduza numero: ");
		scanf("%f",&num);
		if (num<=ult)
			break;
		ult=num;
		n++;
		soma+=num;
	}
	media=soma/n;
	printf("\nSoma - %.2f  Media - %.2f\n\n",soma,media);
}
```

Ou, ainda,

```
#include <stdio.h>

void main()
{
	float	num,ult,soma=0,media;
	int	n=0;

	do
	{
		fflush(stdin);
		if (n>1)
			ult=num;
		printf("Introduza numero: ");
		scanf("%f",&num);
		if (n==0)
			ult=num-1;
		if (num>ult)
		{
			soma+=num;
			n++;
		}
	}while(num>ult);
	media=soma/n;
	printf("\nSoma - %.2f  Media - %.2f\n\n",soma,media);
}
```

Exercício 11 – Ficha 4

Pretende-se efectuar o cálculo das notas finais dos 12 alunos de um curso de Programação com 50 aulas. A nota de cada um dos alunos é obtida através da média aritmética de duas provas que efectuou ao longo do ano. Se o aluno faltou a mais do que 25% das aulas reprova automaticamente.

Desenvolva um programa que leia os valores das notas e o número de faltas para cada um dos alunos e, após isso, calcule a informação seguinte:

- n.º de alunos aprovados
- n.º de alunos reprovados
- Média das notas da turma (os alunos reprovados por faltas não devem ser contabilizados para o cálculo da média).

Especificação ou análise do programa:

Dados de Entrada:
n1, n2 (real) – Notas das provas
faltas (inteiro) – Número de faltas

Resultados Pretendidos:
aprov (inteiro) – Número de alunos aprovados
reprov (inteiro) – Número de alunos reprovados
media (real) – Média das notas dos alunos não reprovados por faltas

Processamento requerido:
Obter (faltas, n1, n2) para os 12 alunos
Calcular a nota final para os alunos que não reprovaram por faltas
Contar os alunos aprovados
Contar os alunos reprovados
Calcular a média das notas dos alunos que não reprovaram por faltas
Mostrar (aprov, reprov, media)

Pseudocódigo:

```
INÍCIO programa
    n ← 0
    media ← 0
    aprov ← 0
    reprov ← 0
    ENQUANTO (n < 12) FAZER
        FAZER
            OBTEM (faltas)
        ENQUANTO (faltas<0 OU faltas>50)
        SE (faltas<=13) ENTAO
            OBTEM (n1, n2)
            nota ← (n1+n2)/2
            media ← media + nota
            SE (nota >=9.5) ENTAO
                aprov ← aprov + 1
            SENAO
                reprov ← reprov + 1
            FIM SE
```

SENAO
MOSTRA (“Reprovado por faltas”)
FIM SE
n←n+1
FIM ENQUANTO
media ← media/(aprov+reprov)
reprov ← 12 – aprov
MOSTRA (aprov, reprov, media)
FIM programa

Codificação em C:

```
#include <stdio.h>

void main()
{
	int		faltas,n,aprov,reprov;
	float		n1,n2,nota,media;

	media=0;
	aprov=0;
	reprov=0;
	for (n=0;n<12;n++)
	{
		do
		{
			fflush(stdin);
			printf("\nNumero de faltas do %d aluno: ",n+1);
			scanf("%d",&faltas);
		}while (faltas<0 || faltas>50);
		if (faltas<=13)
		{
			fflush(stdin);
			printf("\tNota da 1 prova (0 a 20 valores): ");
			scanf("%f",&n1);
			fflush(stdin);
			printf("\tNota da 2 prova (0 a 20 valores): ");
			scanf("%f",&n2);
			nota=(n1+n2)/2;
			media+=nota;
			if (nota>=9.5)
				aprov++;
			else
				reprov++;
		}
		else
			printf("Reprovado por faltas!\n");
	}
	media/=(aprov+reprov);
	reprov=12-aprov;
	printf("\nAlunos aprovados - %d",aprov);
	printf("\nAlunos reprovados - %d",reprov);
	printf("\nMedia dos nao reprovados por faltas - %.2f\n",media);
}
```

Exercício 14 – Ficha 4

Considere o seguinte problema:

Qual é o menor número inteiro positivo, tal que, se retirarmos o algarismo das unidades e o colocarmos do lado esquerdo, obtemos um número 4 vezes maior.

Desenvolva um programa que resolva o problema proposto.

Especificação ou análise do programa:

Dados de Entrada:

Não tem

Resultados Pretendidos:

n (inteiro) – Menor inteiro positivo, tal que, se retirarmos o algarismo das unidades e o colocarmos do lado esquerdo, obtemos um número 4 vezes maior.

Processamento Requerido:

i ← 0

Fazer as operações abaixo enquanto não se encontrar o número desejado:

Obter o último algarismo de i (usar o resto da divisão inteira por 10)
conta ← número de algarismos de i
novonum ← último dígito * $10^{(conta-1)}$ + i/10 (divisão inteira)
incrementar i

Pseudocódigo:

INÍCIO programa
i ← 0
FAZER
i ← i+1
digito ← resto da divisão entre i e 10
r ← i / 10 (divisão inteira)
conta ← 1
ENQUANTO (r/conta) > 0) FAZER
conta ← conta * 10
FIM ENQUANTO
num ← digito * conta + r
ENQUANTO (num ≠ 4*i)
MOSTRA (i)
FIM programa

Codificação em C:

```
#include <stdio.h>

void main()
{
      long int   i=0,digito,r,conta,num;

      do
      {
            i++;
            digito=i%10;
            r=i/10;
            conta=1;
            while (r/conta>0)
```

```
                conta=conta*10;
            num=conta*digito+r;
        }while (num!=4*i);
        printf("\nNumero pretendido - %ld (4 * %ld = %ld)\n",i,i,num);
}
```

Ou, de outra forma,

INÍCIO programa
    i ← 0
    ENQUANTO (VERDADE)
        i ← i+1
        digito ← resto da divisão entre i e 10
        r ← i / 10 (divisão inteira)
        conta ← 1
        ENQUANTO (r/conta) > 0) FAZER
            conta ← conta + 1
        FIM ENQUANTO
        num ← digito * 10^{conta} + r
        SE (num = 4*i)
            MOSTRA (i)
            SAI DO CICLO
        FIM SE
    FIM ENQUANTO
FIM programa

Codificação em C:

```
#include <stdio.h>
#include <math.h>

void main()
{
        long int   i,j,num,digito;

        for (i=1;;i++)
        {
                digito=i%10;
                num=i/10;
                for (j=1;(num/=10)!=0;j++);
                num=digito*(long int)pow(10,j)+i/10;
                if (num==4*i)
                {
                        printf("\nNumero - %ld (4 * %ld = %ld)\n",i,i,num);
                        break;
                }
        }
}
```

– Algoritmos com Funções –

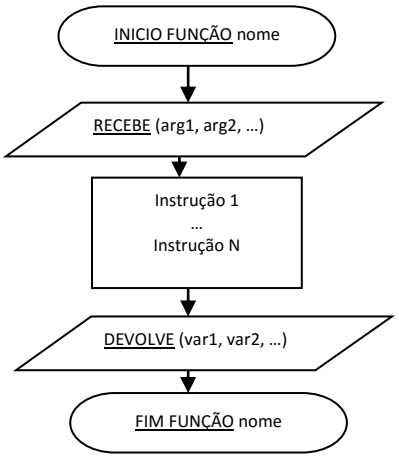
As funções:

- Permitem executar uma acção mais complexa sobre um conjunto de dados de modo a obter um conjunto de resultados;
- Servem para agrupar, debaixo de um só nome, um conjunto e instruções para que estas possam ser invocadas em diversas alturas sem ter de reescrever o código;
- Permitem simplificar a escrita/depuração de programas;
- Devem ter um nome sugestivo, associado à tarefa que se propõe fazer (mas não é obrigatório que o tenham).

Pseudocódigo: ***Fluxograma***

INICIO FUNÇÃO nomeFunção
RECEBE (arg1, arg2, …)
    Instrução 1
    …
    Instrução N
    DEVOLVE (var1, var2, …)
FIM FUNÇÃO nomeFunção

…
CHAMA nomeFunção (arg1, arg2, …)
CHAMA nomeFunção ()
<var1, var2, …> CHAMA nomeFunção ()
<var1, var2, …> CHAMA nomeFunção (arg1, arg2, …)
…



Exemplo: Programa que calcule o valor da expressão $\frac{n!}{(n-p)!p!}$

Especificação ou análise do problema:

Dados de Entrada:
n (inteiro) - Argumento da expressão
p (inteiro) - Argumento da expressão
Resultados Pretendidos:
v (real) - Valor da expressão
Processamento requerido:
Obter (n, p), garantindo que n>=p e que p>=0
v ← valor de $\frac{n!}{(n-p)!p!}$ (calculado com base numa função para obter factoriais)
Mostrar (v)

Especificação ou análise da função factorial:

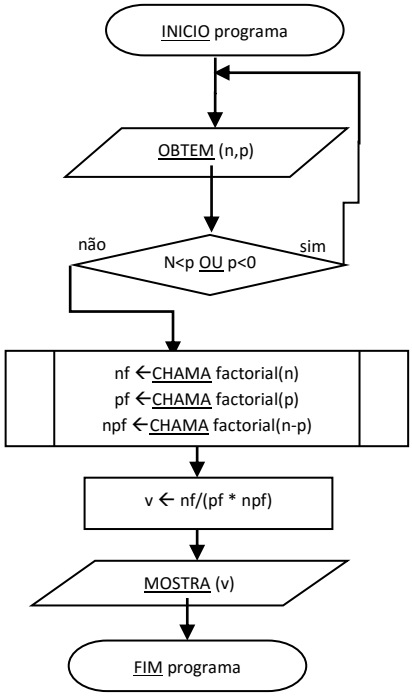
Dados de Entrada:
a (inteiro) - Número a calcular o factorial
Resultados Pretendidos:
f (inteiro) – Factorial de a
Processamento requerido:
f ← 1 * … * a, se a>0 e f ← 1, se a=

Pseudocódigo:

```
INÍCIO FUNÇÃO Factorial
RECEBE(a)
        f ← 1
        i ← 2
        ENQUANTO (i<=a) FAZER
                f ← f*i
                i ← i+1
        FIM ENQUANTO
        DEVOLVE (f)
FIM FUNÇÃO Factorial

INÍCIO Progama
        FAZER
                OBTEM (n, p)
        ENQUANTO (n<p OU p<0)
        nf ← CHAMA factorial(n)
        pf ← CHAMA factorial(p)
        npf ← CHAMA factorial(n-p)
        v ← nf/(pf * npf)
   MOSTRA (v)
FIM Programa
```

Fluxograma:



As funções:

Recebem parâmetros

Devolvem parâmetros

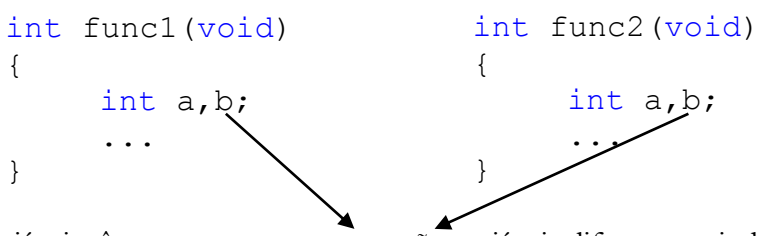
```
int factorial(int a)
{
        int   f,i;
        f=1;
        for (i=2;i<=a;i++)
              f=f*i;
        return f;
}
```

Podem não receber parâmetros

Podem não devolver parâmetros

```
void main()
{
        int n=5,m;
        m=factorial(n);
        printf("%d",m);
}
```

- O valor passado por parâmetro para a função factorial é 5. Isto é, a variável *a* da função factorial fica com o valor 5. A variável m vai ficar com o valor devolvido da função factorial, ou seja, a variável *f* da função factorial.
- As variáveis declaradas dentro do corpo de uma função só existem (ou só são visíveis) aí durante a sua execução. São chamadas de **variáveis locais ou automáticas**:
    - Apenas podem ser usadas na função onde é declarada;

- Quando a função é chamada, é automaticamente criado um espaço em memória para armazenar cada uma das variáveis deste tipo. Esse espaço é libertado quando a função termina a sua execução;
- Quando a função for novamente chamada, as suas variáveis são reinicializadas. Não retêm o valor com que ficaram na altura em que a função terminou;
- 

As variáveis têm o mesmo nome mas são variáveis diferentes e independentes.

- Os argumentos de entrada das funções (parâmetros que são recebidos) têm as mesmas propriedades que as variáveis locais;
- Colocando a palavra-chave ***static*** antes da declaração de uma variável local, faz-se com que ela passe a ter um armazenamento permanente. É chamada de **variável estática**:
  - Este tipo de variável deixa de ser inicializada sempre que se chamar a função, isto é, começa com o valor com que ficou da última vez que a função foi executada;

```
int func4(void)
{
  static int n=8;
  n++;
  return n;
}
```

    - Na primeira vez que esta função for chamada, *n* tem o valor de 8. Na próxima vez que esta função for chamada *n* tem o valor 9.
  - Estas variáveis existem até ao programa terminar. Isto é, apenas quando o programa terminar é que o seu espaço de memória é libertado;
  - Têm as mesmas propriedades que as variáveis locais:
    - São sempre declaradas dentro das funções;
    - São visíveis apenas dentro do bloco onde são declaradas.

Variáveis externas ou globais

- São declaradas fora do corpo de qualquer função e têm um armazenamento permanente;
- Estas variáveis são visíveis a partir do ponto da sua declaração. Por outras palavras, podem ser usadas em qualquer parte do programa que esteja a seguir à sua declaração.

```
int a=4;
int func3(void)
{
    int x=a;
    ...
}
```

Protótipos de Funções

- Na linguagem C, todas as funções devem estar implementadas no código antes da posição onde serão usadas. Caso não estejam, deverão ser declarados os seus protótipos no início do programa;
- O protótipo de uma função consiste na definição dos tipos dos seus parâmetros de entrada e saída, ou seja,

```
Tipo da varOut Nome_função(tipo da var1,…, tipo da varn);
```

- Actualmente é considerado uma boa prática declarar os protótipos de todas as funções do programa.

Exemplo

```
#include <stdio.h>
int   X=10;
int F1(int);
void main()
{
      int   x;
      x=F1(0);
      printf("\nGlobal = %d\tLocal = %d\n\n",X,x);
      X=F1(0);
      printf("\nGlobal = %d\tLocal = %d\n\n",X,x);
}
int F1(int i)
{
      static int X=1;
      int        x=1;
      x=2*i-x;
      X=2*i-X;
      printf("Local Static=%d Apenas Local=%d",X,x);
      return x;
}
```

Exercício 39 – Ficha 1

Identifique o resultado do algoritmo seguinte:

```
INÍCIO FUNÇÃO F1
RECEBE(n)
      x ← 0
      MOSTRA(x, n)
      x <- x+1
      MOSTRA(x, n)
DEVOLVE( )
FIM FUNÇÃO F1

INÍCIO Principal
      i ← 2
      ENQUANTO i ≤ 4 FAZER
            MOSTRA(i)
            CHAMA F1(i)
            i ← i + 1
      FIM ENQUANTO
      MOSTRA(i)
FIM Principal
```

Mostra os seguintes valores:

2
0 2
1 2
3
0 3
1 3
4
0 4
1 4
5

Principal	Função F1	
i	**x**	**N**
2		
	0	2
	1	
3		
	0	3
	1	
4		
	0	4
	1	
5		

Exercício 40 – Ficha 1

Identifique a funcionalidade implementada pelo algoritmo:

```
INÍCIO FUNÇÃO f2
RECEBE(n)
    SE  n ≥ 0 ENTÃO
            res ← 1
            ENQUANTO n > 1 FAZER
                    res ← res * n
                    n ← n - 1
            FIM ENQUANTO
    SE NÃO
            res ← 0
    FIM SE
DEVOLVE(res )
FIM FUNÇÃO  f2

INÍCIO principal
    i ← 5
    ENQUANTO i ≥ -5 FAZER
        x ← CHAMA f2(i)
        MOSTRA(i, x)
        i ← i - 1
    FIM ENQUANTO
FIM principal
```

Mostra os seguintes valores:

1º → 5 120 3º → 3 6 5º → 1 1 7º → -1 0 9º → -3 0 11º → -5 0
2º → 4 24 4º → 2 2 6º → 0 1 8º → -2 0 10º → -4 0

Exercício 41 – Ficha 1

Desenvolva os algoritmos:

a) de uma função que permita calcular o índice de massa corporal (IMC) de uma pessoa. O IMC de um indivíduo é obtido dividindo-se o seu peso (em kg) pela sua altura (em metros) ao quadrado. Assim, por exemplo, uma pessoa de 1,67 m e pesando 55 kg possui um IMC igual a 19,72 (IMC = 55 kg/ (1,67 m * 1,67 m) = 19,72)

b) de um programa que receba o peso e a altura de uma pessoa e que, chamando a função da alínea a), calcule o seu IMC. De acordo com os valores de IMC obtidos deve escrever uma das mensagens seguintes:

Até 18,5 inclusive: escrever " Abaixo do peso normal";
De 18,5 a 25 inclusive: escrever "Peso normal";
De 25 a 30 inclusive: escrever "Acima do peso normal";
Acima de 30: escrever "Obesidade".

a) ***Especificação ou análise da função:***

Dados de Entrada:
p (real) - Peso em kg
a (real) - Altura em metros
Resultados Pretendidos:
v (real) - Índice de massa corporal
Processamento requerido:
Recebe (p, a)
v ← p/(a*a)
Devolve (v)

Pseudocódigo:

```
INÍCIOFUNÇÃO IMC
RECEBE (p, a)
    v ← p/(a*a)
    DEVOLVE (v)
FIM FUNÇÃO IMC
```

b) ***Especificação ou análise do problema:***

Dados de Entrada:
p (real) - Peso em kg
a (real) - Altura em metros
Resultados Pretendidos:
m (string) - Mensagem sobre o peso
Processamento requerido:
Obter (p, a)
Calcular o IMC com a respectiva função
Se IMC até 18,5 inclusive: escrever "Abaixo do peso normal"
Se IMC entre 18,5 a 25 inclusive: escrever "Peso normal"
Se IMC entre 25 a 30 inclusive: escrever "Acima do peso normal"
Se IMC acima de 30: escrever "Obesidade"

Pseudocódigo:

```
INÍCIO progama
    OBTEM (p, a)
    i ← CHAMA IMC(p, a)
    SE (i<=18.5) ENTAO
            m ← "Abaixo do peso normal"
    SENAO
            SE (i<=25) ENTAO
                    m ← "Peso normal"
            SENAO
                    SE (i<=30) ENTAO
                            m ← "Acima do peso normal"
                    SENAO
                            m ← "Obesidade"
                    FIM SE
            FIM SE
    FIM SE
    MOSTRA (m)
FIM programa
```

Codificação em C:

```
#include <stdio.h>

float IMC(float,float);

void main()
{
      float peso,altura,indice;

      printf("Qual o peso: ");
      scanf("%f",&peso);
      printf("\nQual a altura: ");
      scanf("%f",&altura);
      indice=IMC(peso,altura);
      if (indice<=18.5)
            printf("\n\nAbaixo do peso normal\n\n");
      else
            if (indice<=25)
                  printf("\n\nPeso normal\n\n");
            else
                  if (indice<=30)
                        printf("\n\nAcima do peso normal\n\n");
                  else
                        printf("\n\nObesidade\n\n");
}

float IMC(float p,float a)
{
      float v;

      v=p/(a*a);
      return v;
}
```

Exercício 45 – Ficha 1

Realize as seguintes tarefas:

a) Desenvolva uma função que devolva o número de divisores de um valor inteiro positivo, passado como argumento.

b) Escreva o algoritmo de um programa que leia uma sequência de números inteiros. Sempre que nessa sequência surgir um número primo, deve ser escrita no monitor a seguinte mensagem: "O número que introduziu é primo!". O programa termina quando for introduzido um número negativo ou nulo.

a) ***Especificação ou análise da função:***

Dados de Entrada:
n (inteiro) - Número a testar
Resultados Pretendidos:
c (inteiro) - Número de divisores que tem
Processamento requerido:
Recebe (n)
c ← 0

Testar cada um dos números i (entre 1 e n) e ver se o resto de n por i dá 0.
Em caso afirmativo, c ← c + 1
Devolve (c)

Pseudocódigo:

```
INÍCIOFUNÇÃO conta_div
RECEBE (n)
    c ← 0
    i ← 1
    ENQUANTO (i<=n) FAZER
        SE (Resto_da_Divisão entre n e i = 0) ENTÃO
                c ← c + 1
        FIM SE
        i ← i + 1
    FIM ENQUANTO
    DEVOLVE (c)
FIM FUNÇÃO conta_div
```

Codificação em C:

```
int conta_div(int n)
{
      int   c=0,i=1;

      while (i<=n)
      {
            if (n%i==0)
                  c++;
            i++;
      }
      return c;
}
```

b) ***Especificação ou análise do problema:***

Dados de Entrada:
n (inteiro) - Número a testar
Resultados Pretendidos:
m (string) - Mensagem a indicar se o número é primo
Processamento requerido:
Vai obtendo um número e enquanto esse número for positivo:
Se o número de divisores for 2 ou 1 (caso o número seja 1), usando a função conta_div, escrever a mensagem

Pseudocódigo:

```
INÍCIO programa
      FAZER
            OBTEM (n)
            SE (n>0) ENTAO
                  x ← CHAMA conta_div (n)
                  SE (x=2 OU x=1) ENTAO
                        MOSTRA ("O número que introduziu é primo!")
                  FIM SE
            FIM SE
```

ENQUANTO (n>0)
FIM programa

Codificação em C:

```
#include <stdio.h>

void main()
{
	int	num,x;

	do
	{
		printf("Numero: ");
		scanf("%d",&num);
		if (num>0)
		{
			x=conta_div(num);
			if (x==2 || x==1)
				printf("\tNumero primo\n");
		}
	}
	while (num>0);
}
```

Exercício 1 – Ficha 5

Desenvolva uma função que devolva o quadrado de um número real passado como argumento.

Escreva um programa que, utilizando a função definida no ponto anterior, leia números e imprima os seus quadrados até que seja introduzido o valor zero.

Especificação da função:

Dados de Entrada:
n (real) – Número para fazer o cálculo
Resultados Pretendidos:
q (real) – O quadrado de n
Processamento Requerido:
Recebe (n)
q ← n * n
Devolve (q)

Pseudocódigo:

INÍCIO FUNÇÃO qd
RECEBE (n)
q ← n * n
DEVOLVE(q)
FIM FUNÇÃO qd

Especificação do programa:

Dados de Entrada:
n (real) – Número para fazer o cálculo

Resultados Pretendidos:

q (real) – O quadrado de n

Processamento Requerido:

Obter (n), chama qd(n), retornando o resultado q e mostrar q enquanto n for ≠ 0

Pseudocódigo:

INÍCIO Programa
    FAZER
        OBTEM (n)
        SE (n ≠ 0) ENTAO
            q ← CHAMA qd(n)
            Mostra (q)
        FIM SE
    ENQUANTO (n ≠ 0)
FIM Programa

Codificação em C:

```c
#include <stdio.h>

float qd(float n)
{
	float q;

	q = n*n;
	return q;
}
void main()
{
	float n;

	do
	{
		printf("\nIntroduza um numero: ");
		scanf("%f",&n);
		if (n!=0)
			printf("\nQuadrado de %f: %f\n",n,qd(n));
	}
	while (n!=0);
}
```

Exercício 2 – Ficha 5

Desenvolva uma função que verifique se um número inteiro está entre dois limites (também inteiros). Os três valores são passados como argumento. A função deve devolver 1, se o número estiver dentro dos limites, ou 0, caso contrário.

Especificação ou análise da função:

Dados de Entrada:

inf, sup (inteiro) – Limite superior e inferior
n (inteiro) – Número que se pretende testar

Resultados Pretendidos:

estado (inteiro) { 1 se inf <= n <= sup
0 se não

Processamento Requerido:

Receber (inf, sup, n)
Verificar se o número está entre os limites
Devolver (1 se o número estiver dentro dos limites, ou 0, caso contrário)

Pseudocódigo:

INÍCIO FUNÇÃO Verifica
RECEBE (inf, sup, n)
    estado ← 0
    SE (n ≥ inf E n ≤ sup) ENTÃO
        estado ← 1
    FIM SE
    DEVOLVE (estado)
FIM FUNÇÃO Verifica

Codificação em C:

```c
#include <stdio.h>

int verifica(int,int,int);

void main()
{
      int   inf,sup,n;

      printf("Introduza o limite inferior: ");
      scanf("%d",&inf);
      printf("\nIntroduza o limite superior: ");
      scanf("%d",&sup);
      printf("Introduza o numero a testar: ");
      scanf("%d",&n);
      if (verifica(inf,sup,n))
            printf("\nDentro dos limites!\n\n");
      else
            printf("\nFora dos limites!\n\n");
}

int verifica(int inf,int sup,int n)
{
      if (n>=inf && n<=sup)
            return 1;
      return 0;
}
```

Exercício 7 – Ficha 5

Desenvolva uma função que desenhe no monitor um triângulo de números, invertido. O número de linhas é passado como argumento da função. É garantido que o seu valor é superior a 1 e inferior ou igual a 9. Na figura pode ver-se um exemplo para um triângulo com 5 linhas. No desenho do triângulo, em cada uma das linhas o valor do número aumenta até à coluna central e a partir daí diminui.

```
1 2 3 4 5 4 3 2 1
  1 2 3 4 3 2 1
    1 2 3 2 1
      1 2 1
        1
```

Especificação ou análise da função:

Dados de Entrada:

n (inteiro) – Número entre 2 e 9

Resultados Pretendidos:

Mostrar um triângulo invertido na forma, formado por números

Processamento Requerido:

Recebe (n)

nEsp ← 0

Enquanto n for diferente de 0 deve fazer:

- Escrever nEsp espaços em branco no ecrã, todos os números de 1 a n e todos os números de n-1 a 1
- Mudar de linha
- Incrementar nEsp e decrementar n

Pseudocódigo:

```
INÍCIO FUNÇÃO triangulo
RECEBE (n)
        nEsp ← 0
        ENQUANTO (n ≥1) FAZER
                j ← 0
                ENQUANTO (j < nEsp) FAZER
                        MOSTRA (“ ”)
                        j ← j+1
                FIM ENQUANTO
                j ← 1
                ENQUANTO (j ≤ n) FAZER
                        MOSTRA (j)
                        j ← j+1
                FIM ENQUANTO
                j ← n-1
                ENQUANTO (j ≥ 1) FAZER
                        MOSTRA (j)
                        j ← j-1
                FIM ENQUANTO
                MOSTRA (mudança de linha)
                nEsp ← nEsp+1
                n ← n-1
        FIM ENQUANTO
        DEVOLVE()
FIM FUNÇÃO triangulo
```

Codificação em C:

```
#include <stdio.h>

void triangulo(int n)
{
        int i,j,nEsp=0;

        for (i=n;n>=1; n--,nEsp++)
        {
                for (j=0;j<nEsp;j++)
                        printf(" ");
```

```
            for (j=1;j<=n;j++)
                  printf("%d",j);
            for (j=n-1;j>=1;j--)
                  printf("%d",j);
            printf("\n");
      }
}

void main()
{
      int   n;

      do
      {
            fflush(stdin);
            printf("\nQual o numero de linhas (2 a 9): ");
            scanf("%d",&n);
      }while (n<=1 || n>9);
      triangulo(n);
}
```

Exercício 9 – Ficha 5

Desenvolva uma função que devolva o inteiro que mais se aproxima da média de dois reais positivos que são passados como parâmetro.

Nota: Partir do princípio que não existe nenhuma função de arredondamento em C.

Especificação ou análise da função:

Dados de Entrada:

x1, x2 (reais) – Números positivos

Resultados Pretendidos:

n (inteiro) – Número inteiro que mais se aproxima da média entre x1 e x2

Processamento Requerido:

Obtem (x1,x2)

media ← $\frac{x1+x2}{2}$

Se parte decimal de media < 0.5, n ← parte inteira de media

Senão, n ← parte inteira de media + 1

Devolve (n)

Pseudocódigo:

INÍCIO FUNÇÃO f1

RECEBE (x1,x2)

media ← $\frac{x1+x2}{2}$

SE (Parte_decimal(media) ≥ 0.5) ENTÃO

DEVOLVE (Parte_inteira(media)+1)

SENÃO

DEVOLVE (Parte_inteira(media))

FIMSE

FIM FUNÇÃO f1

Codificação em C:

```
#include <stdio.h>

int f1(float x1,float x2)
{
      int   mediainteira;
      float media;

      media=(x1+x2)/2;
      mediainteira=(int)media;
      if (media-mediainteira >=0.5)
            return mediainteira+1;
      else
            return mediainteira;
}

void main()
{
      printf("Com 18.4 e 20.0,numero: %2d\n",f1(18.4,20));
      printf("Com 2.7 e 2.7,numero: %2d\n\n",f1(2.7,2.7));
}
```

Exercício 10 – Ficha 5

Desenvolva as funções especificadas nas alíneas seguintes:

a) Função que devolva o cubo de um número inteiro e positivo passado como argumento;

b) Função que obtenha do utilizador um número inteiro compreendido entre 100 e 999. O valor obtido deve ser devolvido como resultado final;

c) Função que verifique se um determinado número inteiro N, recebido como argumento, obedece à seguinte propriedade: N é igual à soma do cubo dos seus algarismos. Um exemplo de um número que satisfaz esta propriedade é o $371 = 3^3 + 7^3 + 1^3$. É garantido que o valor recebido como argumento é um número com três dígitos. A função deve devolver 1 se a propriedade se verificar, ou 0, no caso contrário;

d) Construa um programa que obtenha do utilizador vários números inteiros pertencentes ao intervalo [100, 999] e verifique quais os que satisfazem a propriedade indicada na alínea c). A introdução de números termina quando o utilizador assim o desejar.

a)

Especificação da função:

Dados de Entrada:
n (inteiro) – Número positivo que se pretende calcular o cubo
Resultados Pretendidos:
r (inteiro) – n^3
Processamento Requerido:
Recebe (n)

```
        r ← n * n * n
        Devolve (r)
```

Pseudocódigo:

```
INÍCIO FUNÇÃO cubo
RECEBE (n)
        r ← n * n * n
        DEVOLVE (r)
FIM FUNÇÃO cubo
```

b)

Especificação da função:

Dados de Entrada:

Não tem ou n (inteiro) – Número entre 100 e 999

Resultados Pretendidos:

n (inteiro) – Número entre 100 e 999

Processamento Requerido:

Obter (n) tal que n ∈ [100, 999]. Caso não esteja, repetir a obtenção

Pseudocódigo:

```
INÍCIO FUNÇÃO ler
RECEBE ()
        FAZER
                OBTEM (n)
        ENQUANTO (n<100 OU n>999)
        DEVOLVE (n)
FIM FUNÇÃO ler
```

c)

Especificação da função:

Dados de Entrada:

n (inteiro) – Número de 3 dígitos a verificar

Resultados Pretendidos:

r (inteiro) – 1 ou 0, consoante n possua ou não a propriedade de ser igual à soma do cubo dos seus dígitos.

Processamento Requerido:

Receber (n)

Separar os dígitos por divisões sucessivas por 10 e verificar se a propriedade é ou não respeitada

Devolver (r)

Pseudocódigo:

```
INÍCIO FUNÇÃO verifica
RECEBE(n)
        du ← resto(n,10)
        aux ← quociente(n,10)
        dd ← resto(aux,10)
        dc ← quociente(aux,10)
        du3 ← CHAMA cubo(du)
        dd3 ← CHAMA cubo(dd)
```

```
        dc3 ← CHAMA cubo(dc)
        r ← 0
        SE (n=du3+dd3+dc3) ENTAO
                r ← 1
        FIM SE
        DEVOLVE (r)
FIM FUNÇÃO verifica
```

d)

Especificação do programa:

Dados de Entrada:

n (inteiro) – Números inteiros positivos a verificar

Resultados Pretendidos:

Mostrar ("verifica" ou "não verifica") consoante o número introduzido verificar a propriedade ou não

Processamento Requerido:

Enquanto o utilizador quiser, Chamar a função ler, Chamar a função verifica e Mostrar a mensagem correspondente

Pseudocódigo:

```
INÍCIO principal
        FAZER
                n ← CHAMA ler()
                r ← CHAMA verifica(n)
                SE (r=1) ENTAO
                        MOSTRA ("Verifica")
                SENAO
                        MOSTRA ("Não Verifica")
                FIM SE
                OBTEM(c)
        ENQUANTO (c = 's' OU c = 'S')
FIM principal
```

Codificação em C:

```c
#include <stdio.h>

int cubo(int n)
{
      r = n*n*n;
      return r;
}

int ler()
{
      int   num;

      do
      {
            fflush(stdin);
            printf("\nNumero entre [100,999]: ");
            scanf("%d",&num);
      }
```

```
        while (num<100 || num>999);
        return num;
}

int   verifica(int n)
{
        int   du,dd,dc,aux;

        du=n%10;
        aux=n/10;
        dd=aux%10;
        dc=aux/10;
        if (n==cubo(du)+cubo(dd)+cubo(dc))
              return 1;
        else
              return 0;
}

void main()
{
        int   n;
        char  c;

        do
        {
              n=ler();
              if (verifica(n)==1)
                    printf("\nVERIFICA\n");
              else
                    printf("\nNAO VERIFICA\n");
              fflush(stdin);
              printf("\nDeseja validar outro numero? (Sim -
              s ou S / Nao - outro qualquer caracter) ");
              c=getchar();
        }
        while (c=='S' || c=='s');
}
```

Exercício 11 – Ficha 5

Desenvolva uma função que leia um conjunto de números inteiros e devolva o número de vezes que o valor máximo surgiu. A dimensão da sequência é passada como argumento.

Especificação da função:

Dados de Entrada:

n (inteiro) – Tamanho da sequencia de números
num (inteiro) – Números da sequência

Resultados Pretendidos:

contan (inteiro) – Número de vezes que o valor máximo surgiu

Processamento Requerido:

Para cada um dos números da sequência:
Se for o primeiro número ou se o número for maior que o máximo então
max ← num e contan ← 1

Senão, se num for igual ao máximo então,
contan ← contan + 1

Pseudocódigo:

INÍCIO FUNÇÃO ctm
RECEBE(n)
i ← 1
ENQUANTO (i ≤ n) FAZER
OBTEM (num)
SE (i = 1 OU num > max) ENTÃO
max ← num
contan ← 1
SENAO
SE (num = max) ENTAO
contan ← contan + 1
FIM SE
FIM SE
i ← i + 1
FIM ENQUANTO
DEVOLVE(contan)
FIM FUNÇÃO ctm

Codificação em C:

```c
#include <stdio.h>

int ctm(int n)
{
	int  contan,num,max,i;

	for (i=1;i<=n;i++)
	{
		printf("Introduza o numero: ");
		scanf("%d",&num);
		if (i==1 || num>max)
		{
			max=num;
			contan=1;
		}
		else
			if (num==max)
				contan++;
	}
	return contan;
}

void main()
{
	printf("O numero maximo surgiu %d vezes\n",ctm(5));
}
```

Exercício 13 – Ficha 5

Pretende-se desenvolver um programa que permita fazer um pequeno jogo entre duas pessoas. O programa escolhe um número entre 0 e 500, que os dois jogadores tentam acertar alternadamente. De cada vez que um dos jogadores indica um número, o programa deve informar se:

- o número está abaixo da solução;
- o número está acima da solução;
- quando o jogador acertou.

O programa deve identificar os jogadores como sendo jogador1 e jogador2.

Neste programa vamos utilizar duas funções (além da função void main(void) que controlará todo o jogo):

- Função que pede uma nova aposta a um dos jogadores. Recebe, como argumento, o número do jogador) e devolve a sua nova aposta;
- Função que compara dois números passados como argumento. Se o primeiro argumento for menor do que o segundo deverá devolver -1. Se o segundo for menor do que o primeiro deverá devolver 1. Se ambos os argumentos forem iguais deverá devolver 0.

Após desenvolver estas funções, desenvolva o programa completo. No início, o programa deverá escolher um número aleatório entre 0 e 500. Após isso, vai perguntando a cada um dos jogadores as suas apostas. O programa termina quando um dos jogadores acertar no número.

Exemplo de uma fase do jogo:

```
E a vez do jogador 1!
Qual a sua aposta? 4
O valor 4 está abaixo do número certo
E a vez do jogador 2!
Qual a sua aposta? 18
O valor 18 está acima do número certo
E a vez do jogador 1!
Qual a sua aposta? 6
O valor 6 está abaixo do número certo
E a vez do jogador 2!
Qual a sua aposta? 9
O valor 9 está acima do número certo
E a vez do jogador 1!
Qual a sua aposta? 7
O jogador 1 acertou!
```

Especificação da função **PedeAposta:**

Dados de Entrada:

jogador (inteiro) – Número do jogador

Resultados Pretendidos:
ap (inteiro) – Aposta feita pelo utilizador n
Processamento Requerido:
Pedir a aposta do utilizador e devolvê-la

Pseudocódigo:

```
INÍCIO FUNÇÃO PedeAposta
RECEBE (jogador)
    OBTEM (ap)
    DEVOLVE (ap)
FIM FUNÇÃO PedeAposta
```

***Especificação da função* Compara:**

Dados de Entrada:
a, b (inteiro) – Números a comparar
Resultados Pretendidos:
r (inteiro) { -1 se a<b; 0 se a=b; 1 se a>b
Processamento Requerido:
Comparar os números e devolve o valor correspondente (r)

Pseudocódigo:

```
INÍCIO FUNÇÃO Compara
RECEBE (a, b)
    SE (a < b) ENTÃO
            r ← -1
    SENAO
            SE (a = b) ENTÃO
                    r ← 0
            SENÃO
                    r ← 1
            FIM SE
    FIM SE
    DEVOLVE (r)
FIM FUNÇÃO Compara
```

Especificação do programa:

Dados de Entrada:
aposta (inteiro) – Apostas dos jogadores (obtidos à vez)
Constantes:
MIN (= 0) – Valor mínimo para o número a adivinhar
MAX (= 500) – Valor máximo para o número a adivinhar
Resultados Pretendidos:
msg (mensagem) – Que indica se o valor da aposta está acima ou abaixo do pretendido ou se acertou
Processamento Requerido:

```
alvo ← número aleatório
j ← 1
Fazer
        alvo ← Chamar função PedeAposta(j)
        r ← Chamar função Compara(aposta, alvo)
```

Se (r = -1)
Mostrar a frase "Valor está abaixo do número certo!"
Senão
Se (r = 0)
Mostrar a frase "Acertou!"
Senão
Mostrar a frase "Valor acima do número certo!"
Troca de Jogador
Enquanto (r ≠ 0)

Pseudocódigo:

INÍCIO Programa
alvo ← numero aleatório
j ← 1
FAZER
aposta ← CHAMA FUNÇÃO PedeAposta(j)
r ← CHAMA FUNÇÃO Compara(alvo, aposta)
SE (r = -1) ENTÃO
MOSTRA ("Valor está acima do número certo!")
SENÃO
SE (r = 1) ENTÃO
MOSTRA ("Valor está abaixo do número certo!")
SENÃO
MOSTRA ("Acertou!")
FIM SE
FIM SE
SE (j = 1) ENTÃO
j ← 2
SENÃO
j ← 1
FIM SE
ENQUANTO (r ≠ 0)
FIM Programa

Codificação em C:

```
#include <stdio.h>
#include <time.h>

#define    MAX 500

int NumeroAleatorioInt(int limite)
{
      srand((unsigned)time(NULL));
      return (rand()%limite);
}

int pedeAposta(int jogador)
{
      int   ap;

      fflush(stdin);
      printf("\nQual a aposta do jogador %d: ",jogador);
      scanf("%d",&ap);
      return ap;
}
```

```
int compara(int a,int b)
{
	int	r;
	if (a<b)
		r=-1;
	else
		if (a==b)
			r=0;
		else
			r=1;
	return r;
}
void main()
{
	int	j=1,resultado,alvo,aposta;
	alvo=NumeroAleatorioInt(MAX);
	do
	{
		aposta=pedeAposta(j);
		resultado=compara(aposta,alvo);
		switch (resultado)
		{
			case -1:
				printf("O numero %d esta abaixo do
				numero certo\n",num);
				break;
			case 0:
				printf("O jogador %d acertou\n",j);
				break;
			case 1:
				printf("O numero %d esta acima do
				numero certo\n",num);
				break;
		}
		if (j==1)
			j=2;
		else
			j=1;
	}while (resultado!=0);
}
```

Exercício 16 – Ficha 5

Desenvolva uma função que receba um carácter passado como argumento, e, caso esse carácter seja uma vogal, devolva a sua ordem ('a', 'A' têm ordem 1, enquanto que 'u', 'U' têm ordem 5). Se o carácter não for uma vogal deve ser devolvido 0.

Nota: pode utilizar as funções existentes na biblioteca <ctype.h>.

Notas:

Funções que <ctype.h> tem

- int isalnum(int c) → 1 se c for uma letra ou um número

- int isalpha(int c) → 1 se c for uma letra
- int isdigit(int c) → 1 se c for um número
- int islower(int c) → 1 se c for uma letra minúscula
- int isupper(int c) → 1 se c for uma letra maiúscula
- int tolower(int c) → Transforma c em minúscula
- int toupper(int c) → Transforma c em maiúscula

Especificação da função:

Dados de Entrada:
c (caracter) – O carácter a testar
Resultados Pretendidos:
r (inteiro) – Ordem da vogal (a-1, e-2, etc.) ou 0 se não for vogal
Processamento Requerido:
Receber (c)
Comparar c com as vogais e atribuir o valor correspondente a r
Devolver (r)

Pseudocódigo:

```
INÍCIO FUNÇÃO vogal
RECEBE (c)
    TransformaParaLetraMinuscula(c)
    CASO (c)
        QUANDO ('a')
            r ← 1
        QUANDO ('e')
            r ← 2
        QUANDO ('i')
            r ← 3
        QUANDO ('o')
            r ← 4
        QUANDO ('u')
            r ← 5
        RESTO
            r ← 0
    FIM CASO
    DEVOLVE (r)
FIM FUNÇÃO vogal
```

Codificação em C:

```
#include <stdio.h>
#include <ctype.h>

int vogal(char c)
{
      c=tolower(c);
      switch(c)
      {
            case 'a':
                  return 1;
            case 'e':
                  return 2;
```

```
                case 'i':
                        return 3;
                case 'o':
                        return 4;
                case 'u':
                        return 5;
                default:
                        return 0;
        }
}

void main()
{
        printf("\n%d\n",vogal('e'));
        printf("\n%d\n",vogal('1'));
        printf("\n%d\n",vogal('U'));
        printf("\n%d\n",vogal('I'));
}
```

Exercício 17 – Ficha 5

Desenvolva uma função que leia uma palavra (terminada com '\n') e a reproduza no monitor, com todas as letras transformadas em maiúsculas. O programa deve mostrar o número de caracteres lidos.

Nota: não se pretende que utilize tabelas de nenhum tipo.

Especificação da função:

Dados de Entrada:

letra (caracter) – Conjunto de letras

Resultados Pretendidos:

letraM (caracter) – Conjunto de letras em maiúsculas

c (inteiro) – Mostrar a quantidade de letras que transformou

Processamento Requerido:

Fazer enquanto letra for diferente do carácter ENTER:

Obter (letra)

Ir contando os caracteres lidos

Converter letra em maiúsculas e mostrá-la

Devolver (número de caracteres lido, sem contar com o ENTER)

Pseudocódigo:

INÍCIO FUNÇÃO TfPalavra
RECEBE ()
conta ← 0
FAZER
OBTEM (letra)
conta ← conta + 1
letraM ← chanar TransformaParaLetraMaiuscula(letra)
MOSTRA (letraM)
ENQUANTO (letra ≠ ENTER)
DEVOLVE (conta-1)
FIM FUNÇÃO TfPalavra

Codificação em C:

```
#include <stdio.h>
#include <ctype.h>
```

```
int TfPalavra()
{
      char  letra,letraM;
      int   conta=0;
      printf("Introduza a palavra:\n");
      do
      {
            scanf("%c",&letra);
            conta++;
            letraM=toupper(letra);
            printf("%c",letraM);
      }
      while (letra!='\n');
      return conta-1;
}
void main()
{
      printf("\nForam lidos %d caracteres.",TfPalavra());
}
```

Exercício 20 – Ficha 5

Indique qual o resultado da execução dos seguintes problemas:

a)

```
#include <stdio.h>
void main(void)
{
      double      d;    //real 8 bytes
      float       f;    //real 4 bytes
      long int    l;    //inteiro 4 bytes
      int         i;    //inteiro 2 bytes
      i=l=f=d=100/3;
      printf("%d\t%ld\t%f\t%f\n",i,l,f,d);
      d=f=l=i=100/3;
      printf("%d\t%ld\t%f\t%f\n",i,l,f,d);
      i=l=f=d=100/3.0;
      printf("%d\t%ld\t%f\t%f\n",i,l,f,d);
      d=f=l=i=100/3.0;
      printf("%d\t%ld\t%f\t%f\n",i,l,f,d);
}
```

Resultado:

```
33      33      33.000000       33.000000
33      33      33.000000       33.000000
33      33      33.333332       33.333333
33      33      33.000000       33.000000
```

b)

```
#include <stdio.h>
void main(void)
{
      double      y=3.2,x;
      int         i=2,j;
      x=(j=y/i)*2;
```

```
        printf("%f\t%d\n",x,j);
        j=(x=y/i)*2;
        printf("%f\t%d\n",x,j);
        x=y*(j=((int)2.9+1.1)/y);
        printf("%f\t%d\n",x,j);
}
```

Resultado:

```
2.000000        1
1.600000        3
0.000000        0
```

Exercício 21 – Ficha 5

Para cada um dos seguintes programas, indique o resultado da sua execução:

a)

```
#include <stdio.h>

int   n=4;

void f1(void)
{
      printf("%d\n",n);
      n++;
      printf("%d\n",n);
}

void f2(void)
{
      int   n=-1;

      printf("%d\n",n);
      n=30;
      printf("%d\n",n);
}

void main(void)
{
      printf("%d\n",n);
      n=10;
      f1();
      printf("%d\n",n);
      f2();
      printf("%d\n",n);
}
```

Solução:

4
10
11
11
-1
30
11

b)

```
#include <stdio.h>

void f1(int n)
{
      static int x=0;
      int        y=0;

      printf("%d\t%d\t%d\n",x,n,y);
      x++;
      n++;
      y++;
```

Solução:

Iteracao 0:

0 0 0

Iteracao 1:

1 1 0

```
}
void main(void)
{
      int   i;
      for (i=0;i<2;i++)
      {
            printf("\nIteracao %d:\n",i);
            f1(i);
      }
   }
```

c)

```
#include <stdio.h>
int   i=1;
int start(void)
{
      return i;
}
int next(int j)
{
      j=i++;
      return j;
}
int last(int j)
{
      static int i=10;
      j=i--;
      return j;
}
void main(void)
{
      int   i,j;
      i=start();
      for (j=1;j<=3;j++)
      {
            printf("%d\t%d\n",i,j);
            printf("%d\n",next(i));
            printf("%d\n",last(i));
      }
}
```

Solução:

```
1   1
1
10
1   2
2
9
1   3
3
8
```

– Vetores ou *arrays* unidimensionais –

Como definir um array unidimensional, em linguagem C?

TipoDeDados NOME[Dimensão];

Esta definição cria um vetor unidimensional chamado NOME, que poderá guardar um número de valores do tipo TipoDeDados igual ao valor da sua Dimensão. Acede-se a esses valores através do respectivo índice, que, na linguagem C, vai de 0 a Dimensão-1.

Como se utilizam os valores do array, em linguagem C?

NOME[indice]

Como se faz uma atribuição de valores ao array, em linguagem C?

- No momento da definição:

  TipoDeDados NOME[ ]={valor0, valor1, …, valorN};

  TipoDeDados NOME[Dimensão]={ValorÚnicoParaTodos os elementos};
- A um dos elementos do array:

  NOME[indice]=valor0;

Exemplo:

int a[5] → O tamanho tem que ser constante

```
a[0]=12;
a[1]=9;
a[2]=14;
a[3]=5;
a[4]=1;
```

12	9	14	5	1

Indice 0

int a[] = {12, 9, 14, 5, 1}

```
for (i=0; i<5; i++)
      a[i]*=2;
```

24	18	28	10	2

Arrays unidimensionais como argumentos de função:

```
#include <stdio.h>
#define N   5
int * mostra(int v1[],int *v2,int tam)
{
      int i;
      static int r[N];
      for (i=0;i<tam;i++)
            r[i]=v1[i]+v2[i];
    return r;
}
void main()
{
      int a[]={12,9,14,5,1},i,*res;
      res=mostra(a,a,N); //Colocar só o nome do array
      for (i=0;i<N;i++)
            printf("%d ",res[i]);
}
```

Exercício 1 – Ficha 6

Desenvolva um programa que calcule a média dos N elementos de um array de inteiros. A inicialização dos elementos do array deverá ser feita pelo utilizador.

Especificação do programa:

Constantes:
N (inteiro) – Tamanho do conjunto de valores
Dados de Entrada:
n1, n2, …, nn (inteiro) – Valor dos elementos do vector
Resultados Pretendidos:
media (real) – Média dos N elementos do vector
Processamento Requerido:
Obter N valores e colocá-los no vector.
Calcular e mostrar a média dos elementos do vector

Pseudocódigo:

```
INÍCIO Programa
    i ← 0
    soma ← 0
    ENQUANTO (i<N) FAZER
        OBTEM (v[i])
        soma ← soma + v[i]
        i ← i + 1
    FIM ENQUANTO
    media ← soma/N
    MOSTRA (media)
FIM Programa
```

Codificação em C:

```c
#include <stdio.h>
#define N  5
void main()
{
      int   i,soma=0,v[N];
      float media;
      for (i=0;i<N;i++)
      {
            printf("Valor da posicao %d do vector: ",i);
            scanf("%d",&v[i]);
            soma+=v[i];
      }
      media=(float)soma/N;
      printf("\nMedia = %.2f\n\n",media);
}
```

Exercício 2 – Ficha 6

Complete o programa anterior de modo a que todos os elementos do array, cujo valor seja inferior à média, sejam colocados a zero.

Especificação do programa:

Constantes:
N (inteiro) – Tamanho do conjunto de valores

Dados de Entrada:

n1, n2, …, n (inteiro) – Valor dos elementos do vector

Resultados Pretendidos:

media (real) – Média dos N elementos do vector

n1, n2, …, nn (inteiro) – Valor dos elementos do vector actualizados

Processamento Requerido:

Obter N valores e colocá-los no vector.

Calcular e mostrar a média dos elementos do vector

Verificar quais os valores do vector que são menores que a média, colocá-los a zero e mostrar o vector actualizado

Pseudocódigo:

```
INÍCIO Programa
    soma ← 0
    i ← 0
    ENQUANTO (i<N) FAZER
        OBTEM (v[i])
        soma ← soma + v[i]
        i ← i + 1
    FIM ENQUANTO
    media ← soma/N
    MOSTRA (media)
    i ← 0
    ENQUANTO (i<N) FAZER
        SE (v[i]<media) ENTÃO
            v[i] ← 0
        FIM SE
        i ← i + 1
    FIM ENQUANTO
    MOSTRA (v)
FIM Programa
```

Codificação em C:

```
#include <stdio.h>
#define N  5
void main()
{
      int   i,soma=0,v[N];
      for (i=0;i<N;i++)
      {
            printf("Valor da posicao %d do vector: ",i);
            scanf("%d",&v[i]);
            soma+=v[i];
      }
      printf("\nMedia:%.2f\nV atualizado:",(float)soma/N);
      for (i=0;i<N;i++)
      {
            if (v[i]<(float)soma/N)
                  v[i]=0;
            printf("%d, ",v[i]);
      }
}
```

Exercício 3 – Ficha 6

Um array pode servir para representar conjuntos de números inteiros. Considerando que o domínio é [0, 9], um array de 10 elementos pode indicar quando é que cada um dos números pertence ou não a um determinado conjunto. Para isso, basta colocar o valor de a[i] a 0 se o elemento i não fizer parte do conjunto e a[i] a 1, se i fizer parte do conjunto.

O array

1	0	1	1	0	1	1	1	0	0

indica que os elementos 0, 2, 3, 5, 6, 7 fazem parte do conjunto especificado pelo array.

Desenvolva um programa que, após obter e armazenar dois conjuntos A, B com domínio [0, 9], calcule as seguintes operações: A ∩ B (intersecção de A com B), A ∪ B (reunião de A com B) e A – B (elementos de A que não pertencem a B).

Especificação do programa:

Dados de Entrada:

v1, v2 (vector de inteiros) – Vectores que representam os conjuntos A e B

Resultados Pretendidos:

I, R, D (vector de inteiros) – Resultado das operações A ∩ B, A ∪ B e A – B

Processamento Requerido:

Obter (v1, v2)

Calcular a operação A ∩ B, A ∪ B e A – B

Mostrar (I, R, D)

Pseudocódigo:

```
INÍCIO Programa
    v1 e v2 ← 0
    i ← 1
    ENQUANTO (i<=2) FAZER
            FAZER
                    FAZER
                            OBTER (n)
                    ENQUANTO (n>9 OU n<0)
                    SE (i=1) ENTÃO
                            v1[n] ← 1
                    SENÃO
                            v2[n] ← 1
                    FIM SE
                    Mostra ("Terminar?")
                    OBTER (c)
            ENQUANTO (c='N')
            i ← i + 1
    FIM ENQUANTO
    MOSTRA ("V1 – V2 – Intersecção – Reunião – Diferença")
    i ← 0
    ENQUANTO (i < 10) FAZER
            I[i] ← v1[i] E v2[i]
            R[i] ← v1[i] OU v2[i]
            D[i] ← v1[i]-v1[i]*v2[i]        ou      D[i] ← v1[i] E (NOT v2[i])
            MOSTRA (V1[i] – V2[i] – I[i] – R[i] – D[i])
            i ← i + 1
    FIM ENQUANTO
FIM Programa
```

Codificação em C:

```
#include <stdio.h>
#include <ctype.h>

void main()
{
      int   i,n,A[10]={0},B[]={0,0,0,0,0,0,0,0,0,0};
      int   I[10]={0},R[10]={0},D[10]={0};
      char  c;

      for (i=1;i<=2;i++)
      {
            printf("\nGrupo %d (valores entre 0 e 9)",i);
            do
            {
                  do
                  {
                        printf("\nNumero: ");
                        scanf("%d",&n);
                  }while (n>9 || n<0);
                  if (i==1)
                        A[n]=1;
                  else
                        B[n]=1;
                  printf("\nTerminar? (s/n) ");
                  fflush(stdin);
                  scanf("%c",&c);
            }while (toupper(c)=='N');
      }
      printf("\nA\tB\tInterseccao\tReuniao\tDiferenca\n");
      for (i=0;i<10;i++)
      {
            I[i]=A[i] && B[i];
            R[i]=A[i] || B[i];
            D[i]=A[i]-A[i]*B[i];
            printf("%d\t%d\t%d\t\t%d\t%d\n",A[i],B[i],I[i]
            ,R[i],D[i]);
      }
}
```

Exercício 5 – Ficha 6

Desenvolva um programa que verifique se um determinado número inteiro positivo é capicua. O número a verificar é especificado pelo utilizador (um número é capicua se for "simétrico", como por exemplo os números 12321 e 694496).

Especificação do programa:

Dados de Entrada:

num (inteiro) – Número a testar

Resultados Pretendidos:

Mostrar ("Capicua!!") se o número introduzido é capicua

Processamento Requerido:

Obter (num)

Separar cada um dos algarismos do número e guardá-los num vector

Verificar se o número é capicua e se é, Mostrar ("Capicua!!")

Pseudocódigo:

INÍCIO Programa
FAZER
OBTER (num)
ENQUANTO (num<=0)
tam ← 0
ENQUANTO (num>0)
algarismos[tam] ← Resto da divisão entre num e 10
num ← num / 10
tam ← tam + 1
FIM ENQUANTO
capicua ← 1
i ← 0
j ← tam-1
FAZER
SE (algarismos[i]≠algarismos[j])
capicua ← 0
i ← tam
FIM SE
i ← i + 1
j ← j - 1
ENQUANTO (i≤(tam-1)/2)
SE (capicua=1)
MOSTRA ("Capicua!!!")
FIM SE
FIM Programa

Codificação em C:

```c
#include <stdio.h>

void main()
{
	int	num,algarismos[100],tam=0,i=0,j,capicua;

	do
	{
		printf("Introduza numero: ");
		scanf("%d",&num);
	}while (num<=0);
	while (num>0)
	{
		algarismos[tam++]=num%10;
		num/=10;
	}
	capicua=1;
	j=tam-1;
	do
	{
		if (algarismos[i]!=algarismos[j])
		{
			capicua=0;
			i=tam;
		}
		i++;
		j--;
```

```
        }while (i<=(tam-1)/2);
        if (capicua)
              printf("\nCapicua!!!\n\n");
}
```

Exercício 6 – Ficha 6

Defina uma função que receba como argumentos um array de inteiros e a sua dimensão e devolva a soma de todas as ocorrências do maior número do array.

Exemplo:

Array: 1 2 8 8 1 8	**devolve : 24**
Array: 2 2 2 2 2 2	**devolve : 12**
Array: 1 2 3 2 -1 0	**devolve : 3**

Usando a função faça um programa que:

- Declare um array de inteiros de tamanho TAM.
- Peça ao utilizador valores inteiros para preencher o array.
- Chame a função anterior enviando como argumentos o array e a sua dimensão.
- Imprima o valor devolvido pela função.

Especificação da função F1:

Dados de Entrada:
vector (inteiro) – Vector de inteiros
tam (inteiro) – Tamanho do vector

Resultados Pretendidos:
soma (inteiro) – Soma de todas as ocorrências do maior número do vector

Processamento Requerido:
Receber (vector,tam)
Encontrar o maior número do vector
Contar quantas vezes esse número aparece no vector
soma ← maior número * número de vezes que o maior número aparece
Devolver (soma)

Especificação ou análise do problema:

Dados de Entrada:
elemento (inteiro) – Elementos do vector

Resultados Pretendidos:
valor (inteiro) – valor devolvido pela função F1

Processamento Requerido:
Preencher um vector com valores pedidos ao utilizador
Calcula a soma de todas as ocorrências do maior número do vector através da função F1
Mostrar (valor)

Pseudocódigo:

INÍCIO FUNÇÃO F1
RECEBE (vector, tam)
maior ← 1º elemento do vector
conta ← 1

```
        i ← 2ª posição dentro do vector
        ENQUANTO (i < tam) FAZER
            SE (maior < elemento do vector da posição i)
                maior ← elemento do vector da posição i
                conta ← 1
            SENAO
                SE (maior = elemento do vector da posição i)
                    conta ← conta +1
                FIM SE
            FIM SE
            i ← próxima posição dentro do vector
        FIM ENQUANTO
        soma ← maior * conta
        DEVOLVE (soma)
    FIM FUNÇÂO F1

    INÍCIO Programa
        pos ← 1ª posição dentro do vector
        FAZER
            OBTER (elemento)
            vector[pos] ← elemento
            pos ← próxima posição dentro do vector
        ENQUANTO (pos < tamanho do vector)
        valor ← CHAMA F1(vector, tamanho do vector)
        MOSTRA (valor)
    FIM Programa
```

Codificação em C:

```c
#include <stdio.h>

#define TAM     6

int   F1(int vector[],int tam)
{
      int   i,maior,conta=1;

      maior=vector[0];
      for (i=1;i<tam;i++)
      {
            if (maior<vector[i])
            {
                  maior=vector[i];
                  conta=1;
            }
            else
                  if (maior==vector[i])
                        conta++;
      }
      return maior*conta;
}

void main()
{
      int   v1[TAM],i;

      for (i=0;i<TAM;i++)
      {
```

```
            printf("\nQual o elemento %d do vector: ",i);
            scanf("%d",&v1[i]);
        }
        printf("\n%d\n\n",F1(v1,TAM));
}
```

Exercício 7 – Ficha 6

Considere um array com N componentes inteiros.

a) Desenvolva uma função que devolva a posição do maior dos elementos do array. No caso de haver repetições do valor no array, a função pode, por exemplo, fornecer como resultado a posição com maior índice.
b) Desenvolva uma função que desloque todos os seus elementos, uma posição para a direita. O último elemento deve deslocar-se para a primeira posição.
c) Elabore um programa que, após a leitura das N componentes inteiras, faça as necessárias rotações para a direita até que o elemento de maior valor do array se encontre na última posição desse array.

a) ***Especificação da função:***

Dados de Entrada:

v (inteiro) – Array de inteiros
tam (inteiro) – Tamanho do array

Resultados Pretendidos:

pos (inteiro) – Posição (índice no array) do maior dos elementos do array

Processamento Requerido:

Receber (v, tam)
Encontrar o maior elemento do array e guardar a posição
Devolver (pos)

Pseudocódigo da função:

INÍCIO FUNÇÃO MaiorElemento
Recebe (v,tam)
    maior ← 1º elemento do vector
    pos ← 0
    i ← 2ª posição dentro do vector
    ENQUANTO (i<tam) FAZER
        SE (maior <= elemento do vector da posição i)
            maior ← elemento do vector da posição i
            pos ← i
        FIM SE
        i ← próxima posição dentro do vector
    FIM ENQUANTO
    DEVOLVE (pos)
FIM FUNÇÃO MaiorElemento

Codificação em C:

```
int   MaiorElemento(int vector[],int tam)
{
      int   i,maior,pos=0;

      maior=vector[0];
      for (i=1;i<tam;i++)
      {
```

```
            if (maior<=vector[i])
            {
                  maior=vector[i];
                  pos=i;
            }
      }
      return pos;
}
```

b) ***Especificação da função:***

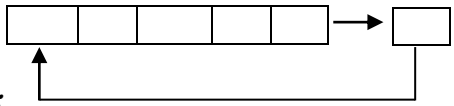
Dados de Entrada:

v (inteiro) – Vector de inteiros cujos elementos se quer deslocar.
tam (inteiro) – Tamanho do vector

Resultados Pretendidos:

v (inteiro) – vector com o conteúdo deslocado 1 posição para a direita

Processamento Requerido:



Pseudocódigo da função:

INÍCIO FUNÇÃO roda
Recebe (v, tam)
aux ← elemento de v da posição (tam-1)
i ← tam-2
ENQUANTO (i≥0) FAZER
elemento de v da posição (i+1) ← elemento de v da posição i
i ← i - 1
FIM ENQUANTO
elemento de v da posição 0 ← aux
FIM FUNÇÃO roda

Codificação em C:

```
void roda(int v[],int tam)
{
      int   i,aux;

      aux=v[tam-1];
      for (i=tam-2;i>=0;i--)
            v[i+1]=v[i];
      v[0]=aux;
}
```

c) ***Especificação do programa:***

Dados de Entrada:

v (inteiro) – Vector de inteiros

Constantes:

N – Tamanho do vector

Resultados Pretendidos:

v (inteiro) – Vector de inteiros atualizado

Processamento Requerido:

vector com o elemento maior deslocado até à sua última posição, usando as funções feitas anteriormente.

Pseudocódigo do programa:

INÍCIO Programa
OBTER (v)
FAZER
i ← CHAMA MaiorElemento(v,N)
SE (i ≠ N-1)
CHAMA roda(v,N)
FIM SE
ENQUANTO (i ≠ N-1)
MOSTRA (v)
FIM Programa

Codificação em C:

```
#include <stdio.h>

#define N  5

void roda(int *,int);
int MaiorElemento(int *,int);

void main()
{
      int   i,v[]={1,2,5,4,3};

      printf("\nVector inicial: ");
      for (i=0;i<N;i++)
            printf("%d ",v[i]);
      printf("\n");
      do
      {
            i=MaiorElemento(v,N);
            if (i!=N-1)
                  roda(v,N);
      }
      while (i!=N-1);
      printf("\nVector final: ");
      for (i=0;i<N;i++)
            printf("%d ",v[i]);
      printf("\n\n");
}
```

Exercício 8 – Ficha 6

Os elementos de um array dizem-se ordenados em pirâmide se o seu valor aumentar até uma certa posição e a partir daí diminuir. Dois possíveis exemplos são:

{1, 2, 4, 5, 6, 3, 2, -23, -23, -120} ou {1, 20, 19, 18, 17, 16, 15, -100, -101, -102}

a) Declare um array local à função main() com capacidade para armazenar DIM_TAB números inteiros (sendo DIM_TAB uma constante simbólica);

b) Considere que o *array* declarado na alínea anterior já foi completamente preenchido com números inteiros ordenados em pirâmide. Desenvolva uma função que receba o *array* (e respectiva dimensão) por argumento e escreva no monitor os valores dos seus elementos, por ordem decrescente.

Codificação em C:

```
#include <stdio.h>
#define DIM_TAB    10
void Decrescente(int v[],int tam)
{
    int i,j=0,maior;
    maior=v[j];
    while (maior<=v[j])
    {
        maior=v[j];
        j++;
    }
    i=j-1;
    printf("Elementos do vector por ordem decrescente: ");
    while (i>=0 || j<tam)
    {
        if (i==-1)
        {
            printf("%d ",v[j]);
            j++;
        }
        else
        {
            if (j==tam)
            {
                printf("%d ",v[i]);
                i--;
            }
            else
            {
                if(v[i]>=v[j])
                {
                    printf("%d ",v[i]);
                    i--;
                }
                else
                {
                    printf("%d ",v[j]);
                    j++;
                }
            }
        }
    }
}
void main()
{
    int v1[DIM_TAB]={1,2,4,5,6,3,2,-23,-23,-120};
    Decrescente(v1,DIM_TAB);
}
```

Exercício 10 – Ficha 6

Numa determinada escola irá decorrer um referendo sobre a proibição, ou não, de fumar no recinto da mesma. Com o objectivo de realizar uma sondagem sobre o resultado da votação, um grupo de docentes resolveu efectuar um pequeno inquérito anónimo a N alunos, escolhidos criteriosamente. Assim cada aluno tinha unicamente que indicar, com uma cruz, uma das seguintes opções, como resposta à pergunta "É a favor da proibição de fumar no recinto da escola?": **SIM, NÃO, ESTOU INDECISO, VOU ABSTER-ME.**

Desenvolva um programa que auxilie os docentes na obtenção dos resultados da sondagem. Assim, para cada um dos N inquéritos, o programa deve pedir ao utilizador um carácter indicativo da resposta do aluno, sendo 'S' para a resposta SIM, 'N' para NÃO, 'I' para INDECISO e 'A' para VOU-ME ABSTER.

No caso do aluno não ter assinalado nenhuma opção, ou mais do que uma, o carácter a introduzir pelo utilizador deverá ser 'X'. No final da introdução dos dados, o programa deve indicar qual será, previsivelmente, o resultado do referendo e, para além disso, disponibilizar os totais de cada uma das opções sob um formato gráfico, recorrendo à função definida na alínea anterior.

Exemplo de execução:

```
 Indique total de inquéritos: 50
--->1º Inquérito
-----> Opção escolhida (S/N/I/A/X): N
--->2º Inquérito
-----> Opção escolhida (S/N/I/A/X): S
--->3º Inquérito
-----> Opção escolhida (S/N/I/A/X): S
(...)
--->50º Inquérito
 -----> Opção escolhida (S/N/I/A/X): I
Segundo a sondagem a opção vendedora será o SIM.
Resultados das várias opções:
SSSSSSSSSSSSSSSSSS
NNNNNNNNN
IIIIIIIIIII
AAAAAA
XXX
```

Nota: O programa deverá fazer a validação dos dados introduzidos pelo utilizador.

Codificação em C:

```
#include <stdio.h>

#define T 100

void main ()
{
    int     ti,j,s=0,n=0,i=0,a=0,x=0,vitoria;
```

```
char    r[T],sim[T],nao[T],indeciso[T],abster[T],nulos[T];
printf("Indique total de inqueritos: ");
scanf("%d",&ti);
fflush(stdin);
for(j=1;j<=ti;j++)
{
    printf("%d inquerito:\nOpcao escolhida (S/N/I/A): ",j);
    gets(resposta);
    if ((strcmp(r,"s")==0) || (strcmp(r,"S")==0))
        sim[s++]='S';
    else
        if ((strcmp(r,"n")==0) || (strcmp(r,"N")==0))
            nao[n++]='N';
        else
            if ((strcmp(r,"i")==0) || (strcmp(r,"I")==0))
                indeciso[i++]='I';
            else
                if ((strcmp(r,"a")==0) || (strcmp(r,"A")==0))
                    abster[a++]='A';
                else
                    nulos[x++]='X';
}
if (s>n && s>i && s>a)
    vitoria=1;
else
    if (n>s && n>i && n>a)
        vitoria=2;
    else
        if (i>s && i>n && i>a)
            vitoria=3;
        else
            if (i>s && i>n && i>a)
                vitoria=4;
            else
                vitoria=5;
printf("\n\nSegundo a sondagem a opcao vendedora ");
switch (vitoria)
{
    case 1:
        printf("sera o SIM\n");
        break;
    case 2:
        printf("sera o NAO\n");
        break;
    case 3:
        printf("serao os INDECISOS\n");
        break;
    case 4:
        printf("sera a ABSTENCAO\n");
        break;
    case 5:
        printf("serao os NULOS\n");
        break;
}
printf("Resultados das varias opcoes:\n");
if (s>0)
```

```
    {
        for(j=0;j<s;j++)
            printf("%c",sim[j]);
        printf("\n");
    }
    if (n>0)
    {
        for(j=0;j<n;j++)
            printf("%c",nao[j]);
        printf("\n");
    }
    if (i>0)
    {
        for(j=0;j<i;j++)
            printf("%c",indeciso[j]);
        printf("\n");
    }
    if (a>0)
    {
        for(j=0;j<a;j++)
            printf("%c",abster[j]);
        printf("\n");
    }
    if (x>0)
    {
        for(j=0;j<x;j++)
            printf("%c",nulos[j]);
        printf("\n");
    }
}
```

– Strings –

As ***strings*** são arrays unidimensionais de caracteres. A sua particularidade é a de que são sempre terminadas pelo caracter \0.

- ***Definição:***

  char NOME[Dimensão];

- ***Atribuição:***
  - No momento da definição:

    char NOME[ ]=”palavra ou frase”;

    Os caracteres da palavra ou da frase são guardados nas diferentes posições da variável NOME. No final é colocado o caracter \0.
  - A um dos elementos do array:

    NOME[indice]=’caracter’;

    Nesta situação, não se pode esquecer de colocar o caracter \0 na última posição.
  - Usando funções especiais:

    strcpy(NOME,” palavra ou frase”);

    A função, depois da palavra ou frase coloca o caracter \0.

Notas:

- Quando as *strings* trabalham com frases, não é possível usar a instrução

  ```
  scanf("%s",str);
  ```

  pois a atribuição à variável termina logo que encontra um espaço. Sendo usada, a variável ficaria apenas com uma palavra.
- Para resolver esta situação, pode-se utilizar os [ ] no `scanf` que especificam os caracteres de entrada possíveis.

  Exemplo:

  `scanf("%[ABC]",str)` → Lê apenas caracteres ABC maiúsculos

  `scanf ("%[^\n]",str)` → Lê tudo até encontrar mudança de linha
- Existem uma série de funções para trabalhar com as *strings*. Dessas destacam-se:
  - strcat – Junta duas *strings* numa só;
  - strcmp – Compara duas strings. Retorna 0 se forem iguais;
  - strcpy – Copia uma string para outra;
  - strlen – Obtém o número de caracteres que a *string* tem, sem contar o \0;
  - gets – Obtém uma *string* via teclado;
  - puts – Mostra uma *string* no ecrã.

Exercício 1 – Ficha 7

Desenvolva um programa que leia uma frase introduzida pelo utilizador e a escreva invertida.

Exemplo: Hoje e Domingo!

O programa deverá escrever:

!ognimoD e ejoH

Especificação da programa:

Dados de Entrada:

frase (string) – Array de caracteres terminada por \0

Resultados Pretendidos:

Frase com os caracteres por ordem inversa.

Processamento Requerido:

Obtém (frase)

ncar ← número de caracteres de frase (tamanho da string)

Mostrar os caracteres um a um de trás para a frente, isto é, começando no carácter da posição ncar-1, ir decrementando o índice dessa posição até chegar ao índice 0.

Pseudocódigo:

```
INÍCIO Programa
    OBTER (frase)
    i ← tamanho(frase)-1
    ENQUANTO (i≥0) FAZER
        MOSTRA (frase[i])
        i ← i - 1
    FIM ENQUANTO
FIM Programa
```

Codificação em C:

```c
#include <stdio.h>

# define   TAM   100

void main()
{
      char  frase[TAM];
      int   i;

      printf("Introduza uma frase:\n");
      gets(frase);
      printf("\nFrase em ordem inversa:\n");
      for (i=((int)strlen(frase))-1;i>=0;i--)
           printf("%c",frase[i]);
      printf("\n");
}
```

Exercício 2 – Ficha 7

Desenvolva uma função, cujo protótipo seja **int contaPrimeiro(char[])**, que receba por argumento uma string e devolva o número de vezes que o caracter inicial (o primeiro caracter da frase que não seja um espaço em branco) surge ao longo da frase. O programa não deve distinguir entre letras maiúsculas e minúsculas.

Exemplo: ContaPrimeiro(" Amanha nao e Domingo!");

Deve devolver o valor: 4

Especificação da função:

Dados de Entrada:

frase (string) – Array de caracteres terminada por '\0'

Resultados Pretendidos:

Contador (inteiro) – Nº de vezes que o 1º caracter não seja espaço ou TAB aparece (em maiúsculas ou minúsculas). Se só existirem caracteres espaço ou TAB, devolve 0.

Processamento Requerido:

Encontrar 1º caracter que não seja espaço ou TAB
Converter para esse carácter para maiúsculas (ou minúsculas)
contador ← 0
Percorrer a string, comparando a versão em maiúscula (ou minúscula) de cada um dos seus caracteres com o 1º carácter:
    Se forem iguais incrementar o contador
Devolver o contador

Pseudocódigo:

```
INÍCIO FUNÇÃO ContaPrimeiro
RECEBE (frase)
    i ←0
    ENQUANTO ((frase[i]=' ' OU frase[i]='\t') E frase[i]≠'\0') FAZER
        i ← i + 1
    FIM ENQUANTO
    caracter ← Maiuscula(frase[i])
    contador ← 0
    ENQUANTO (frase[i]≠'\0') FAZER
        SE (caracter = Maiuscula(frase[i])) ENTÃO
            contador ← contador + 1
        FIM SE
        i ← i + 1
    FIM ENQUANTO
    DEVOLVE (contador)
FIM FUNÇÃO ContaPrimeiro
```

Codificação em C:

```c
#include <stdio.h>
#include <ctype.h>

# define TAM    1000

int contaPrimeiro(char *str)
{
      char  caracter;
      int   i,contador;

      for (i=0;((str[i]!='\0') && (isspace(str[i]) ||
str[i]=='\t'));i++);
      caracter=toupper(str[i]);
      contador=0;
      for (;str[i]!='\0';i++)
            if (caracter==toupper(str[i]))
                  contador++;
      return contador;
}

void main()
{
      char string[TAM];

      printf("Introduza a string: ");
      gets(string);
      printf("\nAparece %d vezes",contaPrimeiro(string));
}
```

Exercício 3 – Ficha 7

Desenvolva um programa que leia uma frase introduzida pelo utilizador e escreva cada uma das palavras que constituem a frase numa linha separada. Considere que as palavras estão separadas por um ou mais espaços em branco, podendo também existir espaços no início e fim da frase.

Exemplo: Hoje e Domingo!

O programa deverá escrever:

Hoje

e

Domingo!

Especificação do programa:

Dados de Entrada:
str (string) – Array de caracteres terminada por '\0'

Resultados Pretendidos:
Palavras que constituem a frase em linhas separadas

Processamento Requerido:
Obtém (frase)
Enquanto não for fim de frase:
Encontrar próximo caracter diferente de espaço
inicio ← guarda essa posição
Encontrar próximo caracter igual a espaço
fim ← guarda essa posição
Mostrar os caracteres da frase posicionados entre inicio e fim

Mostrar o carácter ENTER

Pseudocódigo:

```
INÍCIO Programa
    OBTER (str)
    i ← 0
    ENQUANTO (str[i]≠'\0') FAZER
        ENQUANTO ((str[i]=' ' OU str[i]='\t') E str[i]≠'\0') FAZER
            i ← i + 1
        FIM ENQUANTO
        ENQUANTO (str[i] ≠' ' E str[i] ≠'\t' E str[i]≠'\0') FAZER
            MOSTRAR (str[i])
            i ← i + 1
        FIM ENQUANTO
        MOSTRAR ('\n')
    FIM ENQUANTO
FIM Programa
```

Codificação em C:

```c
#include <stdio.h>

# define TAM    1000

void main()
{
      char  str[TAM];
      int   i=0;

      printf("Introduza a string: ");
      gets(str);
      while (str[i]!='\0')
      {
            for (;(str[i]==' ' || str[i]=='\t') &&
            str[i]!='\0';i++);
            for (;str[i]!=' ' && str[i]!='\t' &&
            str[i]!='\0';i++)
                  printf("%c",str[i]);
            printf("\n");
      }
}
```

Exercício 4 – Ficha 7

Desenvolva um programa que leia uma frase introduzida pelo utilizador e verifique quantas vezes a primeira palavra se repete. O programa não deve distinguir entre letras maiúsculas e minúsculas.

Exemplo: `Ter ou nao ter`

`O programa deverá escrever:`

`A palavra *Ter* repete-se 2 vezes.`

Especificação do programa:

Dados de Entrada:

f (string) – Array de caracteres terminada por '\0'

Resultados Pretendidos:

p (string) – Palavra inicial da frase

rep (inteiro)– Número de vezes que a palavra aparece na frase

Processamento Requerido:

Obtém (f)

Encontrar a primeira palavra e guardar em p

Contar quantas vezes a palavra aparece e mostrar

Pseudocódigo:

```
INÍCIO Programa
    OBTER (f)
    i ← 0
    d ← 0
    conta ← 0
    rep ← 1
    p[0] ← '\0'
    ENQUANTO (f[i] = ' ' OU f[i] = '\t') FAZER
            i ← i + 1
    FIM ENQUANTO
    SE (f[i] ≠ '\0')
            ENQUANTO (f[i] ≠ ' ' E f[i] ≠'\t' E f[i] ≠ '\0') FAZER
                    p[d] ← minúscula(f[i])
                    d ← d +1
                    i ← i +1
            FIM ENQUANTO
            p[d] ← '\0'
            SE (f[i] ≠ '\0')
                    ENQUANTO (i < tamanho de f) FAZER
                            ENQUANTO (f[i] = ' ' OU f[i] = '\t') FAZER
                                    i ← i +1
                            FIM ENQUANTO
                            SE (f[i] ≠ '\0')
                                    SE (minúscula(f[i]) = p[0])
                                            d ← 0
                                            conta ← 0
                                            ENQUANTO
(minúscula(f[i])=p[d] E f[i]≠'\0') FAZER
                                                    conta ← conta +1
                                                    i ← i +1
                                                    d ← d +1
                                            FIM ENQUANTO
                                            SE (conta=tamanho de p E (f[i]=' '
OU f[i]=TAB OU f[i]='\0'))
                                                    rep ← rep +1
                                            FIM SE
                                    SENAO
                                            ENQUANTO (f[i]≠' ' E f[i]≠'\t') FAZER
                                                    i ← i +1
                                            FIM ENQUANTO
                                    FIM SE
                            FIM SE
                            i ← i +1
                    FIM ENQUANTO
            FIM SE
```

FIM SE
SE (tamanho de p ≠ 0)
MOSTRAR (p,rep)
SENAO
MOSTRAR ("Não introduziu nenhuma palavra.")
FIM SE
FIM Programa

Codificação em C:

```
#include <stdio.h>
void main()
{
      char  f[100],p[100];
      int   i=0,d=0,conta=0,r=1;
      printf("\nFrase: ");
      gets(f);
      p[0]='\0';
      for (i=0;(f[i]==' ' || f[i]=='\t') &&
f[i]!='\0';i++);
      if (f[i]!='\0')
      {
            for (;f[i]!=' ' && f[i]!='\t' &&
f[i]!='\0';i++)
                  p[d++]=tolower(f[i]);
            p[d]='\0';
            if (f[i]!='\0')
            {
                  for (;i<(int)strlen(f);i++)
                  {
                        for (;(f[i]==' ' || f[i]=='\t') &&
f[i]=='\0';i++);
                        if (f[i]!='\0')
                        {
                              if (tolower(f[i])==p[0])
                              {
                                    conta=0;
                                    d=0;
                                    while
(tolower(f[i])==p[d] && f[i]!='\0')
                                    {
                                          conta++;
                                          i++;
                                          d++;
                                    }
                                    if (conta==strlen(p) &&
(f[i]==' ' || f[i]=='\t' || f[i]=='\0'))
                                          r++;
                              }
                              else
                                    for (;(f[i]!=' ' &&
f[i]!='\t') && f[i]=='\0';i++);
                        }
                  }
            }
      }
```

```
        if (strlen(p)!=0)
            printf("A palavra %s repete-se %d vezes",p,r);
}
```

Ou, de outra forma (esquecendo os TABs)

```
#include <stdio.h>
void main()
{
    int i=0,j=0,ct=1;
    char f[100],p1[100],pi[100];

    printf("Digite a frase: ");
    gets(f);
    for (;(f[i]==' ') && (f[i]!='\0');i++);
    for (;(f[i]!=' ') && (f[i]!='\0');i++,j++)
        p1[j]=f[i];
    while (f[i]!='\0')
    {
        for(;(f[i]==' ') && (f[i]!='\0');i++);
        for(j=0;(f[i]!=' ') && (f[i]!='\0');i++,j++)
            pi[j]=f[i];
        pi[j]='\0';
        if (strcmp(p1,pi)==0)
            ct+=1;
    }
    printf("a palavra '%s' foi repetida %d vezes",p1,ct);
}
```

Exercício 5 – Ficha 7

a) Desenvolva uma função que insira uma palavra no meio de uma frase. A frase é um conjunto de palavras separadas por um ou mais espaços. A função deve receber por argumento uma frase, uma palavra e um valor inteiro positivo. Este valor indica qual é a posição da tabela frase a partir da qual a nova palavra deve ser inserida.

Por exemplo:

Se função for chamada com o valor 16 e duas tabelas (`frase` e `palavra`) com o conteúdo:
`"A Pizzaria Pepe serve pizzas muito quentes." E "Verde",`
Na primeira tabela (`frase`) deve ficar guardada a seguinte informação:
`"A Pizzaria Pepe Verde serve pizzas muito quentes."`

b) Teste a função anterior. Tanto a frase como a palavra deverão estar armazenadas em tabelas de caracteres locais ao main() com a seguinte declaração:

```
#define TAM_FRASE 150
#define TAM_PALAVRA 20
        ...
void main()
{
        char frase[TAM_FRASE], palavra[TAM_PALAVRA];
        ...
}
```

A frase armazenada é um conjunto de palavras separadas por um ou mais espaços. Considere que, quando a função a desenvolver for chamada, as duas estruturas já devem estar inicializadas (respectivamente com uma frase e uma palavra, terminadas com '\0').

a) ***Especificação da função*:**

Dados de Entrada:

frase (string) – Frase a alterar
palavra (string) – Palavra a colocar no meio da frase
indice (int) – Posição na frase a partir de onde se coloca a palavra

Resultados Pretendidos:

f (string) – Frase inicial alterada

Processamento Requerido:

Receber (frase,palavra,indice)
Copiar para uma *string* todos os caracteres da frase até à posição indice
Juntar à *string* todos os caracteres da palavra
Juntar à *string* um espaço
Juntar à *string* os caracteres da frase desde a posição indice até ao fim
Copiar para a frase todos os caracteres da string auxiliar usada

Codificação em C:

```
void InserePlvra(char frase[],char palavra[],int indice)
{
      char  str[TAM_FRASE];
      int   i,j;

      for (i=0;i<indice;i++)
            str[i]=frase[i];
      for (j=0;j<(int)strlen(palavra);j++)
            str[i++]=palavra[j];
      str[i++]=' ';
      for (;frase[indice]!='\0';i++,indice++)
            str[i]=frase[indice];
      str[i]='\0';
      strcpy(frase,TAM_FRASE,str);
}
```

b)

```
#include <stdio.h>
#include <string.h>

#define TAM_FRASE     150
#define TAM_PALAVRA   20

void InserePalavra(char *,char *,int);

void main()
{
      char  frase[TAM_FRASE],palavra[TAM_PALAVRA];
      int   i;

      strcpy(frase,"A Pizzaria Pepe serve pizzas muito
      quentes.");
      strcpy(palavra,"Verde");
      printf("\nFrase inicial: %s",frase);
      printf("\nPalavra a inserir: %s",palavra);
      printf("\nFrase alterada: ");
      InserePlvra(frase,palavra,16);
      for (i=0;frase[i]!= '\0';i++)
            printf("%c",frase[i]);
}
```

Exercício 6 – Ficha 7

Desenvolva uma função que receba um array de caracteres terminado por '\0' (string), a respectiva dimensão e um caracter (c). Esta função deve transformar a string original de tal forma que duplique o caracter c de cada vez que ele aparecer na string. A função deve devolver o valor inteiro 1 se a string for modificada de acordo com os requisitos especificados atrás e 0 caso contrário. Por exemplo:

> Se a string passada à função for **Como vai o amigo?** e o caracter for **o**,
> a string transformada de acordo com o especificado será **Coomoo vai oo amigoo?**

Especificação da função:

Dados de Entrada:

str (string) – Frase a alterar
len (int) – tamanho da frase
c (char) – Caracter a duplicar na frase

Resultados Pretendidos:

str (string) – Frase inicial alterada
altera (int) – 0 se a frase não foi alterada e 1 se foi

Processamento Requerido:

Receber (frase,len,c)
Ir copiando os caracteres da frase para uma string auxiliar e sempre que se encontrar o caracter c, repetir a sua introdução nessa string auxiliar, até que se chegue ao fim da string (atenção: não ultrapassar o tamanho da string inicial).
Devolver altera, com o valor respectivo

Codificação em C:

```
#include <stdio.h>

#define TAM     100

int InsereCaracter(char str[],int len,char c)
{
      int   i=0,j,pos=1,num=0;

      while (str[i]!='\0')
      {
            if (str[i]==c)
                  if ((int)strlen(str)<len-1)
                  {
                        i++;
                        for (j=(strlen(str));j>=i ;j--)
                              str[j+1]=str[j];
                        str[i]=c;
                        num++;
                  }
                  else
                        pos=0;
            i++;
      }
      if (num==0)
            pos=0;
      return pos;
}
```

```
void main()
{
      char  frase[TAM],car;
      int   i;

      printf("Introduza a frase: ");
      gets(frase);
      printf("\nQual o caracter a inserir: ");
      scanf("%c",&car);
      if (InsereCaracter(frase,TAM,car))
            printf("\nA string foi devidamente
            alterada.Ficou ");
      else
            printf("\nA string NAO foi devidamente
            alterada ou nao havia o caracter na
            frase.\nFicou ");
      for (i=0;frase[i]!='\0';i++)
            printf("%c",frase[i]);
}
```

Exercício 7 – Ficha 7

Escreva uma função que receba uma frase, uma palavra e as dimensões dos respectivos vectores onde estão armazenadas as *strings*, como argumentos. A função deverá substituir a última palavra da frase pela palavra passada como 2º argumento, devolvendo 1 se a frase for, de facto, modificada e 0 caso contrário. Só é possível fazer a alteração se o número de caracteres da palavra for menor ou igual à última palavra da frase. Assuma que as palavras na frase estão separadas por um ou mais espaços, podendo existir espaços no início e no fim da frase. O protótipo da função será: **int func(char frase[], char palavra[], int tamf, int tamp);**

Considerando a seguinte função main:

```
#include <stdio.h>
#define TAMFRASE80
#define TAMPAL       15
void main()
{
      char f[TAMFRASE],p[TAMPAL];
      printf("Indique frase: ");
      gets(f);
      printf("Indique palavra: ");
      gets(p);
      if (func(f,p,TAMFRASE,TAMPAL))
            printf("Frase modificada:\n\t%s\n",f);
      else
            printf("Frase nao modificada!\n");
}
```

O resultado de execução deverá ser o seguinte:

Indique frase: **Hoje e um dia importante**
Indique palavra: **normal**
Frase modificada: **Hoje e um dia normal**

Especificação da função:

Dados de Entrada:

f (string) – Frase a alterar

p (string) – Palavra a usar na alteração
tf (int) – Número de caracteres que a frase tem
tp (int) – Número de caracteres que a palavra tem

Resultados Pretendidos:

f (string) – Frase inicial alterada
altera (int) – 0 se a frase não foi alterada e 1 se foi

Processamento Requerido:

Receber (f,p,ltf,tp)
fim ← índice onde a última palavra da frase termina
inicio ← índice onde a última palavra da frase começa
Se o número de caracteres de p for menor ou igual ao número de caracteres da última palavra da frase, fazer a alteração e devolver 1. Senão, devolver 0

Codificação em C:

```
int func(char f[],char p[],int tf,int tp)
{
      int  i,inicio,fim;

      for (i=(int)(strlen(f))-1;f[i]==' ' && i>=0;i--);
      fim=i;
      for (;f[i]!=' ' && i>=0;i--);
      inicio=i+1;
      if (fim-inicio>=(int)(strlen(p))-1)
      {
            for (i=inicio;i<=fim;i++)
                  f[i]=p[i-inicio];
            f[i]='\0';
            return 1;
      }
      else
            return 0;
}
```

Exercício 8 – Ficha 7

Elabore uma função que:

• Receba como argumentos uma string ***str***, um valor inteiro correspondente ao tamanho do vector onde a string é armazenada ***tam***, um caracter ***c*** e um valor inteiro ***x***.

• Coloque o caracter recebido no final da string o número de vezes possível até um máximo de ***x*** vezes.

• Devolva o número de caracteres adicionados à string

• A string alterada, deve continuar a ser uma string válida.

O protótipo da função será: **int func(char str[], int tam, char c, int x);**

Exemplo de execução:

str: ABCDE ***c***: 'Z' ***x***: 5 ***tam***: 10
após a chamada à função, ***str***: ABCDEZZZZ, devolve 4
str: AA ***c***: 'Z' ***x***: 5 ***tam***: 10
após a chamada à função, ***str***: AAZZZZZ, devolve 5
str: ABCDEFGHI ***c***: 'M' ***x***: 5 ***tam***: 10
após a chamada à função, ***str***: ABCDEFGHI, devolve 0

Especificação da função:

Dados de Entrada:

f (string) – String a alterar
tam (int) – Número de caracteres que a string tem
c (char) – Caracter a acrescentar à string
x (int) – Número máximo de caracteres que se podem introduzir na string

Resultados Pretendidos:

f (string) – String inicial alterada
conta (int) – Número de caracteres introduzidos na string

Processamento Requerido:

Receber (f,tam,c,x)
i ← número de caracteres que a string tem
conta ← 0
Enquanto (i < tam-1 && conta ≠ x) deve
    Introduzir caracter c na posição i da string
    conta ← conta + 1
    i ← i + 1
Introduzir caracter \0 na posição i da string
Devolver conta

Codificação em C:

```
#include <stdio.h>

#define TAM     10

int func(char str[],int tam,char c,int x)
{
      int   i,conta=0;

      for (i=(int)(strlen(str));i<tam-1 && conta!=x;i++)
      {
            str[i]=c;
            conta++;
      }
      str[i]='\0';
      return conta;
}

void main()
{
      char  f[TAM],c;
      int   x=5;

      strcpy(f,"ABCDE");
      c='Z';
      printf("\nstr: %s c: %c x: %d tam: %d\n",f,c,x,TAM);
      printf("str: %s Devolveu: %d\n",f,func(f,TAM,c,x));
      strcpy(f,"AA");
      printf("\nstr: %s c: %c x: %d tam: %d\n",f,c,x,TAM);
      printf("str: %s Devolveu: %d\n",f,func(f,TAM,c,x));
      strcpy(f,"ABCDEFGHI");
      c='M';
      printf("\nstr: %s c: %c x: %d tam: %d\n",f,c,x,TAM);
      printf("str: %s Devolveu: %d\n",f,func(f,TAM,c,x));
}
```

Exercício 9 – Ficha 7

a) Desenvolva uma função que receba como argumentos uma string, um número inteiro x e um carácter c, e verifique se nessa string surge o carácter c, recebido como argumento, x vezes consecutivas. Se surgir a função deve devolver 1, caso contrário deve devolver 0.

b) Desenvolva um programa que, peça ao utilizador um carácter. Depois disso deve pedir ao utilizador um conjunto de strings. A introdução de strings deve terminar quando ele introduzir a string "**fim**". No final do programa deve ser mostrado ao utilizador qual foi a maior das strings introduzidas onde o carácter dado pelo utilizador surgiu 3 vezes consecutivas. Utilize a função anterior para saber se na string introduzida aparece o carácter dado três vezes consecutivas.

Exemplo de execução do programa:

Introduza um caracter: **B**
Introduza uma string: **ABA**
Introduza uma string: **ABBBBBAAA**
Introduza uma string: **BBBXB**
Introduza uma string: **CCC**
Introduza uma string: **FIM**
A maior string onde o carácter B surgiu 3 vezes consecutivas é: ABBBBBAAA

a) ***Especificação da função:***

Dados de Entrada:

str (string) – String a verificar se tem letras repetidas consecutivas
x (int) – Número de vezes que uma letra deverá estar repetida
c (char) – Caracter a ver se está repetido

Resultados Pretendidos:

(int) – 0 se o caracter c não estiver x vezes repetido em str ou 1, se estiver

Processamento Requerido:

Receber (str,x,c)
conta ← número de caracteres consecutivos que existem na string
Se conta ≥ x
    Devolver 1
Senão
    Devolver 0

Codificação em C:

```
int func(char str[],int x,char c)
{
      int   i,conta=0,maior=0;
      for (i=0;i<(int)strlen(str)+1;i++)
      {
            if (str[i]==c)
                  conta++;
            else
            {
                  if (maior<conta)
                        maior=conta;
                  conta=0;
            }
      }
      if (maior>=x)
            return 1;
      else
            return 0;
```

```
}
```

b) ***Codificação em C:***

```
#include <stdio.h>

#define TAM     100

int func(char,int,char);

void main()
{
      char  str[TAM],c,aux[TAM],maior=0,x;

      x=3;
      printf("Introduza um caracter: ");
      scanf("%c",&c);
      printf("\n");
      aux[0]='\0';
      do
      {
            printf("Introduza ums string: ");
            gets(str);
            if (func(str,x,c)==1)
            {
                  if (maior<(int)strlen(str))
                  {
                        maior=(int)strlen(str);
                        strcpy(aux, str);
                  }
            }
      }
      while (strcmp(str,"FIM"));
      printf("\n\nMaior string onde o caracter %c surgiu %d
      vezes consecutivas: %s\n\n",c,x,aux);
}
```

– Vectores ou *arrays* multidimensionais –

- ***Definição:***

  TipoDeDados NOME[Dimensão1]…[DimensãoN];

  É assim que se criam vectores multidimensionais que poderão guardar um número de valores do tipo TipoDeDados igual ao valor da multiplicação entre as suas dimensões.

  Para aceder a esses valores usa-se o seu respectivo índice. O índice dos *arrays* na linguagem C vai de zero a Dimensão-1.

- ***Atribuição:***
  - No momento da definição:

    TipoDeDado NM[2][N] = {{val00, val01, …, val0N}, {val10, val11, …, val1N}};
  - A um dos elementos do array:

    NOME[indice1]…[indiceN]=valor;

- ***Utilização:***
  - Sempre que se quiser aceder a um dos elementos do vector, usar

    NOME[indice]…[indiceN]

    Exemplo:

    int a[2][2] → O tamanho tem que ser constante

    a[0][0]=1;

    a[0][1]=9;

    a[1][0]=14;

    a[1][1]=5;

1	9
14	5

- ***Arrays multidimensionais como argumentos de função:***

```
#include <stdio.h>
void mostra(int v[][3],int lin,int col)
{
      int   i,j;
      for (i=0;i<lin;i++)
      {
            for (j=0;j<col;j++)
                  printf("%d ",v[i][j]);
            printf("\n");
      }
}
void main()
{
      int a[][3]={{12,9,14},{5,1,0}};
      mostra(a,2,3);
}
```

Exercício 13 – Ficha 6

Desenvolva um programa que inicialize um array 10×10 com valores aleatórios entre 0 e 10. Após a inicialização, o programa deve contar o número de zeros existentes no array.

Especificação da programa:

Constantes:

TAM (10) – Valor a usar para o número de linhas e colunas da tabela

Dados de Entrada

Não tem

Resultados Pretendidos:

$\begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix}$ (inteiro) – Tabela com valores aleatórios, entre 1 e 10

cont (inteiro) – Número de elementos com valor igual a 0

Processamento Requerido:

cont ← 0
Para cada uma das linhas da tabela:
    Para cada uma das colunas da tabela:
        a[linha][coluna] ← valor aleatório, entre 0 e 10
        Se (a[linha][coluna]=0) então
            incrementar o contador
Mostrar (a,cont)

Codificação em C:

```
#include <stdio.h>
#include <time.h>

#define    TAM 10

void main()
{
      int         a[TAM][TAM],i,j,cont=0;

      srand((unsigned)time(NULL));
      for (i=0;i<TAM;i++)
      {
            for (j=0;j<TAM;j++)
            {
                  a[i][j]=rand()%TAM;
                  printf("%d ",a[i][j]);
                  if (a[i][j]==0)
                        cont++;
            }
            printf("\n");
      }
      printf("\nNumero de zeros no array: %d\n\n",cont);
}
```

Exercício 14 – Ficha 6

Desenvolva um programa que inicialize um array 10×3 da seguinte forma: em cada uma das linhas, a primeira coluna deve ficar com um inteiro entre 1 e 100 introduzido pelo utilizador, a segunda coluna com o quadrado deste valor e a terceira com o cubo. Após a inicialização, o programa deve contar quantas posições do array têm valores superiores a 1000.

Especificação da programa:

Constantes:
TAM (10) – Valor a ser usado para o número de linhas da tabela
LM (1000) – Valor a usar na comparação com os elementos da tabela
Dados de Entrada
v1, …, v10 – Valores da primeira coluna da tabela, entre 1 e 100
Resultados Pretendidos:
tab[TAM][3] (inteiro) – Tabela com os números
c (inteiro) – Número de elementos com valor superior a LM
Processamento Requerido:
c ← 0
Para cada uma das linhas da tabela:
Obtém (v entre 1 e 100)
Elemento da primeira coluna dessa linha ← v
Elemento da segunda coluna dessa linha ← v^2
Elemento da terceira coluna dessa linha ← v^3
Para cada uma das linhas da tabela:
Para cada uma das colunas da tabela:
Se (tab[linha][coluna]>LM) então
c ← c + 1
Mostrar (tab,c)

Pseudocódigo:

```
INÍCIO Programa
    contador ← 0
    i ← 0
    ENQUANTO (i<TAM) FAZER
        FAZER
            OBTER (tab[i][0])
        ENQUANTO (tab[i][0]<1 OU tab[i][0]>100)
        tab[i][1] ← (tab[i][0])^2
        tab[i][2] ← (tab[i][0])^3
        i ← i + 1
    FIM ENQUANTO
    i ← 0
    j ← 0
    ENQUANTO (i<TAM) FAZER
        ENQUANTO (j<3) FAZER
            SE (tab[i][j]>LM) ENTÃO
                c ← c + 1
            FIM SE
            j ← j + 1
        FIM ENQUANTO
```

```
            i ← i + 1
        FIM ENQUANTO
        MOSTRA (tab,contador)
    FIM Programa
```

Codificação em C:

```c
#include <stdio.h>
#include <math.h>

#define TAM     10
#define LM      1000

void main()
{
      int tab[TAM][3],i,j,c=0;

      printf("Introduza o valor para posicao:\n");
      for (i=0;i<TAM;i++)
      {
            do
            {
                  printf("(1,%d) [1-100] - ",i+1);
                  scanf("%d",&tab[i][0]);
            }while (tab[i][0]<1 || tab[i][0]>100);
            tab[i][1]=tab[i][0]*tab[i][0];
            tab[i][2]=(int)pow((float)tab[i][0],3);
      }
      printf("\n");
      for (i=0;i<TAM;i++)
      {
            for (j=0;j<3;j++)
            {
                  if (tab[i][j]>LM)
                        c++;
                  printf("%d\t",tab[i][j]);
            }
            printf("\n");
      }
      printf("Numero de valores da tabela > %d: %d",LM,c);
}
```

Exercício 18 – Ficha 6

Considere a seguinte tabela:

	A	B	C	D
A	0	-1	12	1
B	-1	0	5	6
C	12	5	0	-1
D	1	6	-1	0

A informação da tabela representa as distâncias (utilizando uma ligação directa) entre 4 cidades (A, B, C, D). Quando aparece a informação -1, as cidades não estão ligadas directamente.

a) Declare a estrutura de dados local à função main que permita armazenar a informação da tabela.

b) Desenvolva uma função que receba a tabela e o nome de uma das cidades e devolva a distância a que se encontra a cidade mais próxima (ligação directa).

Especificação da função:

Dados de Entrada

tdist (int [4][4]) – Tabela das distâncias
cid (char) – Nome da cidade

Resultados Pretendidos:

mDist – Distancia a que se encontra a cidade mais próxima

Processamento Requerido:

mDist ← Menor número que o computador conhece
Calcular o índice da cidade na tabela (A – 0, B – 1, C – 2, D – 3)
Enquanto não chega ao fim da linha cuja coluna é igual a índice:
    Se o valor de uma posição for positivo e menor que menorDist
        Actualizar menorDist
Devolver menorDist

Pseudocódigo:

INÍCIO FUNÇÃO menorDistancia
RECEBE (tdist,cid)
    menorDist ← Menor número que o computador conhece
    ind ← cid - 'A'
    i ← 0
    ENQUANTO (i<NCID) FAZER
        SE (tdist[ind][i]<mDist E tdist[ind][i]>0) ENTÃO
            mDist ← tdist[ind][i]
        FIM SE
        i ← i + 1
    FIM ENQUANTO
    DEVOLVE (mDist)
FIM FUNÇÃO menorDistancia

Codificação em C:

```c
#include <stdio.h>
#include <ctype.h>

#define NCID    4

int menorDistancia(int tdist[][NCID],char cid)
{
      int   i,mDist=INT_MAX,ind=cid-'A';

      for (i=0;i<NCID;i++)
            if (tdist[ind][i]<mDist && tdist[ind][i]>0)
                  mDist=tdist[ind][i];
      return mDist;
}

void main()
{
      int   dist[NCID][NCID]={{0,-1,12,1},{-
      1,0,5,6},{12,5,0,-1},{1,6,-1,0}},i,j;
      char  cd;

      for(i=0;i<NCID;i++)
      {
            for(j=0;j<NCID;j++)
                  printf("%4d\t",dist[i][j]);
            printf("\n");
```

```
        }
        do
        {
              fflush(stdin);
              printf("\nQual a cidade [ A | B | C | D ]? ");
              scanf("%c",&cd);
              cidade=toupper(cd);
              if (cd!='A' && cd!='B' && cd!='C' && cd!='D')
                    printf("\nCidade invalida\n");
        }
        while (cd!='A' && cd!='B' && cd!='C' && cd!='D');
        printf("\nA cidade mais proxima fica a um distancia
        de %d\n\n",menorDistancia(dist,cd));
}
```