Flávio Morgado

EXERCÍCIOS DE ALGORITMOS

Parte I

Santo André
2007

SUMÁRIO

1. PROBLEMAS ALGORÍTMICOS

Segundo Gardner (1990), os problemas podem ser classificados em seis categorias: combinatórios, geométricos, numéricos, lógicos, processuais e verbais, podendo haver sobreposições entre as categorias. Neste curso, os problemas são do tipo processual, ou seja, que possuem solução algorítmica, como em uma receita de bolo.

Segundo as Diretrizes Curriculares de Cursos da Área de Computação e Informática (MEC/SESu/CEEInf, 2006), "um algoritmo é um método abstrato mas bem definido para resolução de um problema em tempo finito".

O uso de algoritmos é anterior à própria palavra, originada do sobrenome de um matemático persa do século IX, Al-Khwarizmi, que teve sua obra sobre o sistema de numeração decimal (indiano) publicado no Ocidente no século XII (http://pt.wikipedia.org/wiki/Algoritmo). Como exemplo de algoritmos, temos:

- Dividir um número inteiro por outro
- Soma de frações
- Máximo Divisor Comum (MDC) de dois números inteiros (Euclides)
- Decomposição de números inteiros em fatores primos (Fatoração)

1.1. Baixa produtividade

Uma estória que ilustra o que são os algoritmos e como eles podem ser mais, ou menos, eficientes é a de um pintor que foi contratado para pintar uma faixa contínua em uma estrada. Passados alguns dias, o supervisor chama o pintor e diz que a produtividade dela tem diminuído muito, pois no 1º dia ela pintou 120 metros de faixa, no 2º pintou 60, no 3º, 45 e foi sempre diminuindo. O pintor respondeu:

– Eu não estou trabalhando menos. A lata de tinta é que está cada vez mais longe.

Moral da estória: "Não basta apenas funcionar. Os algoritmos têm de ser eficientes".

1.2. Adivinhação

Um mágico se propõe a adivinhar o número que uma pessoa pensou, no intervalo de 0 a 1024 e usa a seguinte estratégia, fazendo a seguinte seqüência de perguntas para a pessoa:

- O número que você pensou é o zero?
- O número que você pensou é o um?
- O número que você pensou é o dois?

e assim por diante. Quando a pessoa responder "sim", o mágico chegou na solução.

Outro mágico utiliza a seguinte estratégia: A pessoa deve responder se o número falado pelo mágico é o número pensado ou se é menor ou maior que o número pensado.

O primeiro número perguntado pelo mágico é o 512, que é a metade do intervalo de adivinhação, de 0 a 1024.

Se a pessoa responder que não é o 512 e que o número pensado é menor que 512, o mágico desconsidera os números maiores ou iguais a 512 e acha o meio do intervalo de 0 a 511, que é 256. Se a pessoa responder que não é o 512 e que o número pensado é maior que 512, o mágico desconsidera os números menores ou iguais a 512 e acha o meio do intervalo de 513 a 1024, que é 768.

O processo segue assim: elimina-se metade de cada intervalo e divide-se ao meio o intervalo em que se encontra o número pensado, até que o número pensado seja encontrado.

O segundo processo é mais eficiente que o primeiro, ou seja, gasta menos recursos computacionais (perguntas do mágico). Para determinar quanto um algoritmo é melhor que outro, compara-se o esforço computacional (complexidade) de cada algoritmo.

Uma forma de comparação, neste caso, é supor que cada algoritmo será usado para fazer 1024 adivinhações e compara-se a quantidade de perguntas (esforço computacional) das duas estratégias:

1ª estratégia: A quantidade de perguntas de cada adivinhação é, em média, 512 (alguns pensarão em números no início da faixa; outros, querendo ver o mágico se cansar, pensarão em números próximos de 1024).

2ª estratégia: A quantidade de perguntas é, no máximo 10 (cada vez que o mágico faz a pergunta, o intervalo é dividido ao meio: 1024 / 2 = 512; 512 / 2 = 256; 256 / 2 = 128; 64; 32; 16; 8; 4; 2; 1. Quando o tamanho do intervalo fica igual a um, o número é "adivinhado").

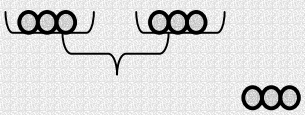
Exemplo: Se o número pensado for o 47, na primeira estratégia seriam feitas 47 perguntas. Na segunda, seriam feitas as seguintes perguntas:

Pergunta	Início do intervalo	Fim do intervalo	Meio (Número perguntado)
1	0	1024	512
2	0	512	256
3	0	256	128
4	0	128	64
5	0	64	32
6	32	64	48
7	32	48	40
8	40	48	44
9	44	48	46
10	46	48	47 (número pensado)

1.3. Moeda mais pesada

Num conjunto de 9 moedas, uma é mais pesada que as demais. Usando uma balança de 2 pratos [1], identificar a moeda mais pesada.

Uma estratégia é dividir as moedas em 3 conjuntos de 3 moedas, colocar um conjunto em cada prato e deixar um conjunto de fora:



1ª Pesagem
Se houver equilíbrio, a mais pesada estará entre as que estão fora. Senão, estará no prato mais baixo.
2ª Pesagem
Do conjunto que contém a mais pesada, colocar uma em cada prato. Se houver equilíbrio, a que ficou fora é a mais pesada, senão, a mais pesada será aquela do prato mais baixo.

Outra estratégia é dividir as moedas em 2 conjuntos de 4 moedas e colocar um conjunto em cada prato, ficando uma moeda de fora:

1ª Pesagem
Se houver equilíbrio, a mais pesada é a que ficou fora. Senão, está no prato mais baixo.
2ª Pesagem
Do conjunto que contém a mais pesada, colocar duas em cada prato. A mais pesada estará no prato que descer.
3ª Pesagem
Colocar uma moeda em cada prato. A mais pesada estará no prato que descer.

Ao examinarmos as duas estratégias notamos que na 1ª sempre haverá duas pesagens, enquanto que na 2ª, a moeda mais pesada poderá ser localizada já na 1ª pesagem.

Qual é a mais eficiente?

[1] Trata-se de uma balança analógica, que não mostra o peso, como numa balança digital, mas se uma coisa tem o mesmo peso que outra.

Podemos supor que temos de identificar 9 lotes de moedas. Serão necessárias 18 pesagens na 1ª estratégia (9 lotes X 2 pesagens), e 25 pesagens na 2ª (1/9 de probabilidade de encontrarmos na primeira pesagem, e as outras 8 vezes com 3 pesagens), ou seja:

1 / 9 de 9 + 8 X 3

Para esse número de moedas, a 1ª estratégia é melhor.

Outra comparação é quanto à reutilização das estratégias para conjuntos com mais moedas. Como exemplo, usaremos as duas estratégias para um conjunto de 40 moedas.

Nesse momento, os "vendedores" de cada estratégia criarão *slogans* para "vender seu peixe", algo como: "Divida em 3 partes", ou "Um é pouco, dois é bom, três é demais".

1ª Estratégia	**Dividir em 3 partes. Nesse caso, teríamos 13 moedas em cada prato e 14 moedas fora.**
1ª Pesagem	Se houver equilíbrio, a mais pesada estará nas moedas de fora. Senão, estará no prato que desceu. Teríamos, nesse caso, 6 ou 7 moedas para a próxima pesagem.
2ª Pesagem	Colocar 2 moedas em cada prato e 2 ou 3 fora. A conclusão seria a mesma da primeira pesagem.
3ª Pesagem	Com 2 ou 3 moedas, uma em cada prato, acharemos a mais pesada

2ª estratégia	**Dividir em 2 partes iguais. Se a quantidade de moedas for impar, 1 moeda ficará de fora.**
1ª Pesagem	Colocar 20 moedas em cada prato. A mais pesada estará no prato que descer
2ª Pesagem	Por 10 moedas em cada prato. A mais pesada estará no prato que descer
3ª Pesagem	Por 5 moedas em cada prato. Se houver equilíbrio, a mais pesada será a de fora. Senão, está no prato que desceu.
4ª Pesagem	Por 2 moedas em cada prato. Se houver equilíbrio, a mais pesada será a de fora. Senão, estará no prato que desceu.
5ª Pesagem	Colocar uma moeda em cada prato. A mais pesada estará no prato que desceu.

Tente mostrar que, exceto por sorte, a 1ª estratégia é melhor que a 2ª.

1.4. Caixa eletrônico

Um procedimento (algoritmo) para retirar dinheiro de um caixa eletrônico é algo simples, que está ao alcance de quase todos, devido às facilidades crescentes das interfaces com o usuário, apesar do não entendimento, ainda, da voz ou do pensamento do cliente.

Existem variações de procedimentos entre os bancos, tais como o tipo de acesso ao quiosque, retenção ou não do cartão durante a transação, posição do cartão para inserção etc.

1	**Entrar no quiosque**	Envolve a leitura do cartão na porta do quiosque e verificação se é um cartão válido. A leitura é feita num determinado sentido, pois existem 4 possibilidades com o cartão na horizontal, e só uma é válida). Nesse caso, um desenho do cartão (ícone) próximo ao mecanismo de leitura pode ajudar.
2	**Colocar cartão na máquina**	Outra vez ocorre o problema sobre como colocar o cartão.
3	**Digitar a senha**	A senha digitada deve ser verificada com a senha do cartão. Deve haver um limite de tentativas (normalmente 3). O programa deve emitir mensagens sobre o que ocorrerá após as tentativas frustradas.
4	**Solicitar valor**	Um menu (cardápio) deve ser mostrado para que o cliente escolha o tipo de serviço (saldo, extrato, retirada, transferência, pagamento etc.). Se a escolha for "retirada", o programa deve sugerir alguns valores comuns, ou abrir um campo para a digitação de outro valor que, normalmente, deve ser múltiplo de R$ 10,00. Devem, também, ser verificados os limites (saldo, acumulado de retiradas no dia, horários especiais etc.)
5	**Retirar o cartão**	Normalmente, o cartão é liberado antes do dinheiro, para evitar esquecimentos. Pode-se também verificar se o cliente quer utilizar algum outro serviço antes de devolver o cartão.
6	**Retirar o dinheiro**	Deve-se ter um mecanismo de contagem e escolha de notas para cada pagamento.
7	**Sair do quiosque**	Mesmo este ato pode não ser tão simples, existindo seguros instantâneos para casos de roubos em caixas eletrônicos.

No refinamento deste procedimento não foram consideradas as implicações tecnológicas, de *performance,* e dos inúmeros contratempos que podem ocorrer, como máquina quebrada, papel enroscado, cartão "engolido", sem comunicação (*off-line*) etc.

1.5. Problemas propostos

a) Ligação telefônica

Escreva um procedimento detalhado para efetuar uma ligação telefônica de um aparelho público.

b) Moeda diferente

Num grupo de oito moedas, uma delas é diferente (mais leve ou mais pesada). Usando uma balança de dois pratos, escreva um procedimento para descobrir qual.

c) Moedas falsas

Existem 10 montes de moedas, cada um com 10 moedas de 10 gramas. Num dos montes todas as moedas são falsas (cada uma delas pesa um grama a mais que as verdadeiras). Descreva os passos para descobrir o monte de moedas falsas, usando uma balança digital.

d) Travesia 1 (Canibais)

Três missionários estão de um lado de um rio com três canibais e desejam atravessar um rio usando um barco com capacidade para, no máximo, duas pessoas. O problema é que se o número de canibais for maior que o número de missionários, em qualquer uma das margens, os canibais comem os missionários (Barbosa, 1999).

e) Travessia 2 (Maridos ciumentos)

Três maridos, com suas respectivas esposas, querem atravessar um rio. Acontece que, no barco só cabem duas pessoas e os 3 maridos são muito ciumentos e não permitem que sua esposa fique numa das margens com outro homem, sem que ele também esteja presente. Como pode ser feita a travessia? (Barbosa, 1999).

f) Problema do vinho

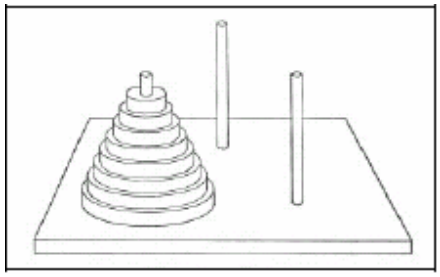
Existem três jarras com capacidade para exatamente 8, 5 e 3 litros. A primeira jarra está cheia de vinho. Como fazer para repartir esse conteúdo, em partes iguais, nas duas jarras maiores. Não existem graduações nas jarras, nem é permitido faze-las (Barbosa, 1999).

g) TIC-TAC-TOE (Jogo da velha)

No jogo da velha, quem começa pelo centro ganha com certeza, desde que o adversário responda com uma casa lateral. Dê a receita para ganhar (Barbosa, 1999).

h) Torre de Hanoi

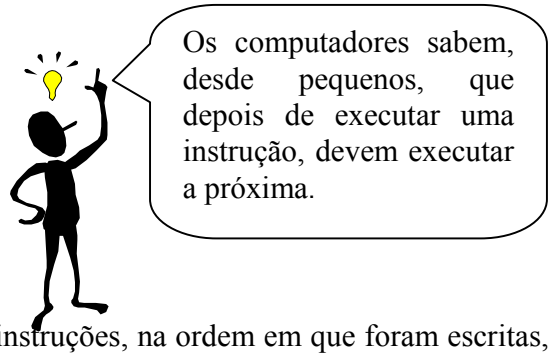
(ENADE Computação – 2005) No famoso jogo da Torre de Hanói, é dada uma torre com discos de raios diferentes, empilhados por tamanho decrescente em um dos três pinos dados, como ilustra a figura abaixo. O objetivo do jogo é transportar-se toda a torre para um dos outros pinos, de acordo com as seguintes regras: apenas um disco pode ser deslocado por vez, e, em todo instante, todos os discos precisam estar em um dos três pinos; além disso, em nenhum momento, um disco pode ser colocado sobre um disco de raio menor que o dele; é claro que o terceiro pino pode ser usado como local temporário para os discos.



Imaginando que se tenha uma situação em que a torre inicial tenha um conjunto de 5 discos, qual o número mínimo de movimentações de discos que deverão ser realizadas para se atingir o objetivo do jogo?

A – 25
B – 28
C – 31
D – 34
E – 38

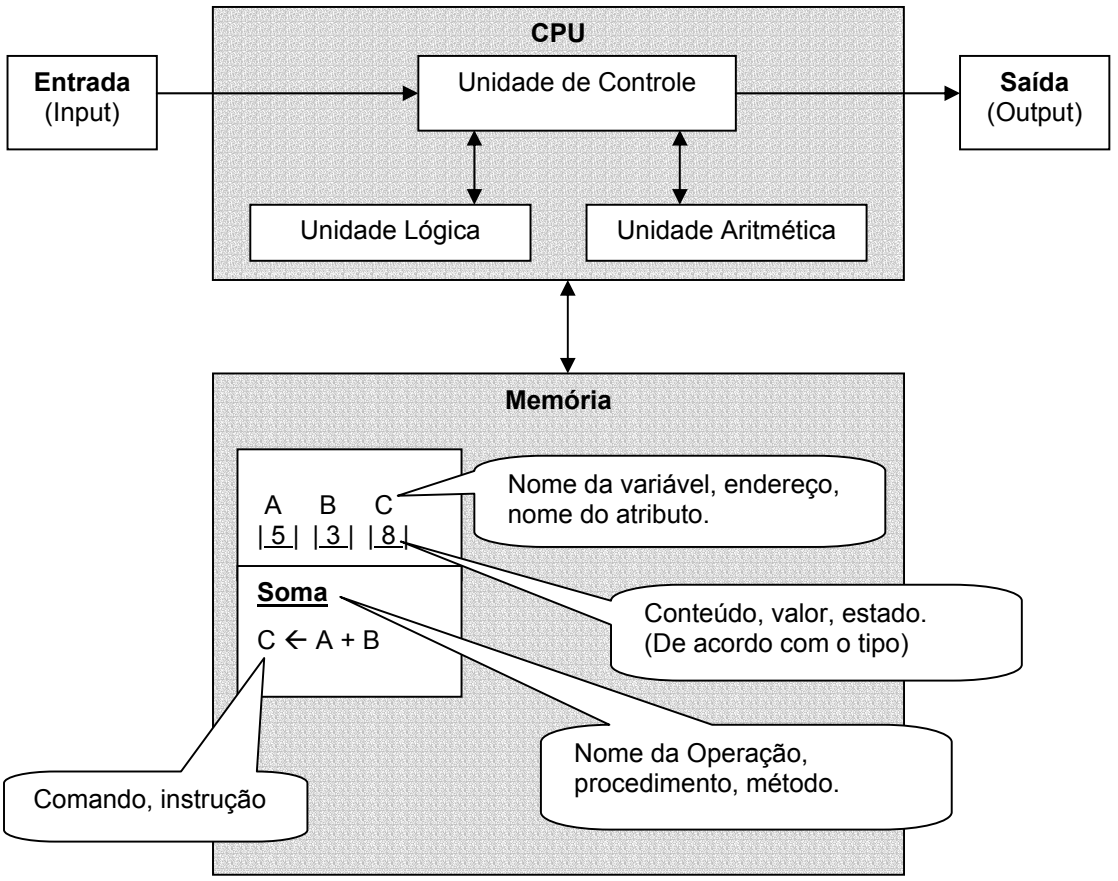
2. CONCEITOS FUNDAMENTAIS



Um computador executa seqüências de instruções, na ordem em que foram escritas, exceto se houver alguma instrução de desvio.

2.1. Organização dos Computadores

Uma máquina que queira ser chamada de computador deve conter os componentes básicos mostrados no diagrama a seguir:



2.2. Variáveis

Segundo Sebesta (2000, p. 161), "uma variável pode ser caracterizada como um sêxtuplo de atributos: nome, endereço, valor, tipo, tempo de vida e escopo".

"O tipo de uma variável determina a faixa de valores que ela pode ter e o conjunto de operações definidas para os valores do tipo" (SEBESTA, 2000, p.163).

2.2.1. Tipo Inteiro

É utilizado para armazenar valores entre –32.768 e 32.767 ($2^{15} - 1$), sem parte decimal, usando 2 bytes, com as seguintes operações possíveis:

Operação	**Símbolo**	**Exemplo**
Adição	+	7 + 2 = 9
Subtração	-	7 – 2 = 5
Multiplicação	*	7 * 2 = 14
Divisão inteira	\ → quociente **mod** → resto	7 \ 2 = 3 7 mod 2 = 1

O tipo inteiro é utilizado para representar valores enumeráveis: quantidades de peças, contadores, número de passagens por uma rotina, dias de atraso etc.

2.2.2. Tipo Real

Números de ponto flutuante entre $-3.402823 * 10^{38}$ até $3.402823 * 10^{38}$, na precisão simples, ocupando 4 bytes (padrão IEEE).

Como exemplo, o nº 5.483 seria armazenado de forma normalizada: $0.5483 * 10^{3}$, na qual a parte decimal é chamada de mantissa (precisão) e o expoente da base 10 é chamado de característica (grandeza).

As operações são as mesmas que as do tipo inteiro, exceto pela divisão, na qual existe apenas um operador, cujo símbolo é uma /, e que fornece valores com casas decimais, sem resto.

O tipo Real é usado para representar números em geral: salários, notas, preços, saldos etc.

2.2.3. Tipo Caractere

Usado para armazenar as letras do alfabeto, numerais e sinais de pontuação. O código ASCII (*American Standard Code for Information Interchange*), representa, em 7 bits, o padrão de caracteres para a língua inglesa, além dos caracteres de controle. Na versão estendida, o código ASCII inclui caracteres próprios das diversas línguas e símbolos gráficos.

Tabela ASCII

Decimal	ASCII	Decimal	ASCII	Decimal	ASCII	Decimal	ASCII
0	NUL	32	Espaço	64	@	96	`
1	SOH	33	!	65	A	97	a
2	STX	34	“	66	B	98	b
3	ETX	35	#	67	C	99	c
4	EOT	36	$	68	D	100	d
5	ENQ	37	%	69	E	101	e
6	ACK	38	&	70	F	102	f
7	BEL	39	‘	71	G	103	g
8	BS	40	(	72	H	104	h
9	HT	41	)	73	I	105	i
10	LF	42	*	74	J	106	j
11	VT	43	+	75	K	107	k
12	FF	44	,	76	L	108	l
13	CR	45	-	77	M	109	m
14	SO	46	.	78	N	110	n
15	SI	47	/	79	O	111	o
16	DLE	48	0	80	P	112	p
17	DC1	49	1	81	Q	113	q
18	DC2	50	2	82	R	114	r
19	DC3	51	3	83	S	115	s
20	DC4	52	4	84	T	116	t
21	NAK	53	5	85	U	117	u
22	SYN	54	6	86	V	118	v
23	ETB	55	7	87	W	119	w
24	CAN	56	8	88	X	120	x
25	EM	57	9	89	Y	121	y
26	SUB	58	:	90	Z	122	z
27	ESC	59	;	91	[	123	{
28	FS	60	<	92	\	124	|
29	GS	61	=	93	]	125	}
30	RS	62	>	94	^	126	~
31	US	63	?	95	_	127	DEL

Em sua versão estendida, com 8 bits, o código ASCII representa também caracteres especiais, como os acentuados, em algumas línguas.

Decimal	ASCII	Decimal	ASCII	Decimal	ASCII	Decimal	ASCII
128	Ç	132	ä	136	ê	163	ú
130	é	133	à	160	á	166	ª
131	â	135	ç	162	ó	167	º

2.2.4. Tipo String

Cadeias de 0 a 65.535 caracteres (*string* = cordão, em inglês). O tipo *string* serve para armazenar nomes de pessoas, endereços e códigos em geral, mesmo aqueles que contenham apenas dígitos, pois, normalmente, não são feitas operações aritméticas com os códigos.

Ex.: "INFORMÁTICA" ; "ERRO: Cliente já existe"; "12345-6"

2.2.5. Tipo Data

Datas e horas, ocupando 8 bytes, usadas para representar datas de nascimento, vencimento, casamento, pagamento, etc. Permite-se operações de diferença entre datas (em dias), acrescentar dias a uma data, e existem funções para obter só o dia, só o mês ou só o ano.

Ex.: 05/04/98 – 26/03/98 → 9 27/02/97 + 4 → 03/03/97

2.2.6. Tipo Booleano

Representa valores Verdadeiro ou Falso. O resultado das operações de negação (não), conjunção (e) e disjunção (ou) é dado pelas tabelas verdade.

2.3. Regras para identificadores

Os nomes de variáveis seguem algumas regras:

- Nomes devem ser mnemônicos, ou seja, devem ajudar a lembrar o conteúdo e os objetivos propostos. Exemplo: Salario, MediaFinal, Seno.
- Variáveis de uma letra devem ser usadas apenas em um escopo local.
  Exemplo: K (contador), X (auxiliar), I (índice).
- Nomes muito parecidos podem confundir. Exemplo Cli e Cii.
- As letras "I", "L" e "O" podem ser confundidas com os nºs 0 e 1.
- Utilizar prefixos ou sufixos para identificar categorias.
  Ex.: **frm**Cliente (formulário), **db**Cliente (base de dados).

2.4. Expressões e Hierarquia de Operadores

Na avaliação de expressões feita pelo computador, valem as mesmas regras utilizadas na 6ª série, para "destruir" aquelas expressões que ocupavam uma página inteira do caderno:

- A avaliação de expressões segue uma hierarquia (precedência) de operadores;
- Os parênteses são usados para forçar a avaliação;
- A resolução é feita a partir do nível de parênteses mais interno para o externo;
- Quando existirem operadores de mesma precedência, a ordem de avaliação será da esquerda para a direita;

- Quando existirem várias categorias de operadores, a avaliação obedecerá esta ordem: aritméticos, relacionais e, por fim, os lógicos.

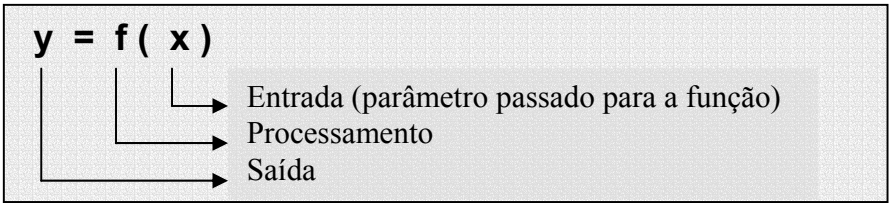
Aritméticos		**Relacionais**		**Lógicos**	
Exponenciação	^	Igual	=	Não	**NOT**
Negação ou Inversão de Sinal	–	Diferente	<>	E	**AND**
Multiplicação e Divisão	* /	Maior	>	OU	**OR**
Divisão Inteira	\	Menor	<	OU Exclusivo	**XOR**
Resto	**Mod**	Maior ou igual	>=		
Adição e Subtração	+ -	Menor ou igual	<=		
Concatenação de *strings*	**&** +				

Exemplos de expressões matemáticas e sua codificação em uma linha:

Expressão matemática	**Codificação**
$b^2 - 4ac$	b ^ 2 – 4 * a * c
$\frac{1 + N}{2}$	(1 + N) / 2
$X = \frac{\sqrt{b \pm \Delta}}{2a}$.	X1 = (– b + Delta ^ 0.5) / (2 * a) X2 = (– b – Delta ^ (1/2)) / (2 * a)

2.5. Funções pré-definidas

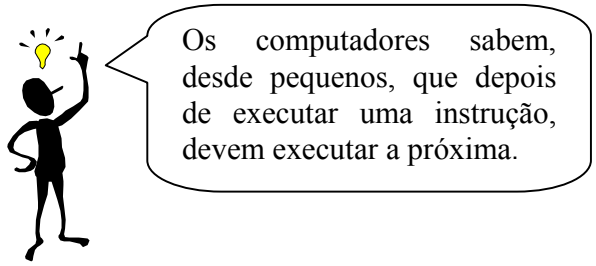
O conceito de função, em computação, é o mesmo que em matemática:



Função	**Descrição**	**Exemplo (retorno da função)**
int (Expr.Numérica)	Parte inteira do número	int (2.4) = 2
Date ( )	Data de hoje	09/02/2000
Time ( )	Hora corrente	13:50:08
Abs (Expr.Numérica)	Valor absoluto	abs (-5) = 5
rnd	Número aleatório entre 0 e 1	0.564326
Sin (Expr.Numérica)	Seno de um arco, dado em radianos	Sin (3.141592654/6) = 0.5

É necessário respeitar o tipo de parâmetro da função. No caso da função seno, se escrevermos sin (30), a resposta será –0.988, que é o seno de 30 radianos, e não de 30 graus.

3. CONSTRUTOR LÓGICO SEQUENCIAL



Um computador executa seqüências de instruções, na ordem em que foram escritas, exceto se houver alguma instrução de desvio.

As instruções básicas, que não executam desvios são:

Variável ← Expressão	A instrução de atribuição avalia a expressão escrita do lado direito do símbolo ← (calcula o valor da expressão) e atribui (armazena) o resultado na variável escrita do lado esquerdo.
Entrada (lista de variáveis)	Esta instrução faz a leitura de valores de um dispositivo de entrada (teclado, disco, mouse,...) para as variáveis de memória.
Saída (lista de variáveis ou constantes)	Esta instrução copia valores das variáveis de memória, ou outros conteúdos constantes, para um dispositivo de saída (tela, disco, impressora,...).

As instruções de entrada e saída serão usadas em alguns poucos casos na descrição dos algoritmos deste curso, pois o foco é a solução dos problemas, não a interface, ou seja, os dados de entrada estão disponíveis (já foram lidos) para a execução dos processos e as variáveis de saída (resultados) ficam disponíveis para serem mostradas, impressas, gravadas, etc.

3.1. Termômetro

Um termômetro mede a temperatura em graus Fahrenheit. Criar um processo para converter a temperatura para graus Celsius.

Uma solução

As temperaturas, em graus Celsius (º C) e em graus Fahrenheit (º F), são dados do tipo real. A fórmula para conversão de Fahrenheit para Celsius é: $º C = \frac{5}{9} (º F - 32)$

Exemplo: Dada a temperatura de 68 º F, o procedimento calcula º C = 20.

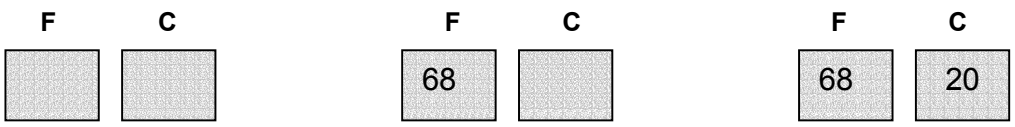
A descrição do algoritmo para resolver esse problema é:

TERMÔMETRO	
Real C	// *Graus Celsius*
Real F	// *Graus Fahrenheit*
Converte ( ) *Objetivo: Converter graus Fahrenheit para Celsius* C ← 5 / 9 * (F – 32)	

As variáveis do problema são definidas numa seção especial, incluindo os respectivos tipos (C e F são do tipo Real).

"Converte ( )" é um algoritmo que faz a conversão de Fahrenheit (variável F) para Celsius (variável C).

As variáveis passam pelas seguintes mudanças de estado (valor) na memória:



3.2. Área do Círculo

Dado o raio de um círculo, obter sua área.

Uma solução

O raio é um dado do tipo real, que deve ser > 0. A área do círculo é do tipo real, e é obtida pela fórmula: $\mathbf{Área = \Pi * Raio^2}$

CIRCULO	
Real Raio	// *Raio do círculo*
Real Area	// *Área*
Calc_Area ( ) *Objetivo: Obter a área do círculo, dado seu raio* Pré-condição: Raio > 0 Area ← 3.1416 * Raio ^ 2	

3.3. Cronômetro

Um cronômetro mede, em segundos, a quantidade de tempo decorrida entre dois instantes. Converter o tempo em segundos para horas, minutos e segundos, para facilitar a leitura.

Exemplos do estado final das variáveis Horas, Minutos e Segundos:

Tempo		Horas	Minutos	Segundos
100	⇨	0	1	40

Tempo		Horas	Minutos	Segundos
3800	⇨	1	3	20

Uma solução

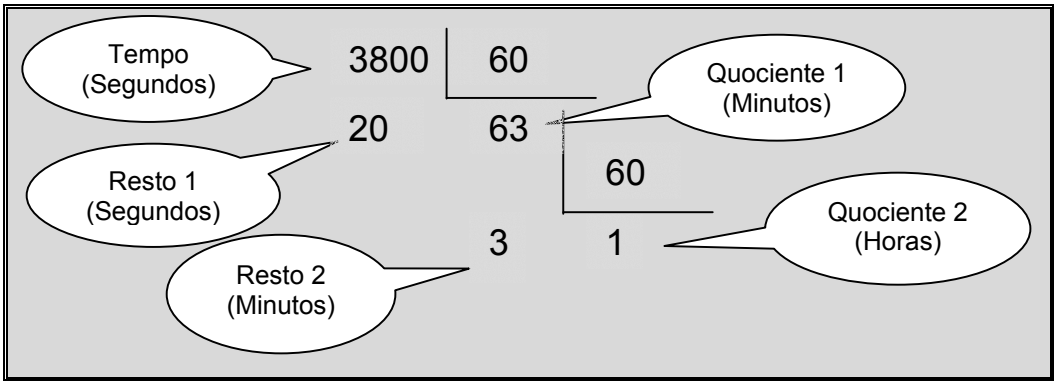
A quantidade de segundos é um valor do tipo inteiro. A resposta do problema é a quantidade de Horas, Minutos e Segundos, também valores do tipo inteiro.

Para fazer conversões de tempo, consideram-se as seguintes relações:

1 minuto = 60 segundos; 1 hora = 60 minutos.

Dado um valor em segundos, pode-se calcular a quantidade de minutos existentes, dividindo a quantidade de segundos por 60. O resto dessa divisão é a quantidade de segundos mesmo (não forma um minuto).

A quantidade de minutos obtida pode ser maior que 60. Para calcular a quantidade de horas existentes, obtém-se o quociente da divisão dos minutos por 60.

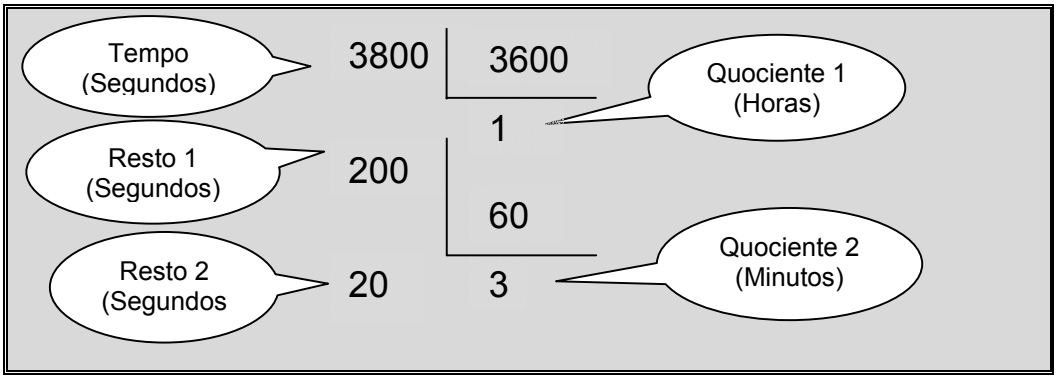


A descrição do algoritmo para resolver esse problema é:

CRONÔMETRO	
Inteiro Tempo	// *Tempo em segundos*
Inteiro H, M, S	// *Horas, Minutos e Segundos, após a conversão.*
Conv_HMS() *Objetivo: Converter tempo em segundos para Horas, Minutos e Segundos* Pré-condição: Tempo > 0 S ← Tempo mod 60 M ← Tempo \ 60 H ← M \ 60 M ← M mod 60	

Outra solução

A partir da relação 1 hora = 3600 segundos, temos:



CRONÔMETRO	
Inteiro Tempo	// *Tempo em segundos*
Inteiro H, M, S	// *Horas, Minutos e Segundos, após a conversão*
Conv_HMS_2 ( ) Objetivo: *Converter tempo, em segundos, para Horas, Minutos e Segundos* Pré-condição: Tempo > 0 H ← Tempo \ 3600 S ← Tempo mod 3600 M ← S \ 60 S ← S mod 60	

3.4. Rateio

Dividir uma quantidade de alunos em 3 classes, de modo que a diferença entre elas seja mínima (Oliveira, 2005).

Uma solução

Trata-se de um problema típico de Administração Escolar: dividir uma quantidade de alunos em um determinado número de classes, para que todas as classes fiquem com o mesmo número de estudantes. Nem sempre a divisão por 3 resulta exata, e evidentemente a quantidade de alunos por sala deve ser um número inteiro.

Ex: 46 / 3 = 15,33..., ou seja, Classe 1 = 15, Classe 2 = 15 e Classe 3 = 16

A idéia inicial é obter o quociente da divisão por 3 e atribuir esse valor à Classe 1 (quantidade da menor sala, já que a parte fracionária é perdida). A Classe 2 pode ter a mesma quantidade da Classe 1. A Classe 3 é obtida pela diferença entre a quantidade de alunos inicial e os alunos já colocados nas duas primeiras classes.

ALUNOS
Inteiro Alunos, Classe1, Classe2, Classe3
Balanceamento ( ) Objetivo: Dividir uma quantidade de alunos em 3 salas, de forma balanceada Classe1 ← Alunos \ 3 Classe2 ← Classe1 Classe3 ← Alunos – Classe1 – Classe2

Vamos verificar se o algoritmo está correto para 63, 64 e 65 alunos:

Classe	Cálculo	Quantidade de alunos		
		63	64	65
1	Quantidade \ 3	21=63 \ 3	21=64 \ 3	21=65 \ 3
2	= Classe 1	21	21	21
3	Alunos–Classe1–Classe2	21=63–21–21	22=64–21–21	23=65–21–21

Para a quantidade = 65 a distribuição seria 21, 21 e 23 para as classes 1, 2 e 3. Neste caso, o resultado 21, 22, 22 seria melhor, pois as classes estariam com uma divisão mais balanceada. Isso indica que o algoritmo, embora tenha feito uma divisão correta, não obteve o melhor resultado. Isso ocorre, pois na divisão por 3 podem sobrar 1 ou 2 alunos.

Uma melhoria no balanceamento seria a divisão por 2 dos alunos não alocados na primeira sala e o cálculo da terceira sala por diferença.

ALUNOS
Inteiro Alunos, Classe1, Classe2, Classe3
Balanceamento ( ) Objetivo: Dividir uma quantidade de alunos em 3 salas Classe1 ← Alunos \ 3 Classe2 ← (Alunos – Classe1) \ 2 Classe3 ← Alunos – Classe1 – Classe2

Neste caso, o algoritmo fará a distribuição corretamente:

Classe	Cálculo	Quantidade de alunos		
		63	64	65
1	Quantidade \ 3	21=63 \ 3	21=64 \ 3	21=65 \ 3
2	(Alunos–Classe1) \ 2	21	21	22
3	Alunos–Classe1–Classe2	21=63–21–21	22=64–21–21	22=65–21–21

3.5. Erro comum (para iniciantes)

Um erro comum para iniciantes é achar que uma expressão, uma vez escrita, torna-se uma verdade para o restante do programa:

Ex.: B ← A / 2
A ← A + 1

Se o valor de A for alterado, o valor de B não será alterado, como ocorre nas planilhas eletrônicas.

3.6. Problemas Propostos

a) Área do triângulo

Dado um triângulo, definido pelos valores dos lados, calcular sua área.

$$\text{Área} = \sqrt{s\,(s-a).(s-b).(s-c)} \qquad \text{onde} \quad s = \frac{a+b+c}{2}$$

Exemplo: Dado um triângulo com lados a = 3, b = 4 e c = 5, a área seria = 6

b) Distância entre dois pontos num plano

Dadas as coordenadas de 2 pontos num plano (X1, Y1) , (X2, Y2), calcular a distância entre esses pontos.

$$d = \sqrt{(X_2 - X_1)^2 + (Y_2 - Y_1)^2}$$

Exemplo: Dados os pontos (3,2) e (6,2), a distância seria = 3

c) Juros Compostos

Dados o valor presente (VP), um nº de períodos (n) e uma taxa de juros do período (i, em decimais, ou seja, 5% = 0,05), calcular seu valor futuro (VF), a juros compostos.

$VF = VP\ (1 + i)^n$

Exemplo: Dado VP = R$ 1.000, 00, capitalizado em 12 meses, a uma taxa de 2% de juros ao mês, o valor futuro será = R$ 1.268,24 = (1 + 0.02) ^ 12.

d) Separar dígitos

Dado um número inteiro, positivo, < 1000, obter a quantidade de centenas, dezenas e unidades desse número.

Exemplo: Dado o nº 764, obter Centena = 7, Dezena = 6 e Unidade = 4

e) Caixa eletrônico

Um caixa eletrônico trabalha com cédulas de 100, 50, 20, 10, 5, 2 e 1. Obter a quantidade de cédulas de cada tipo, para efetuar um pagamento de uma quantia escolhida pelo cliente, usando a menor quantidade de cédulas.

Exemplos:

Valor	Quantidade de notas	Valor das notas
283	2	100
	1	50
	1	20
	1	10
	0	5
	1	2
	1	1

Valor	Quantidade de notas	Valor das notas
64	0	100
	1	50
	0	20
	1	10
	0	5
	2	2
	0	1

f) Dígitos centrais

Dado um número inteiro entre 1000 e 9999, mostrar a soma dos dígitos centrais (2º e 3º dígitos).

g) Decimal para binário

Dado um número inteiro, positivo, < 32, mostrar sua representação em binário.

Um número na base binária (Base 2), utiliza os princípios do sistema de numeração posicional e aditivo, o mesmo que usamos para a base 10, ou seja:

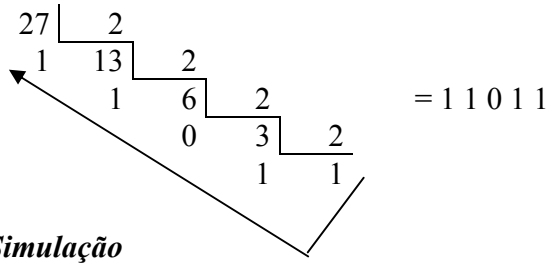
$$342 = 3 \text{ X } 100 + 4 \text{ X } 10 + 2 = 3 \text{ X } 10^{2} + 4 \text{ X } 10^{1} + 2 \text{ X } 10^{0}$$

Para representar as quantidades, escrevemos apenas os coeficientes das potências da base, ou seja, 3, 4 e 2. A posição do dígito no número determina o seu valor. Assim, 3 dígito 3 vale mais que o 4, pois está na posição (casa) da centena (10^{2}) e, portanto, vale 300, enquanto o 4 vale 40, pois está na posição da dezena (10^{1}).

Quando usamos a base 2, cada posição tem o peso de uma potência da base 2, e os dígitos, em vez de 0 a 9 da base 10, serão apenas 0 e 1. Assim, para representarmos a quantidade 27 na base 2, escrevemos:

$$\mathbf{1} \text{ X } 2^{4} + \mathbf{1} \text{ X } 2^{3} + \mathbf{0} \text{ X } 2^{2} + \mathbf{1} \text{ X } 2^{1} + \mathbf{1} \text{ X } 2^{0} = 11011$$

Um método para obter os coeficientes das potências de 2 é dividir sucessivamente por 2, enquanto o quociente for ≥ 2, e utilizar como resposta o último quociente e os restos das divisões por 2, na ordem inversa que foram calculados:



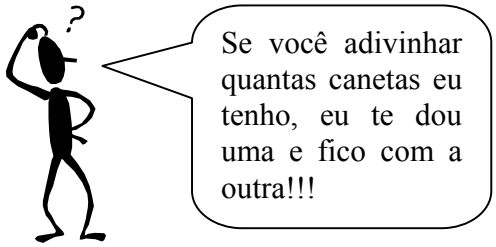
= 1 1 0 1 1

h) Simulação

Simular e determinar qual o objetivo do seguinte algoritmo.

Variáveis N, N1, N2, N3, N4, N5 (Inteiro)
Procedimento ( ) *Objetivo:* N1 ← N \ 16 N2 ← (N mod 16) \ 8 N3 ← N mod 8) \ 4 N4 ← N mod 4) \ 2 N5 ← N mod 2

4. CONSTRUTOR LÓGICO DE SELEÇÃO (DECISÃO)



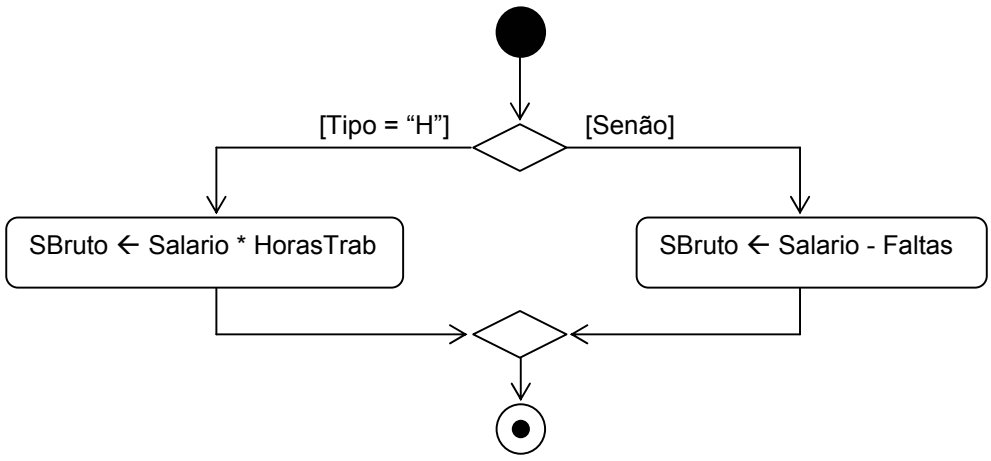
O construtor de seleção é usado para testar uma condição e, se esta for verdadeira, executar um bloco de instruções, ou, se for falsa, executar, opcionalmente, outro bloco.

A condição é uma expressão envolvendo constantes, variáveis e operadores aritméticos, relacionais e lógicos e que retorna um valor Verdadeiro (*True*) ou Falso (*False*).

4.1. Salário Bruto

Dados o tipo do funcionário ("H" = Horista ou "M" = Mensalista), o salário por hora (horistas) ou por mês (mensalistas) e, o número de horas trabalhadas (horistas) ou o valor de faltas (mensalistas), calcular o valor do salário bruto.

O fluxograma da solução é:



A seguir, a mesma estrutura escrita em linguagem algorítmica:

```
Se     Tipo = "H" então
       SBruto ← Salario * Horas Trab
Senão
       SBruto ← Salario – Faltas
Fim-se
```

4.2. Verificar se é par

Dado um número inteiro, positivo, verificar se ele é par.

Uma solução

Se o número dado for par, atribuir à variável Par, do tipo booleano, o valor "Verdadeiro", senão, atribuir o valor "Falso".

Um número par é divisível por 2, ou seja, o resto da divisão por 2 é igual a zero.

NÚMERO	
Inteiro N	// *Número inteiro dado*
Booleano Par	// *Informação se o número é par*
Verifica_Par( )	
Objetivo: Verificar se um número inteiro positivo é par	
Se N mod 2 = 0 então	
Par ← Verdadeiro	
Senão	
Par ← Falso	
Fim-Se	

4.3. Maior de 2 números

Dados 2 números distintos, obter o maior.

Uma solução

A pré-condição é que os números são distintos.

Compara-se os nºs e o maior é armazenado em uma variável que será a cópia do maior entre os dois números.

2 NÚMEROS	
Inteiro A, B	// *2 números dados*
Inteiro Maior	// *Cópia do maior dos dois números*
Maior_de_2 ( )	
Objetivo: Obter o maior de 2 números	
Pré-condição: A e B são distintos	
Se A > B então	
Maior ← A	
Senão	
Maior ← B	
Fim-Se	

4.4. Maior de 3 números

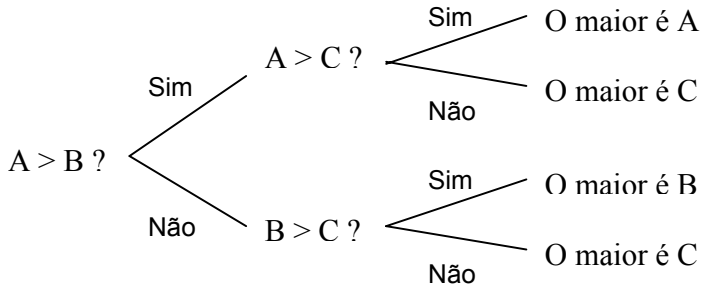
Dados 3 n°s distintos, mostrar o maior.

Uma solução

Sejam A, B e C os 3 n°s. As comparações entre eles são feitas dois a dois. Primeiramente, acha-se o maior entre A e B. Se A for maior que B, A será comparado com C. Se A for maior que C, então será o maior dos 3. Senão, C será o maior, ou seja:

O vencedor da briga entre A e B disputa com C, e quem ganhar a 2ª briga, é o maior dos 3.

A árvore de possibilidades que mostra essa estratégia é:



As instruções estão numeradas para que seja feita, posteriormente, uma simulação do algoritmo.

```
   3 NÚMEROS
   Inteiro A, B, C                    // 3 números dados
   Inteiro Maior                      // Cópia do maior dos 3 números
   Maior_de_3 ( )
   Objetivo:  Achar o maior de 3 números
   Pré-condição: A, B e C são distintos
1  Se  A > B então
2      Se   A > C então
3          Maior ← A
       Senão
4          Maior ← C
       Fim-Se
   Senão
5      Se   B > C então
6          Maior ← B
       Senão
7          Maior ← C
       Fim-Se
   Fim-Se
```

Simulação

Na simulação, o algoritmo é "executado" para determinados valores (instâncias), sendo verificado se produz a resposta esperada.

Exemplo: supondo-se que os valores de A, B e C sejam 6, 8, 4, o algoritmo executaria as seguintes instruções, produzindo o seguinte resultado:

Instrução	Variáveis				Teste	Resultado do teste
	A	B	C	Maior		
	6	8	4			
1					A > B (6 > 8) ?	F
5					B > C (8 > 4) ?	V
6				8		

A variável Maior ficou com uma cópia do maior dos 3 números.

Usa-se, nesse casso, o encadeamento de perguntas, ou seja, uma pergunta dentro da outra.

Se for usado o operador ≥ nas comparações e houver repetição do maior, o primeiro número da comparação será escolhido como maior.

Segunda estratégia

Pode-se tornar as "brigas" independentes, ou seja, a variável Maior fica com o "vencedor" entre A e B e disputa com C. Se o valor da variável Maior for menor que o de C, o conteúdo da variável Maior será substituído pelo conteúdo de C.

```
3 NÚMEROS
Inteiro A, B, C, Maior              // 3 números dados e cópia do maior
Maior_de_3 ( )
Objetivo:  Achar o maior de 3 números
Pré-condição: A, B e C são distintos
Se  A ≥ B  então
    Maior ← A
Senão
    Maior ← B
Fim-Se
Se  Maior < C  então
    Maior ←  C
Fim-Se                              // Neste caso, não é necessário o SENÃO.
```

Essa estratégia é melhor, pois, no caso do acréscimo de um número, muda-se muito a estrutura da primeira solução, enquanto que nesta é só acrescentar uma comparação da variável Maior com uma quarta variável (D).

Terceira estratégia

Outra estratégia é descobrir se A é o maior dos três. Se não for, o maior estará entre B e C. Para descobrir se A é o maior, pode-se utilizar uma pergunta composta, ou seja, para ser o maior, A deve ser maior que B e maior que C.

As proposições podem ser combinadas por meio de conectivos lógicos. O resultado dessa combinação pode ser visualizado numa **tabela-verdade**.

A tabela-verdade para 2 proposições (**p** e **q**), com dois valores possíveis para cada uma (V ou F), tem 4 possibilidades. A tabela da negação tem 2 possibilidades.

p	**q**	**Conjunção (E)** **p ^ q**	**Disjunção (OU)** **p v q**
F	F	F	F
V	F	F	V
F	V	F	V
V	V	V	V

p	**Negação (NÃO)** **~ p**
V	F
F	V

3 NÚMEROS
Inteiro A, B, C, Maior // *3 números dados e cópia do maior*

Maior_de_3 ( )
Objetivo: Achar o maior de 3 números
Pré-condição: A, B e C são distintos
Se A ≥ B e A ≥ C então
  Maior ← A
Senão
  Se B ≥ C então
  Maior ← B
  Senão
  Maior ← C
  Fim-Se
Fim-Se

Um erro comum, ao verificar se o maior é A, é a seguinte pergunta:

Se A ≥ B e B ≥ C então Maior ← A

A variável A pode até ser a maior e o comando Maior ← A não ser executado (no caso de B < C), ou seja, para que A seja o maior não é necessário que B ≥ C.

Ex.: A = 7; B = 4; C = 5.

Outro erro muito comum ocorre quando tornamos os SEs independentes ou em seqüência, ou seja, não encadeados. O algoritmo a seguir funciona somente se o maior valor for B ou C.

```
3 NÚMEROS
Inteiro A, B, C, Maior                // 3 números dados e cópia do maior
Maior_de_3_Com_Erro ( )
Objetivo:  Achar o maior de 3 números
Pré-condição: A, B e C são distintos
Se  A ≥ B e A ≥ C então
    Maior ← A
Fim-Se
Se  B ≥ C e B ≥ A então
    Maior ← B
Senão
    Maior ← C
Fim-Se
```

Convém lembrar que, de acordo com a Lei de Murphy, o programador tende a testar justamente os casos em que o programa não apresenta erro, e que o erro só será descoberto quando puder causar sérios prejuízos à reputação do programador.

Uma dica interessante, quando temos vários casos para decisão, é verificar quais casos ocorrem menos, e então perguntarmos por eles. No exercício acima, para impedirmos a entrada de números iguais, temos de identificar os casos em que pode ocorrer igualdade, ou seja:

A = B e A = C
A = B e B ≠ C
A = C e C ≠ B
B = C e C ≠ A

Como temos somente um caso válido (A ≠ B ≠ C), é melhor perguntar por ele, deixando os casos de erro para o "senão". A função a seguir efetua essa verificação:

```
Função Distintos (Real A, B, C) Booleano
Objetivo: Verificar se 3 números são distintos
Distintos ← A ≠ B  And  B ≠ C  And  A ≠ C
```

A definição sobre a possibilidade de entrarem números iguais, ou não, depende da aplicação. Se for para escolher a melhor nota de prova entre três, não há problema se entrarem três notas iguais. Se os três valores são preços de compra, devemos utilizar outro critério de desempate (condições de pagamento, prazo de entrega, qualidade etc.)

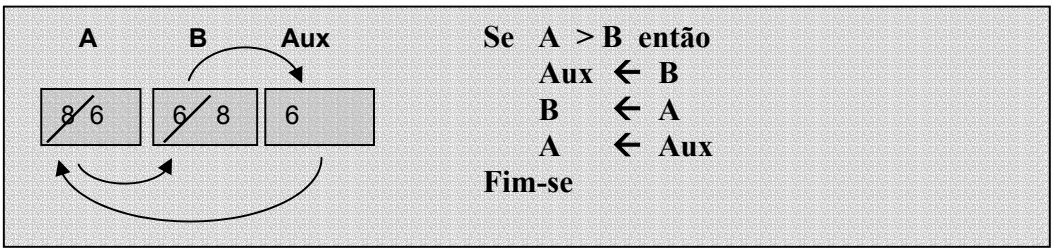
4.5. Ordem crescente

Dados 3 números distintos, colocá-los em ordem crescente.

Uma solução

Uma estratégia é comparar os nºs , dois a dois, e, se estiverem fora da ordem desejada, trocar o conteúdo deles. Não basta, porém, comparar A com B, e B com C, pois, se B e C forem permutados, a ordem entre A e B também pode ter sido alterada.

Para permutarmos o valor de duas variáveis, é necessário utilizar uma terceira variável, que funciona como auxiliar da troca (*backup)*:

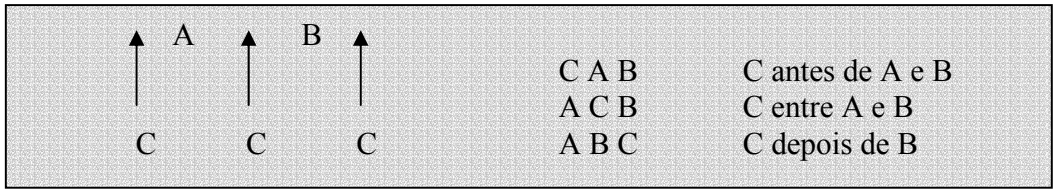


3 NÚMEROS
Real A, B, C
Ordenar3 ( )
Objetivo: Colocar 3 números em ordem crescente
Se A > B então
Trocar (A, B)
Fim-Se
Se B > C então
Trocar (B, C)
Fim-Se
Se A > B então
Trocar (A, B)
Fim-Se

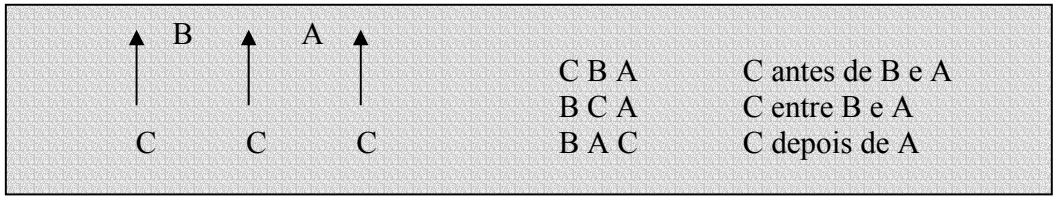
Procedimento Trocar (X, Y) *// X e Y são passados por referência.*
Objetivo: Permutar o conteúdo de X com Y
Real Aux
Aux ← X
X ← Y
Y ← Aux

Segunda estratégia

Supondo que não precisássemos trocar o conteúdo das variáveis, tendo apenas que mostrar os números em ordem. Se A < B, resta saber a posição de C, que pode estar antes de A, entre A e B, ou depois de B.



Se B < A, resta saber a posição de C, que pode estar antes de B, entre B e A, ou depois de A.




```
3 NÚMEROS
Real A, B, C
Ordenar3 ( )
Objetivo: Mostrar 3 números em ordem crescente
Se   A < B então                    // A antes de B
     Se   C < A  então              // C antes de A
          Imprimir (C, A, B)
     Senão
          Se  C < B então           // C entre A e B
              Imprimir (A, C, B)
          Senão
              Imprimir (A, B, C)    // C depois de B
          Fim-Se
     Fim-Se
Senão                               // B antes de A
     Se   C < B  então              // C antes de B
          Imprimir (C, B, A)
     Senão
          Se  C < A então           // C entre B e A
              Imprimir (B, C, A)
          Senão
              Imprimir (B, A, C)    // C depois de A
          Fim-Se
     Fim-Se
Fim-Se
```

4.6. Contar Pares

Dados 3 nºs inteiros e positivos, mostrar quantos são pares.

Uma solução

O objetivo é mostrar quantos são pares, e não quais. Uma das soluções é criar uma variável com a função de contador. Essa variável seria iniciada com zero e a cada vez que ocorrer um nº par, é somado 1 no contador.

No final, o contador poderá continuar com 0 (nenhum par), ou ter valor 1, 2 ou 3.

```
3 NÚMEROS
Inteiro A, B, C                          // 3 números
Inteiro QP                               // Quantidade de pares
Conta_Pares ( )
Objetivo: Dados 3 nºs inteiros, mostrar quantos são pares
QP ← 0
Se A mod 2 = 0 então    QP ← QP + 1
Se B mod 2 = 0 então    QP ← QP + 1
Se C mod 2 = 0 então    QP ← QP + 1
```

Estratégia 2 (Função característica)

Outra estratégia é utilizar a função característica (seqüência de bits), representando respectivamente casos de sim ou não. Como são 3 casos, usamos 3 variáveis, com valor inicial zero, e mudando-se para 1 se o valor correspondente for par. No final, soma-se as 3 variáveis e o resultado é a quantidade de pares.

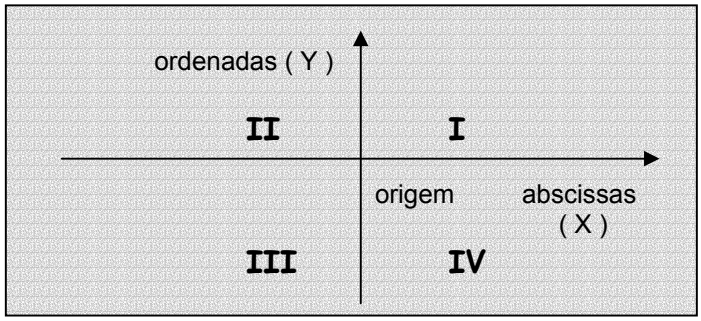
```
3 NÚMEROS
Inteiro A, B, C, QP                      // 3 números e Quantidade de pares
Conta_Pares ( )
Objetivo: Dados 3 nºs inteiros, mostrar quantos são pares
Inteiro Bit1, Bit2, Bit3
Bit1 ← 0
Bit2 ← 0
Bit3 ← 0
Se A mod 2 = 0 então    Bit1 ← 1
Se B mod 2 = 0 então    Bit2 ← 1
Se C mod 2 = 0 então    Bit3 ← 1
QP ← Bit1 + Bit2 + Bit3
```

4.7. Quadrante

Dadas as coordenadas de um ponto no plano cartesiano (X e Y), localizar onde está o ponto: quadrante, eixo ou origem.

Uma solução

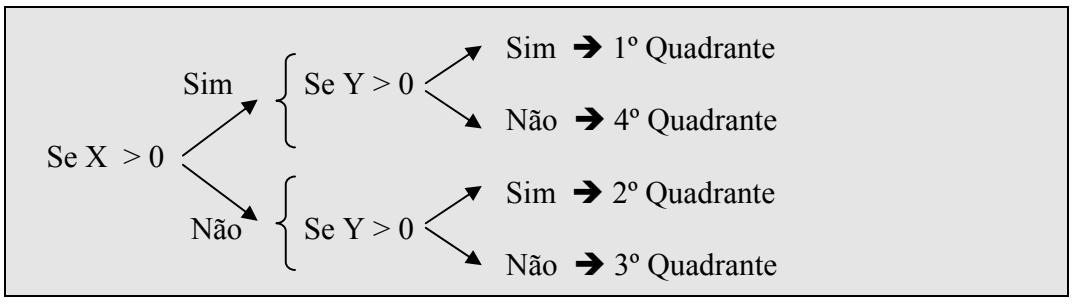
Parte-se do conhecimento sobre coordenadas cartesianas, como a ordem dos quadrantes (anti-horária, a partir do quadrante superior direito), os eixos (abscissas e ordenadas) etc.



Desconsiderando-se, inicialmente, os pontos localizados nos eixos ou na origem, pode-se fazer perguntas específicas para determinar o quadrante (3 perguntas e 1 senão)

Se X > 0 e Y > 0 ➔ 1º quadrante
Se X < 0 e Y > 0 ➔ 2º quadrante
Se X < 0 e Y < 0 ➔ 3º quadrante
Senão ➔ 4º quadrante

Porém, observa-se que se X > 0, o ponto só poderá estar no 1º ou 4º quadrante. Isso evita perguntas desnecessárias.



Para determinar se um ponto está em um eixo ou quadrante, perguntamos:
Se $X \neq 0$ e $Y \neq 0$

```
PONTO NO PLANO
Real X, Y                          // Abscissa e ordenada do ponto
String Loc                         // Localização do ponto
Localizar_Ponto ( )
Objetivo: Localizar ponto no plano cartesiano
Se   X ≠ 0  e  Y ≠ 0 então         // Quadrante
     Se   X > 0 então              // 1º ou 4º quadrantes
          Se    Y > 0 então
                Loc ← "1º Quadrante"
          Senão
                Loc ← "4º Quadrante"
          Fim-se
     Senão                         // 2º ou 3º quadrantes
          Se    Y > 0 então
                Loc ← "2º Quadrante"
          Senão
                Loc ← "3º Quadrante"
          Fim-se
      Fim-se
Senão                              // Eixo ou origem
      Loc ← "Eixo ou origem"
Fim-Se
```

Pode-se inverter as ações do "então" e do "senão", negando a condição:

Não ($X \neq 0$ e $Y \neq 0$)

Usando De Morgan, temos: $X = 0$ ou $Y = 0$

Estratégia 2

Pode-se, também, usar outra estratégia para determinação do quadrante, perguntando por X * Y. Se X * Y for > 0, X e Y têm o mesmo sinal, que ocorre para pontos do 1º ou 3º quadrantes. Se X * Y for < 0, então os sinais são invertidos, que ocorre para pontos do 2º ou 4º quadrantes. Se o ponto estiver no 1^o ou 3^o quadrante (sinais iguais), se X for positivo, o ponto estará no 1^o quadrante, senão, estará no 3^o.

Nessa solução será refinada, também, a localização do ponto quando este não estiver em um dos quadrantes, ou seja, estive em um dos eixos ou na origem.

```
PONTO NO PLANO
Real X, Y                          // Abscissa e ordenada do ponto
String Loc                         // Localização do ponto
Localizar_Ponto ( )
Objetivo: Localizar ponto no plano cartesiano
Se   X = 0  ou  Y = 0   então      // Eixo
     Se   X = 0 então
          Se   Y = 0 então
               Loc ← "Origem"
          Senão
               Loc ← "Ordenada"
          Fim-se
     Senão
          Loc ← "Abscissa"
     Fim-se
Senão                              // Quadrante
    Se   X * Y > 0 então
         Se    X > 0 então
               Loc ← "1º Quadrante"
         Senão
               Loc ← "3º Quadrante"
         Fim-se
    Senão
         Se    X > 0 então
               Loc ← "4º Quadrante"
         Senão
               Loc ← "2º Quadrante"
         Fim-se
     Fim-se
Fim-se
```

4.8. Alunos

Obter a média final (MF) e a situação do aluno, dadas as notas P1 e P2 (Provas), A1 e A2 (Atividades), de acordo com os seguintes cálculos e critérios:

$0 \leq$ Qualquer nota ou média ≤ 10, em frações de 0.5 ponto

$$M1 = \frac{P1 \times 3 + A1 \times 2}{5} \qquad M2 = \frac{P2 \times 3 + A2 \times 2}{5}$$

M1 e M2 são calculadas com arredondamento para 1ª casa decimal, pelo critério universal, ou seja, a partir da metade, inclusive, arredonda-se para cima. Senão, arredonda-se para baixo.

MF = $\frac{M1 + M2}{2}$ arredondada para o mais próximo inteiro ou 0.5.

Exemplo: 6.2 → 6.0 6.25 → 6.5 6.7 → 6.5 6.75 → 7.0

A situação do aluno é obtida em função da média final (MF):

Média Final (MF)	**Situação**
Abaixo de 3.0	Reprovado
De 3.0 até abaixo de 7.0	Exame
De 7.0 em diante	Aprovado

Uma solução

O algoritmo terá as seguintes macro-instruções, que serão depois refinadas:

- Verificar se as notas são válidas
- Calcular e arredondar a média final
- Obter a situação do aluno

Para verificar se as notas são válidas, testamos cada uma delas, no momento (evento) da entrada dos dados.

Se Nota ≥ 0 **e** Nota ≤ 10.0 **e** (Nota * 10) mod 5 = 0 então **é válida**

Exemplo: A nota 6,7 será inválida, pois (6,7 X 10) mod 5 = 2, que é ≠ 0

De acordo com De Morgan: ~ (P ^ Q ^ R) ⇔ ~P v ~Q v ~R .

Assim, podemos perguntar se a nota é válida, ou usar a forma complementar (nota não válida)

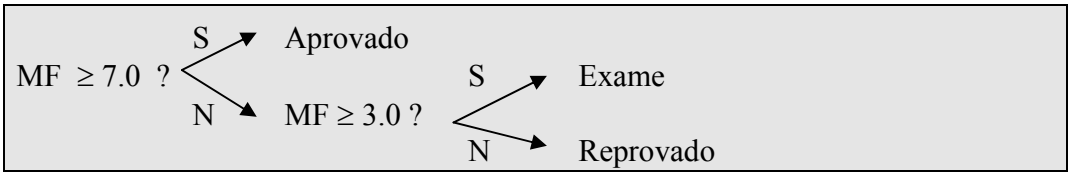
Se Nota < 0 **ou** Nota > 10.0 **ou** (Nota * 10) mod 5 ≠ 0 então **não é valida**

A função NotaOK verifica se a nota é valida:

```
Função NotaOK (Real Nota) Booleano
Objetivo: Verificar se uma determinada nota é válida
Se   Nota ≥ 0   e   P1 ≤ 10.0   e   (P1 * 10) mod 5 = 0   então
     NotaOK ← Verdadeiro
Senão
     NotaOK ← Falso
Fim-se
```

Para arredondar MF examina-se a parte fracionária de MF, que pode ser calculada como: MF – Int (MF). Se MF < 0.25, arredonda-se para baixo; se MF ≥ 0.25 e MF < 0.75, arredonda-se para 0.5; senão, arredonda-se para cima.

Dada a Média Final (MF), temos:



Quando se lida com faixas, é melhor começar a perguntar pelos extremos (primeira ou última faixa). Se a primeira pergunta for da faixa central ela seria mais complexa:

Se MF ≥ 3 e MF < 7 então
    Situação ← "Exame" // Desse jeito, as perguntas não são compostas

```
ALUNO
Real P1, P2, A1, A2                          // Notas de prova e atividade
Real MF                                      // Média final
String Situação                              // Situação do aluno
CalcularSituação ( )
Objetivo: Determinar a média fina e a situação do aluno, dadas suas notas
Real Frac                                    // Parte fracionária da nota
M1   ← (P1 * 3 + A1 * 2) / 5                 // Calcular  médias
M2   ← (P2 * 3 + A2 * 2) / 5
M1   ← Int ( (M1 + 0,05 ) * 10 ) / 10        // Arredondar médias
M2   ← Int ( (M2 + 0,05 ) * 10 ) / 10
MF   ← (M1 +  M2 ) / 2                       // Calcular média final
Frac ← MF – int ( MF )                       // Arredondar média final
Se  Frac < 0.25 então
    MF ← int (MF)
Senão
    Se  Frac < 0.75 então
        MF ← int(MF) + 0.5
    Senão
        MF ← int (MF) + 1
    Fim-se
Fim-se
Se  MF ≥ 7 então                             // Obter situação do aluno
    Situação ← "Aprovado"
Senão
    Se  MF ≥ 3.0 então
        Situação ← "Exame"
    Senão
        Situação ← "Reprovado"
    Fim-se
Fim-se
```

4.9. Melhores notas

Dadas 4 notas de provas de um aluno, obter a média das 2 maiores notas. Se houver notas repetidas, considerar a primeira da seqüência.

Uma solução

Uma estratégia é iniciar as variáveis Maior1 e Maior2 com os valores das duas primeiras notas, a maior delas em Maior1 e a outra em Maior2.

Então, coloca-se a 3ª nota para "brigar" com os valores de Maior1 e Maior2. Se a 3ª nota for > Maior1, desloca-se o conteúdo de Maior1 para Maior2 e substituí-se Maior1 pela 3ª nota. Senão, ainda tem-se de comparar a 3ª nota com Maior2. Se a 3ª nota for > Maior2, substituí-se Maior2 por ela. Senão, deixa-se como está, ou seja, a 3ª nota perdeu as 2 "brigas". Repete-se o mesmo processo para a 4ª nota.

```
ALUNO
Real N1, N2, N3, N4, Média          // Notas de prova e média das duas maiores
MédiaMaiores ( )
Objetivo: Dadas 4 notas, calcular a média das 2 melhores notas
Real Maior1, Maior2
Se  N1 > N2 então                   // Colocar N1 e N2 em Maior1 e Maior2,
    Maior1 ← N1                     // em ordem decrescente.
    Maior2 ← N2
Senão
    Maior1 ← N2
    Maior2 ← N1
Fim-se
Se  N3 > Maior1 então               // Comparação da 3ª nota ( N3)
    Maior2 ← Maior1                 // com Maior1 e Maior2
    Maior1 ← N3
Senão
    Se  N3 > Maior2 então
        Maior2 ← N3
    Fim-se
Fim-se
Se  N4 > Maior1  então              // Comparação da 4ª nota ( N4 )
    Maior2 ← Maior1                 // com Maior1 e Maior2
    Maior1 ← N4
Senão
    Se  N4 > Maior2 então
        Maior2 ← N4
    Fim-se
Fim-se
Média ← (Maior1 + Maior2) / 2       // Média das maiores notas
```

Nessa estratégia, o código aumenta 5 instruções para cada nota acrescentada.

Estratégia 2

Outra estratégia é achar o maior das 6 somas de pares possíveis de notas:

N1 + N2
N1 + N3
N1 + N4
N2 + N3
N2 + N4
N3 + N4.

Pode-se usar uma variável para guardar a maior soma. Esta variável é iniciada com o valor da soma do primeiro par (N1 + N2)

```
ALUNO
Real N1, N2, N3, N4                 // Notas de prova
Real Média                          // Média
MédiaMaiores ( )
Objetivo: Dadas 4 notas, calcular a média das 2 melhores notas
Real Maior
Maior ← N1 + N2
Se   N1 + N3 > Maior   então  Maior ← N1 + N3
Se   N1 + N4 > Maior   então  Maior ← N1 + N4
Se   N2 + N3 > Maior   então  Maior ← N2 + N3
Se   N2 + N4 > Maior   então  Maior ← N2 + N4
Se   N3 + N4 > Maior   então  Maior ← N3 + N4
Média ← Maior / 2
```

Neste caso, quando se acrescenta uma nota, a quantidade de pares sobe para 10. Se forem mais duas notas, sobe para 15, mais 3, para 21, ...

Algumas perguntas podem induzir a erros, como por exemplo:

Se (N1 + N2) > (N3 + N4) então Maior ← (N1 + N2) / 2

No exemplo anterior, pode ocorrer o seguinte: N1 N2 N3 N4 ,
(9 + 3) > (7 + 4)

sendo obtido (N1 + N2) / 2, quando o correto seria (N1 + N3) / 2

4.10. Maior quantidade de produtos [2]

Uma loja do tipo preço único só possui produtos de 3 e 5 reais. O menor valor de venda é de 8 reais. Dado que sempre é possível compor qualquer quantia ≥ 8, apenas com produtos de 3 e 5 reais, mostrar como pode ser feita uma compra, de modo que seja adquirido a maior quantidade de produtos, e não sobre troco.

Uma solução

Como a quantidade de produtos adquiridos deve ser a maior possível, deve-se gastar a quantia com a maior quantidade de produtos de 3 reais. Quando não for possível, utiliza-se produtos de 5.

Se a quantia for dividida por 3, verifica-se que o resto pode ser 0 (múltiplos de 3), 1 ou 2. Se o resto for = 0, compra-se apenas produtos de 3 reais (Quantia / 3).

Se o resto for = 1, substituí-se 3 produtos de 3 reais, mais 1 real do resto, por 2 produtos de 5, ou seja, a quantia será gasta com 2 produtos de 5 e o restante (Quantia – 10), com produtos de 3.

Se o resto for = 2, substituí-se 1 produto de 3 reais, mais 2 reais do resto, por 1 produto de 5, ou seja, a quantia será gasta com 1 produtos de 5 e o restante (Quantia – 5), com produtos de 3.

A tabela a seguir, mostra as quantidades de produtos, em função do resto:

Resto	**Produtos de 3**	**Produtos de 5**
= 0	Quantia / 3	0
= 1	(Quantia – 10) / 3	2
= 2	(Quantia – 5) / 3	1

Usa-se neste algoritmo a estrutura de seleção múltipla:

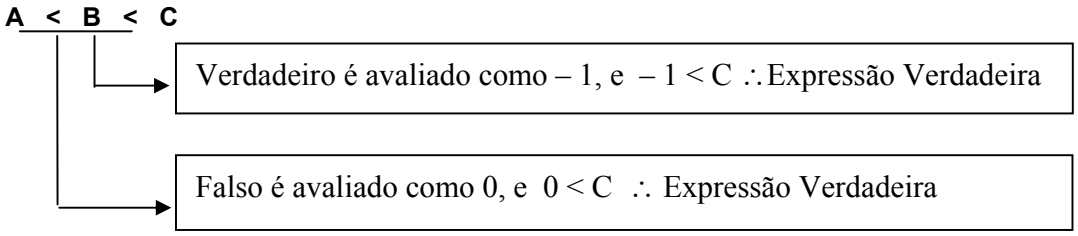
```
LOJA DE 3 E 5
Inteiro Quantia                 // Quantia disponível para compra
Inteiro P3, P5                  // Quantidade de produtos de 3 e de 5
Calcular3_5 ( )
Objetivo : Calcular maior quantidade de produtos
Caso Quantia mod 3              // Resto de Quantia por 3
0    : P3 ← Quantia \ 3         // Se resto for = 0
       P5 ← 0
1    : P3 ← (Quantia – 10 ) \ 3 // Se resto for = 1
       P5 ← 2
2    : P3 ← (Quantia – 5  ) \ 3 // Se resto for = 2
       P5 ← 1
Fim-caso
```

[2] Baseado no problema dos selos, apresentado em Salvetti & Barbosa (1998, p. 57)

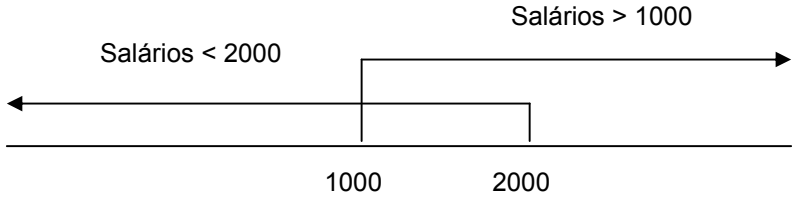
4.11. Erros Comuns (para iniciantes)

Em algumas expressões lógicas não ocorre erro de sintaxe, nem de execução, mas o resultado é diferente do esperado pelo programador.

- A expressão **A < B < C** pode ser avaliada como verdadeira, independente dos valores de A, B ou C. Exemplo: Supondo C = 1, tem-se:

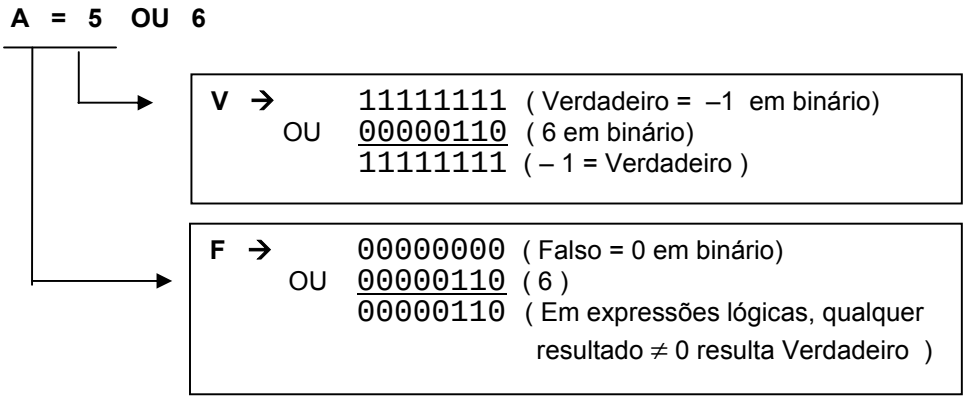


- Troca de "E" por "OU" para selecionar valores numa faixa.
  Exemplo: Selecionar salários entre 1000 e 2000
  A expressão **Salário > 1000 OU Salário < 2000** será sempre verdadeira, pois qualquer salário é > 1000 ou < 2000.



A expressão correta seria: Salário > 1000 E Salário < 2000

- Uso incorreto do "OU"
  Exemplo: A expressão A = 5 OU 6, será sempre verdadeira.



O operador = (relacional) não é distributivo em relação ao "OU" (lógico).
A expressão correta seria: A = 5 OU A = 6

4.12. Exercícios propostos

a) Equação do 2º grau

Dados a, b e c, a $\neq$ 0, calcular as raízes de $ax^2 + bx + c$ (equação do 2º grau).

Exemplo: Dados 1 -2 e 1 obter X = 1
Dados 1 -3 e 2 obter X1 = 1 e X2 = 2
Dados 1 1 e 2 obter "Não existem raízes reais"

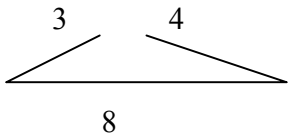
b) Triângulo (lados)

Dados 3 nºs positivos, possíveis lados de um triângulo, mostrar o tipo de triângulo.

Exemplo: Dados 3, 4 e 5
Obter "Escaleno", pois os 3 lados são diferentes um do outro.

Observação 1: No caso acima, não deve ser informado que o triângulo é retângulo, pois não foi pedida a classificação pelos ângulos.

Observação 2: Nem todo conjunto de 3 nºs, representando lados, formam triângulos, como no caso de 3, 4 e 8.



c) Política de descontos

Uma loja utiliza a seguinte política de descontos sobre o valor calculado da venda (quantidade vendida * preço unitário):

Valor calculado da venda	**% de desconto**
Até $ 200,00	Sem desconto
Acima de $ 200,00 até $ 1000,00	5%
Acima de $ 1000,00	10%

Dados a quantidade vendida e o preço unitário de um produto, calcular o valor a pagar, que considera a política de descontos.

Exemplo: Dado o preço unitário = $2,00 e a quantidade = 600, obter $1.080,00.
Dado o preço unitário = $2,00 e a quantidade = 400, obter $760,00.

d) Substitutiva

Em determinada escola, a prova substitutiva (PS) substitui a menor nota entre a P1 e a P2, se for maior que uma delas. Dadas as notas P1, P2 e PS, determinar o valor final da P1 e da P2, após a verificação da substituição de uma delas pela PS.

Exemplo: Dados P1 = 6, P2 = 4 e PS = 7,
Obter P1 = 6, P2 = 7

e) Triângulo (ângulos)

Dados 3 nºs positivos, possíveis ângulos de um triângulo, mostrar o tipo de triângulo.

Exemplo: Dados 30, 60 e 90
Obter "Retângulo"

f) Febre

Dada a temperatura de uma pessoa, mostrar sua situação, de acordo com as seguintes faixas de temperatura:

Temperatura	**Situação**
Abaixo de 36,5º	Hipotermia
De 36,5º até 37º	Normal
Acima de 37º até 38º	Estado febril
Acima de 38º	Febre

g) IMC (Índice de Massa Corporal)

Dados a massa (Kg) e a altura (m) de uma pessoa, calcular seu IMC e mostrar sua situação, de acordo com a seguinte tabela:

$$IMC = \frac{Massa}{Altura^2}$$

IMC	**Situação**
Abaixo de 20	Desnutrição
De 20 até 25	Normal
Acima de 25 até 30	Excesso de peso
Acima de 30 até 40	Obesidade leve
Acima de 40	Obesidade grave

h) Número de 3 algarismos

Dado um nº inteiro e positivo de 3 algarismos, obter:

- quantos algarismos têm valor ≥ 5;
- qual o menor algarismo;
- se a soma dos valores dos algarismos é par.

i) Multas

Dadas a velocidade de um veículo e a velocidade máxima permitida para um local, obter a multa e os pontos perdidos na carteira de motorista, de acordo com a seguinte tabela:

Velocidade do veículo	**Multa (R$)**	**Pontos**
Até 10% acima da velocidade permitida	0	0
Acima de 10% até 20% da velocidade permitida	120	4
Acima de 20% da velocidade permitida	520	7

Exemplo: Se um veículo trafegasse a 80 Km/h numa via com velocidade máxima permitida de 60 Km/h, e fosse pego pelo radar, a multa seria de R$ 520,00 e receberia 7 pontos na carteira, pois 80 Km/h está 33% acima da velocidade permitida. Se o veículo fosse pego a 70 Km/h, a multa seria de R$ 120 e receberia 4 pontos, pois 70 Km/h está entre 10% e 20% da velocidade permitida.

j) Objetivo

Descrever o objetivo do seguinte algoritmo:

PARA QUE SERVE?

Variáveis
N1, N2, N3, N4, X (Real)

Algoritmo ( )
Objetivo: _____
X ← N1 + N2
Se N1 + N3 > X então X ← N1 + N3
Se N1 + N4 > X então X ← N1 + N4
Se N2 + N3 > X então X ← N2 + N3
Se N2 + N4 > X então X ← N2 + N4
Se N3 + N4 > X então X ← N3 + N4
X ← X / 2

k) Senha

Dados os 4 algarismos de uma senha, verificar se é válida, ou seja, se não existem algarismos repetidos e se os algarismos não são consecutivos, sejam em ordem crescente ou decrescente.

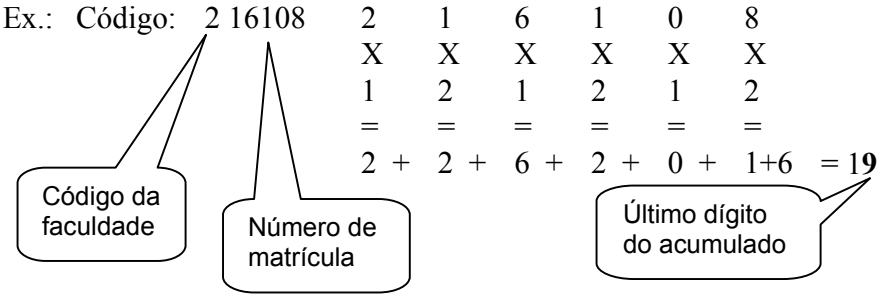
Exemplos:

- 1123 Inválida, pois contém algarismos consecutivos (123) e repetição (1)
- 1635 Válida
- 5935 Inválida, pois o algarismo 5 se repete

l) Dígito de controle

Dados o código da faculdade e número de matrícula de um aluno, calcular o dígito verificador, de acordo com o seguinte critério:

- Multiplicar cada dígito, da esquerda para direita por 1 2 1 2 1 2 e acumular os produtos.
- Se um dos produto for ≥ 10, somar os dígitos do produto.
- O dígito verificador será = 10 – último dígito do acumulado.
- Se o último dígito for = 0, o dígito verificador será = 0.



O dígito seria = 10 – 9 = 1, e o código completo, usado para digitação, seria 2 16108 – 1

No caso do digitador errar um número, o acumulado seria outro e o dígito também.

Uma coincidência seria a troca de 2 números com peso 1, ou cruzamento (2 X 1, 1 X 2) que resultaria num mesmo acumulado, e portanto num mesmo dígito de controle.

Exemplo: Na digitação do número anterior, o usuário troca o 1 com o 8, ou seja, digita 2 16801 – 1

Ex.: Código: 2 16801

2	1	6	8	0	8	
X	X	X	X	X	X	
1	2	1	2	1	2	
=	=	=	=	=	=	
2 +	2 +	6 +	1+6 +	0	2	= 19

A soma é a mesma. Portanto, temos o mesmo dígito, e o erro não seria percebido.

m) Folha de pagamento

Tendo-se o salário bruto (já descontadas as faltas), o número de dependentes e o nº de filhos menores de 14 anos, de um dado funcionário, calcular o salário líquido a receber, de acordo com as seguintes regras:

Salário Líquido	Salário Bruto – INSS – Imposto de Renda + Salário Família
INSS	Aplicar alíquota (%) de INSS sobre o Salário Bruto. O teto para pagamento de INSS é de R$ 176,00.
Salário Base de IR	Salário Bruto – INSS – R$ 100,00 por dependente.
IR	Aplicar a alíquota (%) de imposto (coluna 2 da tabela 1) sobre o Salário Base de IR. Do valor calculado, tirar o valor a Deduzir (coluna 3 da tabela 1).
Salário Família	R$ 12,00 por filho menor de 14 anos, para salários até R$ 500,00.

Tabela 1 – Tabela progressiva de IR

Salário Base de IR (R$)	**Alíquota (%)**	**Deduzir (R$)**
Até 1.000,00	Isento	
Acima de 1.000,00 até 2.000,00	15	150,00
Acima de 2.000,00	27,5	400,00

Tabela 2 – Alíquotas para INSS

Salário Bruto (R$)	**Alíquota (%)**
Até 500,00	7,65
Acima de 500,00 até 600,00	8,65
Acima de 600,00 até 800,00	9
Acima de 800,00 até 1600,00	11

Exemplo: Calcular o salário líquido de um funcionário com salário bruto de R$ 1.500,00 e 2 dependentes para IR, sendo um filho menor de 14 anos.

INSS = 176,00 (teto)
Salário Base de IR = 1500,00 – 176,00 – 200,00 = 1.124,00
IR = (15 % de 1.124,00) – 150,00 = 18,60
Salário Líquido = 1.500,00 – 176,00 – 18,60 = 1.305,40

n) Rodízio

Dado o final da placa de um automóvel, obter qual o dia da semana que este veículo está sujeito ao rodízio. Para o final de placa 1 e 2, o veículo não roda às segundas, para o final 3 e 4, às terças, e assim por diante.

Exemplo: Dado o final 7, a saída será “Quinta”

o) Desconto

Um Hotel adotou a seguinte política de descontos na reserva de pacotes para o *Reveillon*:

- Estudante → R$ 50,00 de desconto
- Trabalhador da Indústria → R$ 30,00 de desconto
- Sócio do Clube de Viagem → R$ 80,00 de desconto
- Aposentado → R$ 100,00 de desconto

Os descontos para Estudante, Industriário e Sócio não são cumulativos, e o hóspede que se enquadrar em mais de uma categoria recebe o maior deles. Já o desconto para Aposentado é cumulativo sobre os demais.

O hóspede informa sua situação em cada categoria: Estudante ("S" ou "N"); Trabalhador da Indústria ("S" ou "N"); Sócio do Clube de Viagem ("S" ou "N"); Aposentado ("S" ou "N").

Dadas as categorias de um hóspede, calcular o desconto a que terá direito.

(Oliveira, 2005).

p) Escola de samba

Na apuração das notas de um desfile de escolas de samba utiliza-se um sistema em que são desprezadas a nota mais baixa e a mais alta, para evitar perseguições ou favorecimentos.

Dadas as 4 notas de um quesito qualquer (bateria, evolução, ...), obter a soma das notas que não foram desprezadas. Se houver notas iguais, considerar a primeira ocorrência da nota.

Exemplo: Dadas as notas 8, 6, 7, 10 num determinado quesito, obter 15.

Verifique, também, se a sua estratégia é boa para 5 notas ou mais.

q) Sobreposição

Dados dois intervalos, representados por 4 números, [a, b] e [c, d], verificar se existe sobreposição (pontos em comum) entre os intervalos. As extremidades dos intervalos podem não estar em ordem crescente. O intervalo [a, b] pode não estar antes de [c, d].

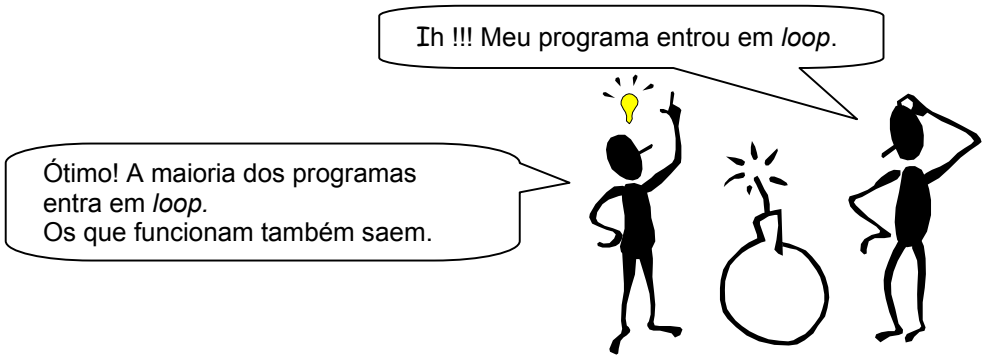
Exemplo: Dados os intervalos [3, 7] e [4, 8] a resposta é "Sim"

r) Paralelas

Dadas as coordenadas de dois pontos distintos (X1, Y1) e (X2, Y2), verificar que tipo de reta passa por esses pontos: eixo, paralela a um eixo, não paralela a um eixo.

Exemplo: Dadas as coordenadas (3,2) e (6,2), a reta é paralela ao eixo das abscissas.
Dadas as coordenadas (3,1) e (6,4), a reta não é paralela a nenhum eixo.

5. CONSTRUTOR LÓGICO DE REPETIÇÃO



É usado para repetir um bloco de instruções, sendo a quantidade de repetições controlada por uma variável ou por uma condição, verificada antes ou após o bloco de instruções.

5.1. N primeiros pares

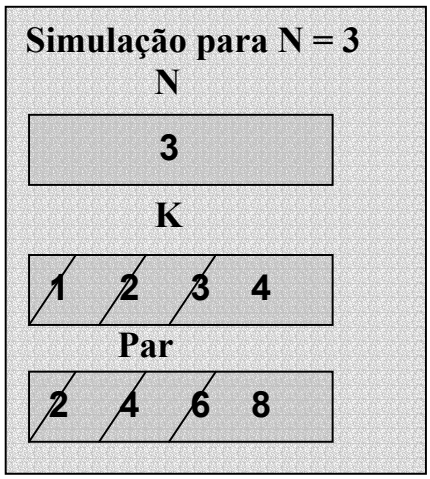
Imprimir os N primeiros números pares positivos.

Uma solução

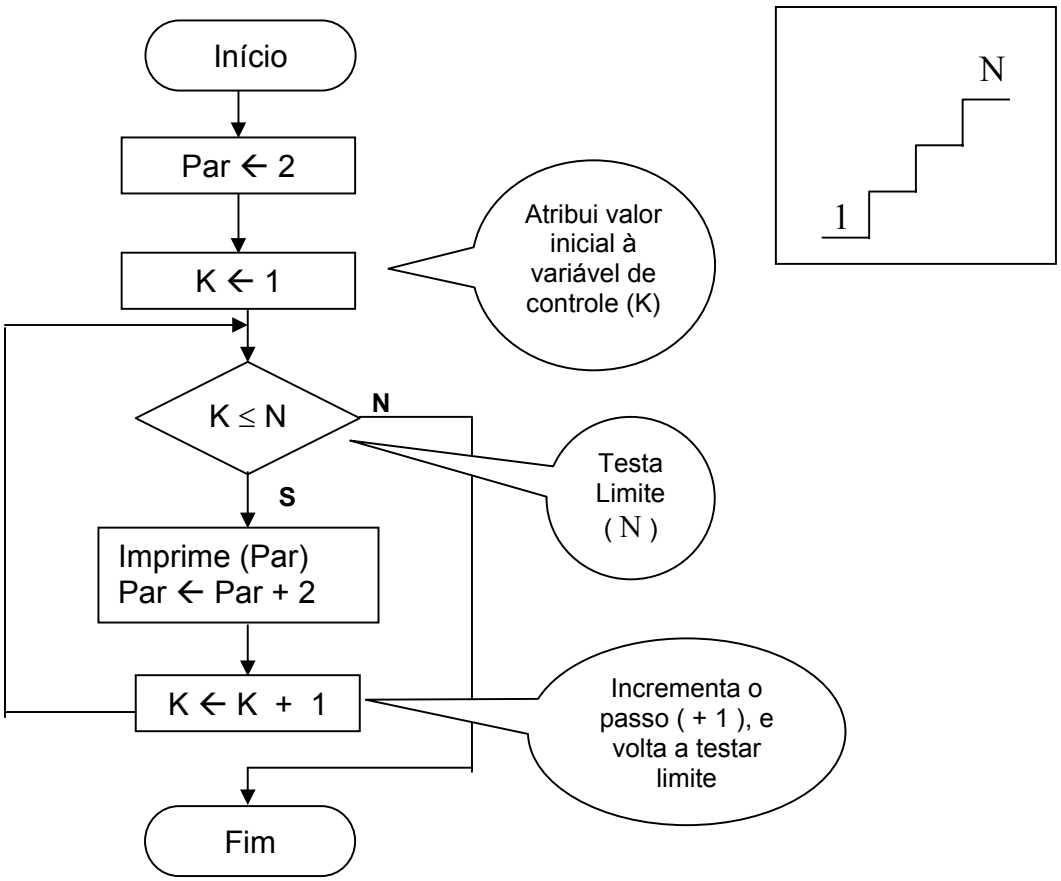
Podemos iniciar a variável "Par" com o valor do primeiro par (2). Para calcular o próximo par pode-se somar 2 na variável Par, e repetir o processo para a quantidade de pares a serem impressos.

Para contar as repetições (voltas do *loop*), podemos usar uma variável (K, por exemplo) que inicia com 1 e aumenta 1 a cada volta, como em uma catraca, usada para contar pessoas que passam por ela. O controle das repetições é feito comparando, a cada volta, o valor de K com o limite (N). O *loop* termina quando K fica maior que N.

```
Par ← 2
K ← 1
Teste: K ≤ N (Verdadeiro)
Imprime (Par)                  1º par
Par ← Par + 2                  próximo par
K ← K + 1
Teste: K ≤ N (Verdadeiro)
Imprime (Par)                  2º par
Par ← Par + 2                  próximo par
K ← K + 1
Teste: K ≤ N (Verdadeiro)
Imprime (Par)                  3º par
Par ← Par + 2                  próximo par
K ← K + 1
Teste: K ≤ N (Falso. O loop termina)
```



O seguinte fluxograma representa a solução do problema. A variável K é responsável pelo controle de repetição, pois é ela que é iniciada com 1, é testada contra o limite de repetições (N) e é incrementada com o passo (1) a cada volta, ou iteração. [3]



PARES	
Inteiro N	// *Quantidade de pares a serem impressos*
Inteiro Par	// *Números pares*
ImprimePares ( )	
Objetivo: Imprimir os N primeiros pares positivos	
Inteiro K	// *Variável de controle de repetição*
Par ← 2	
K ← 1	
Enquanto K ≤ N faça	
Saída (Par)	// *Imprime um par*
Par ← Par + 2	//*Calcula o próximo par*
K ← K + 1	// Conta + 1 volta
Fim-Enquanto	

[3] Os problemas envolvendo iterações podem ser resolvidos em duas etapas: 1) Identificar o que deve ser repetido; 2) Quantas vezes será repetido e como controlar a quantidade de repetições.

O controle de repetição pode ser efetuado por uma instrução especial para o caso de se saber de antemão quantas voltas serão executadas (controle feito por variável).

```
PARES
Inteiro N                          // Quantidade de pares a serem impressos
Inteiro Par                        // Números pares
ImprimePares ( )
Objetivo:  Imprimir os N primeiros pares positivos
Inteiro K                          // Variável de controle de repetição
Par     ← 2
Para  K de 1 até N  [passo 1]  repita
      Saída (Par)                  // Imprime um par
      Par ← Par + 2                //Calcula o próximo par
Fim-Para
```

Nesse caso, o comando faz automaticamente a inicialização da variável de controle (K), acumula o passo (=1, por *default*), e volta para fazer o teste.

Algumas linguagens utilizam uma forma de instrução de controle por variável que faz a atribuição do valor inicial, o teste de final de *loop* e o avanço da variável de controle (passo), de forma explícita:

```
PARES
Inteiro N                          // Quantidade de pares a serem impressos
Inteiro Par                        // Números pares
ImprimePares ( )
Objetivo:  Imprimir os N primeiros pares positivos
Inteiro K                          // Variável de controle de repetição
Par     ← 2
Para  (K ← 1;  K ≤ N; K ← K + 1) repita
      Saída (Par)                  // Imprime um par
      Par ← Par + 2                //Calcula o próximo par
Fim-Para
```

Simulação

Antes de fazer a simulação, devemos numerar as instruções do algoritmo.

Instrução	Algoritmo
	ImprimePares ( )
	Inteiro K
1	Par ← 2
2	Para K de 1 até N [passo 1] repita
3	Saída (Par)
4	Par ← Par + 2
	Fim-Para

Supondo que o usuário solicitou a impressão dos 3 primeiros pares, o algoritmo irá funcionar da seguinte forma:

Instrução	Variáveis			Teste	Saída
	N	Par	K	K ≤ N	
	3				
1	3	**2**			
2	3	2	**1**	**V** (1 ≤ 3)	
3	3	2	1		**2**
4	3	**4**	1		
2	3	4	**2**	**V** (2 ≤ 3)	
3	3	4	2		**4**
4	3	**6**	2		
2	3	6	**3**	**V** (3 ≤ 3)	
3	3	6	3		**6**
4	3	**8**	3		
2	3	8	**4**	**F** (4 > 3)	

Na simulação rápida de algoritmos que contém *loops*, é interessante separar, em linhas diferentes, as mudanças dentro do *loop* (uma linha para cada volta):

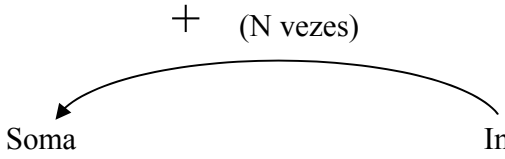
Variáveis			
N	Par	K	Saída
3	2		
	4	1	2
	6	2	4
	8	3	6
		4	

5.2. Impares

Dado um número N, inteiro e positivo, obter a soma dos N primeiros ímpares.

Uma solução

O resultado do algoritmo estará na variável Soma, que começa com 0. Utilizamos a variável Impar, com valor inicial = 1, para calcular a seqüência de N ímpares que serão acumulados em Soma.



0 1 4 9 16 ... 1 3 5 7 ... N impares, N somas

O algoritmo tem as seguintes macro-instruções, que serão depois refinadas:

Iniciar { Zerar Soma; Determinar 1º Termo (= 1)

Repetir (N vezes) { Acumular Termo em Soma; Calcular próximo Termo (+ 2)

Finalizar { Imprimir Soma

IMPARES	
Inteiro N	*// Quantidade de ímpares que serão acumulados*
Inteiro Soma	*// Acumulado dos N primeiros ímpares*
SomaImpares ( )	
Objetivo: Imprimir a soma dos N primeiros ímpares	
Inteiro K	
Soma ← 0	
Impar ← 1	
Para K de 1 até N repita	
Soma ← Soma + Impar	// *Acumula impar*
Impar ← Impar + 2	// *Calcula próximo impar*
Fim-Para	

O modo como está sendo gerada a seqüência dos números ímpares é chamado de relação de recorrência, em que o cálculo do próximo termo é feito a partir de um ou mais termos anteriores.

No caso dos números ímpares, cada novo número impar é obtido somando-se 2 ao impar anterior, sendo o primeiro impar = 1.

$Impar_1 = 1$
$Impar_2 = Impar_1 + 2 = 1 + 2 = 3$
$Impar_3 = Impar_2 + 2 = 3 + 2 = 5$

A relação de recorrência é $Impar_{k+1} = Impar_k + 2$, $k = 1,2,3, ...$ $Impar_1 = 1$

Na linguagem algorítmica, a relação de recorrência é implementada pelo comando de atribuição, que contém dois momentos de uma mesma variável, o valor futuro e o valor atual: Impar ← Impar + 2 (Lê-se: Impar será igual ao valor atual de Impar, mais 2.)

Termo geral

Termo geral é uma fórmula usada para calcular qualquer elemento de uma seqüência em função de sua posição (índice) na seqüência.

O termo geral dos números ímpares é = 2 * K – 1, para K = 1, 2, 3, ...

Neste exercício, trata-se de obter a somatória dos números com formato 2*K – 1, para K pertencente a N*

$$\sum_{K=1}^{N} 2 \cdot K - 1 = 1 + 3 + 5 + 7 + \ldots + 2N - 1$$

O operador $\sum$ traz em si o conceito de iteração.

Lê-se: somatória de termos no formato 2*K – 1, para K variando de 1 até N, de 1 em 1. Para N = 6, por exemplo, temos:

$$\sum_{K=1}^{6} 2 \cdot K - 1 = 1 + 3 + 5 + 7 + 9 + 11 = 36$$

Quando o termo geral é usado não é necessário definir um valor inicial para o termo da seqüência, pois necessita-se apenas da posição do termo na seqüência. A posição do termo é dada pela variável de controle da somatória (K).

Uma maneira de descobrir o termo geral é listar os valores dos termos e as posições que eles ocupam, procurando uma relação entre esses dois conjuntos:

Termo	→	1	3	5	7	9	...
Posição (K)	→	1	2	3	4	5	...

A relação entre a posição (K) e o Termo (Impar) é: $\text{Impar}_K = 2 * K - 1$

No termo geral, cada termo é função da posição (K), ou seja, $\text{Termo}_K = f(K)$

Na relação de recorrência, o primeiro termo é definido e cada termo a seguir é função do anterior, ou seja: $\text{Termo}_K = f(\text{Termo}_{K-1})$

```
IMPARES
Inteiro N                    // Quantidade de ímpares que serão acumulados
Inteiro Soma                 // Acumulado dos N primeiros ímpares
SomaImpares ( )
Objetivo: Imprimir a soma dos N primeiros ímpares
Inteiro K
Soma ← 0
Para   K de 1 até N  repita
       Impar ← 2 * K – 1          // Calcula impar
       Soma ← Soma + Impar        // Acumula impar
Fim-Para
```

5.3. P.A.

Imprimir os N primeiros termos de uma P.A. definida por $a_1 = 2$ e razão = 3.

Uma solução

Pode-se gerar os termos da P.A. pela relação de recorrência e pelo termo geral.

Recorrência	**Termo Geral**
$a_1 = 2$	$a_1 = 2$
$a_2 = a_1 + 3 = 5$	$a_2 = a_1 + 3 = 5$
$a_3 = a_2 + 3 = 8$	$a_3 = a_1 + 6 = 8$
$a_4 = a_3 + 3 = 11$	$a_4 = a_1 + 9 = 11$
...	...
$a_n = a_{n-1} + r$	$a_n = a_1 + (n - 1)\, r$

Recorrência

O primeiro termo é = 2 e cada termo é gerado somando-se a razão (=3) ao anterior.

```
P. A.
Inteiro N                    // Quantidade de termos da P.A.
Termo                        // Valor do termo
ImprimeTermos ( )
Objetivo: Imprimir os N primeiros termos da P.A. ( a1 = 2 e razão = 3).
Inteiro K
Termo ← 2
Para   K de 1 até N  repita
       Imprime ( Termo )
       Termo  ← Termo  + 3        // Termo anterior + razão
Fim-Para
```

Termo geral

Cada termo é calculado em função da posição do termo na seqüência. A posição do termo na seqüência pode ser dada pela variável de controle do *loop* (1ª volta do loop = 1º termo da seqüência, e assim por diante). [4]

P. A.	
Inteiro N	*// Quantidade de termos da P.A.*
Termo	*// Valor do termo*
ImprimeTermos ( )	
Objetivo: Imprimir os N primeiros termos da *P.A. (a_1 = 2 e razão = 3).*	
Inteiro K	
Para K de 1 até N repita	
Termo ← 2 + (K – 1) * 3	// $a_k \leftarrow a_1 + (K - 1)$ razão
Imprime (Termo)	
Fim-Para	

5.4. Números triangulares

Dado um número N, inteiro e positivo, imprimir o N-ésimo elemento da seguinte seqüência: 1 3 6 10 15 ...

Uma solução

Os números são chamados de triangulares devido a essa formação:

				Soma
		1		1
	1	2		3
1	2	3		6
1	2	3	4	10

Não se trata de uma progressão aritmética. O acréscimo de um termo para outro também é variável:

Termo→	1	3	6	10	15	....
Acréscimo →		+ 2	+ 3	+ 4	+ 5	

Como o problema pede apenas o valor do N-ésimo termo, inicia-se o primeiro termo com 1 e calcula-se mais (N – 1) termos, além do primeiro. A variável de controle de

[4] Johann Carl Friedrich Gauss, com 7 anos de idade, ao receber a tarefa de somar os números inteiros de 1 a 100, respondeu, imediatamente, que a soma era 5050 (50 pares de números, cada par somando 101).

repetição começa com 2 e vai até N (do 2º até o último termo). O valor do termo após a última volta (N) é o valor do N-ésimo termo.

```
NÚMEROS TRIANGULARES
Inteiro N                          // Ordem do termo a ser calculado
Inteiro Termo                      // Conteúdo do N-ésimo termo
Triangulares ( )
Objetivo: Obter o N-ésimo termo da seqüência dos números triangulares
Inteiro K
Inteiro A                          // Acréscimo
Termo ← 1
A ← 2                              // Valor do primeiro acréscimo
Para  K de 2 até N  repita
      Termo ← Termo + A
      A ← A + 1                    // Próximo acréscimo, na próxima volta
Fim-Para
```

Nota-se que a variável A (acréscimo) tem o mesmo comportamento da variável K, usada para controle do *loop*. Assim, pode-se utilizar apenas uma delas, de acordo com a descrição a seguir:

```
NÚMEROS TRIANGULARES
Inteiro N                          // Ordem do termo a ser calculado
Inteiro Termo                      // Conteúdo do N-ésimo termo
Triangulares ( )
Objetivo: Obter o N-ésimo termo da seqüência dos números triangulares
Inteiro K
Termo ← 1
Para  K de 2 até N  repita
      Termo ← Termo + K
Fim-Para
```

Usando a relação de recorrência, tem-se:

$T_1 = 1$

$T_2 = T_1 + 2 = 1 + 2 = 3$

$T_3 = T_2 + 3 = 3 + 3 = 6$

$T_4 = T_3 + 4 = 6 + 4 = 10$

A relação de recorrência é: $T_{k+1} = T_k + (k + 1)$, $k = 1, 2, 3 \ldots$, $T_1 = 1$, ou

$$T_k = T_{k-1} + k \text{ , } \quad k = 2, 3, 4 \ldots \text{ , } T_1 = 1$$

O cálculo do termo seguinte é dado pelo seguinte comando de atribuição, a partir da relação de recorrência Termo ← Termo + K, no qual K é a ordem do termo que está sendo calculado.

Se o termo geral dessa seqüência for encontrado, não é necessário um processo interativo para obter-se o N-ésimo termo. Basta aplicar o valor de N na fórmula.

5.5. Fatorial

Dado um número inteiro e positivo N, calcular N!

Uma solução

A função fatorial é definida por: $N! = \prod_{k=1}^{N} k$ para todo $N \geq 0$.

Por exemplo: 5! = 1 . 2 . 3 . 4 . 5 = 120

Por se tratar de um produtório, inicia-se a resposta (variável Fatorial) com 1 (elemento neutro na multiplicação). Em seguida, multiplica-se Fatorial pelos inteiros até N. Pode-se começar a multiplicar por 2, pois a multiplicação por 1 não altera o resultado.

NÚMERO INTEIRO	
Inteiro N	// *Fatorial a ser calculado*
Inteiro Fatorial	// *N!*
Triangulares ( )	
Objetivo: Calcular N!	
Inteiro Fator	
Fatorial ← 1	
Para Fator de 2 até N repita	
Fatorial ← Fatorial * Fator	
Fim-Para	

O cálculo pode ser feito com a seqüência de fatores na ordem inversa, ou seja:
5! = 5 . 4 . 3 . 2 . 1 = 120. A variável Fator inicia com 5 e vai até 2, de –1 em –1. Para isso, usa-se o comando de repetição com passo negativo.

NÚMERO INTEIRO	
Inteiro N	// *Fatorial a ser calculado*
Inteiro Fatorial	// *N!*
Triangulares ( )	
Objetivo: Calcular N!	
Inteiro Fator	
Fatorial ← 1	
Para Fator de N até 2 **passo –1** repita	
Fatorial ← Fatorial * Fator	
Fim-Para	

5.6. MDC

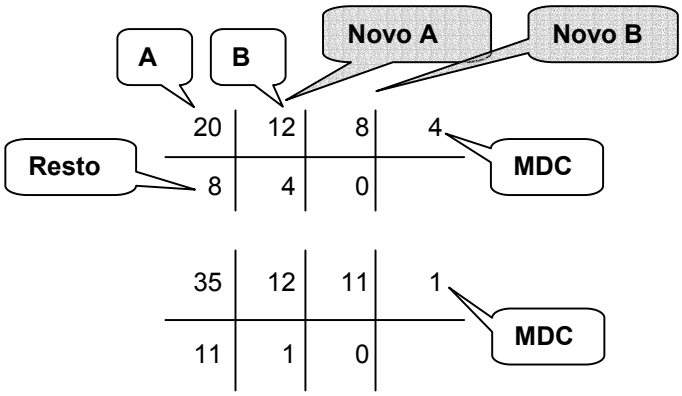
Dados 2 números inteiros positivos, diferentes de 0, calcular o Máximo Divisor Comum (MDC) deles, pelo método de Euclides.

Uma solução

O método (algoritmo) de Euclides para cálculo do MDC contém os seguintes passos: Calcular o resto da divisão do 1º número pelo 2º. Se o resto for = 0, o MDC é o 2º número. Senão, substituímos o 1º número pelo 2º, e 2º número pelo resto, e calculamos um novo resto.

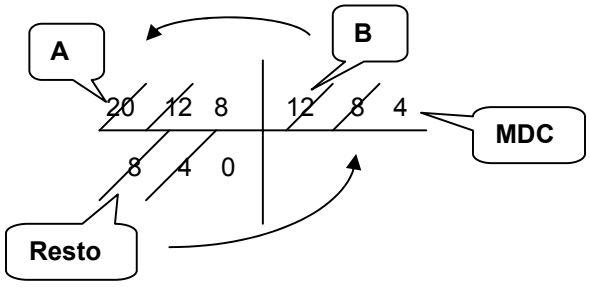
Exemplo: Calcular o MDC entre 20 e 12.

Chamando o primeiro número de A e o segundo de B, podemos representar o processo da seguinte forma:



O MDC será o último valor de B.

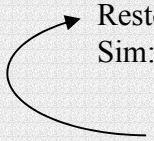
Outra forma de representar o processo é:



Traduzindo o processo para uma linguagem algorítmica temos:

```
Resto ← A mod B
Resto ≠ 0 ?    Não → Said do loop e MDC ← B
Sim:  A ← B
      B ← Resto
```

Para que o teste não fique no meio do *loop* pode-se repetir o cálculo do resto antes de retornar para o teste.



```
Resto ← A mod B
Resto ≠ 0 ?     Não: Sai do loop e MDC ← B
Sim:  A ← B
      B ← Resto
      Resto ← A mod B
```

No caso do MDC, não se sabe de antemão quantas vezes o *loop* será executado. Utiliza-se, então, um tipo de *loop* que permite a repetição de um bloco de instruções enquanto uma determinada condição for verdadeira.

```
2 NÚMEROS INTEIROS
Inteiro A, B, MDC
CalcMDC ( )
Objetivo: Calcular o MDC de 2 números inteiros
Inteiro Resto
Resto ← A mod B
Enquanto Resto ≠ 0  faça
      A      ← B
      B      ← Resto
      Resto  ← A mod B
Fim-Enquanto
MDC ← B
```

Se a condição for verdadeira, o bloco de instruções será executado até o fim e volta-se para o início do ciclo, para testar novamente a condição. Se a condição continuar verdadeira, executa-se novamente o bloco e assim por diante. Se a condição tornar-se falsa, o programa prossegue após o fim da estrutura de repetição (Fim-Enquanto).

Nesse construtor de repetição é necessário iniciar a variável de controle do *loop* e incrementar o contador de iterações. Sempre que for possível determinar o número de "voltas" de um *loop*, é melhor utilizar o controle de repetição por variável, não por condição.

Observa-se que se A é múltiplo de B o *loop* não é executado e o MDC = B.

Observa-se, também, que se o usuário informar A < B, o próprio algoritmo corrige a inversão.

12	20	12	**...**
12	8		

5.7. Seqüência Oscilante

Dado um número N, inteiro e positivo, imprimir os N primeiros termos da seguinte seqüência: 1 –2 3 –4 5 –6 ...

Estratégia 1: Intercalar ímpares positivos com pares negativos, ou seja, a seqüência 1 3 5 7 ... intercalada com a seqüência –2 –4 –6 ...

Impares positivos → 1 3 5 7
Pares negativos → –2 –4 –6 ...

A cada uma das N iterações calcula-se um impar positivo ou um par negativo, dependendo do nº da volta (voltas ímpares → impar positivo; voltas pares → par negativo). O nº da volta será controlado pela variável K, que será testada para saber se é uma volta de ordem par ou impar.

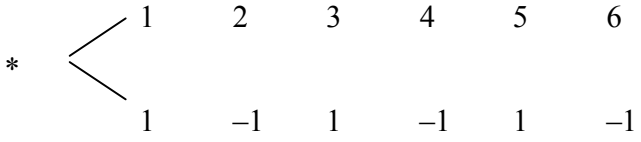
O primeiro impar positivo é 1. Os próximos serão obtidos somando-se 2 na variável Impar. O primeiro par negativo é –2. Os próximos serão obtidos subtraindo-se 2 da variável Par.

```
SEQÜÊNCIA OSCILANTE
Inteiro N                    // Quantidade de termos
Inteiro Termo                // Termos da seqüência
GerarSeq ( )
Objetivo: Gerar seqüência oscilante
Inteiro Impar, Par, K
Impar  ← 1
Par    ← –2
Para  K de 1 até N  repita
      Se   K mod 2 = 0       // Se K é divisível por 2, é uma volta de ordem par
           Termo ← Par
           Par ← Par – 2
      Senão
           Termo ← Impar
           Impar ← Impar + 2
      Fim-se
      Imprima (Termo)
Fim-Para
```

Uma variação dessa estratégia, baseada também na ordem da volta, utiliza a própria variável de controle, nas voltas pares como par negativo (– K) e nas voltas ímpares como impar positivo (K).

```
SEQÜÊNCIA OSCILANTE
Inteiro N, Termo
GerarSeq ( )
Objetivo: Gerar seqüência oscilante
Inteiro K
Para  K de 1 até N  repita
      Se    K mod 2 = 0
            Termo ← – K
      Senão
            Termo ← K
      Fim-se
      Imprima (Termo)
Fim-Para
```

Estratégia 2: Números naturais multiplicados por oscilação de sinal.



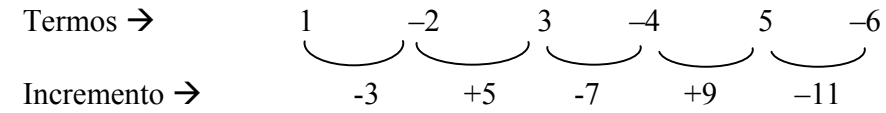
Percebe-se a existência da seqüência: 1 2 3 4 5 6 multiplicada, respectivamente, pela seqüência 1 –1 1 –1 1 –1 .

A primeira seqüência pode ser o próprio contador de voltas do *loop*, pois se comporta da mesma maneira (Início = 1 e incrementos de 1 em 1).

Na segunda seqüência, pode-se obter a oscilação por meio de uma variável de sinal, que começa com 1, e a cada volta é multiplicada por –1.

```
SEQÜÊNCIA OSCILANTE
Inteiro N, Termo
GerarSeq ( )
Objetivo: Gerar seqüência oscilante
Inteiro K, Sinal
Sinal ← 1
Para  K de 1 até N  repita
      Termo ← K * Sinal
      Imprima (Termo)
      Sinal ← – Sinal
Fim-Para
```

Estratégia 3: Pêndulo



Nota-se que a seqüência se comporta como um pêndulo, indo para trás 3 e para frente 5, para trás 7 e para frente 9, sendo que cada oscilação segue um padrão, dado pela seqüência dos ímpares, a partir do-3.

```
GerarSeq ( )
Inteiro K, Sinal, Termo
Sinal   ← –1
Termo  ←  1
Para  K de 1 até N  repita
        Imprima (Termo)
        Termo ← Termo + Sinal * (2 * K + 1)
        Sinal ← – Sinal
Fim-Para
```

Estratégia 4: Termo Geral

Quando se tem o termo geral da seqüência, é simples gerá-la, bastando aplicar na fórmula do termo geral, o valor da variável de controle do *loop* (K = 1 até N).

Posição do Termo (K)	Valor do Termo	$K * -1^{(K+1)}$
1	1	1 * 1
2	–2	2 * –1
3	3	3 * 1
4	–4	4 * –1
5	5	5 * 1

O termo geral dessa seqüência é:
$-1^{(K+1)} * K$
ou
$-1^{(K-1)} * K$

```
GerarSeq ( )
Inteiro K
Para  K de 1 até N  repita
        Termo ← (– 1) ^ ( K + 1 ) *  K
        Imprima (Termo)
Fim-Para
```

5.8. Wallis

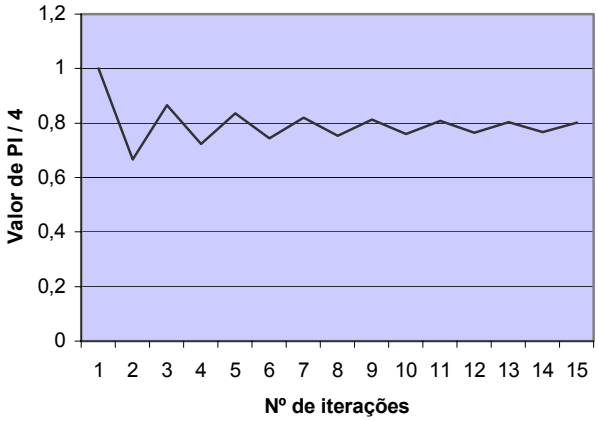
Dado um número N, inteiro e positivo, calcular o valor aproximado, com N termos, da série de Wallis:

$$1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \ldots$$

Uma solução

No limite, a série converge para Π / 4, ou seja, a medida que são acumulados novos termos, o valor calculado é um valor mais preciso de Π / 4 (0.7854)

Termo	Valor do termo	Valor acumulado
1	1	1
2	– 1 / 3	0.6667
3	1 / 5	0.8667
4	– 1 / 7	0.7238
5	1 / 9	0.8349
6	– 1 / 11	0.7440
7	1 / 13	0.8209
8	– 1 / 15	0.7543



Cada termo pode ser decomposto em numerador e denominador. No 1º termo o numerador vale 1 e o denominador também. Os próximos termos serão gerados multiplicando-se o numerador por –1 e somando 2 no denominador. Usa-se uma variável acumuladora dos termos calculados, iniciada com 0.

```
Wallis
Inteiro N                    // Quantidade de termos
Real Soma                    // Soma dos termos (valor aproximado de Π / 4)
Calcular ( )
Objetivo: Calcular a série de Wallis, com N termos
Inteiro  K, Num, Den
Soma   ← 0
Num    ← 1
Den    ← 1
Para  K de 1 até N  repita
      Soma ← Soma + Num / Den
      Num  ← – Num
      Den  ← Den + 2
Fim-Para
```

5.9. Fibonacci

Dado um número N, inteiro e positivo, obter os N primeiros termos da seguinte seqüência: 1 1 2 3 5 8 13 ...

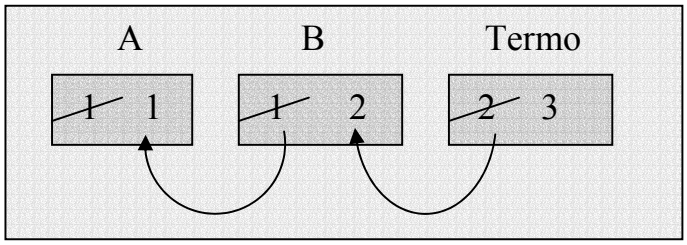
Uma solução

Um dos problemas do livro Liber Abaci, sobre técnicas algébricas, de Leonardo de Pisa (Fibonacci), em 1202 foi: "Quantos casais de coelhos serão produzidos num ano, começando com um só par, se em cada mês cada casal gera um novo par que se torna produtivo a partir do segundo mês?". Esse problema dá origem à seqüência de Fibonacci. (Boyer, 1974:186)

A tabela a seguir mostra a evolução das variáveis do problema:

Período (Mês)	Casais jovens	Casais adultos (reprodutivos)	Total de casais
1	1	0	1
2	0	1	1
3	1	1	2
4	1	2	3
5	2	3	5
6	3	5	8
7	5	8	13
8	8	13	21
9	13	21	34
10	21	34	55
11	34	55	89
12	55	89	144

Cada termo, a partir do terceiro, é obtido somando-se os 2 termos anteriores. Os 2 primeiros são iguais a 1.



Pode-se usar duas variáveis, A e B, contendo inicialmente o valor 1. A soma de A e B gera um novo Termo, com valor 2. Para continuar o processo, pode-se deslocar os valores de B e Termo para A e B e voltar a somar A e B, gerando um novo Termo, com valor 3, e assim por diante.

```
FIBONACCI
Inteiro N                          // Quantidade de termos
Inteiro Termo                      // Termos da seqüência
GerarSeq ( )
Pré Condição N > 2
Objetivo: Gerar os termos da seqüência de Fibonacci
Inteiro K, A, B, C
A ← 1
B ← 1
Imprima (A, B)
Para K de 3 até N repita
      Termo ← A + B
      Imprima (Termo)
      A ← B
      B ← Termo
Fim-Para
```

K inicia com 3, pois 2 termos já foram impressos, e o programa pede N termos

A solução a seguir, mais criativa, usa uma variável a menos.

```
FIBONACCI
Inteiro N, Termo
GerarSeq ( )
Inteiro K, A
Termo ← 1
A ← 0
Para K de 1 até N repita
      Imprima (Termo)
      Termo ← Termo + A
      A ← Termo – A
Fim-Para
```

Termo	Razão entre 2 termos
1	
1	1
2	2
3	1,5
5	1,6667
8	1,6
13	1,625
21	1,6153
34	1,6190
55	1,6176
89	1,6181

A razão entre um termo e o anterior, no limite, tende a Φ (phi maiúscula), e é chamada de razão ou proporção áurea.

Na geometria do pentágono regular, se um lado AB tiver comprimento unitário, qualquer diagonal AC é =

$$\frac{(1 + \sqrt{5})}{2} = 2 \cdot \cos \Pi / 5 = 1{,}61803 \ldots = \Phi$$

A razão áurea é encontrada na natureza na distribuição das sementes de girassol e do cacto, que é regida pela espiral logarítmica do número áureo. O girassol possui 55 espirais orientadas no sentido horário, sobrepostas a 34 ou 89 espirais em sentido anti-horário

No corpo humano podemos descobrir o significado metafísico do Φ tal como expresso pelo aforismo de Heráclito: "O homem é a medida de todas as coisas". As medidas da figura humana se ajustam ao antigo símbolo biométrico do corpo dividido em dois pelos órgãos sexuais ou em Φ pelo umbigo (Lawlor, 1996:59).

As frações contínuas fornecem a seguinte fórmula: $\Phi = 1 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{1 + \ldots}}}}$

5.10. Acumular seqüência

Usando um processo iterativo, obter quantos termos são necessários para que a soma dos primeiros termos de uma P.A., definida pelo valor do primeiro termo e pela razão, seja maior que 100.

Uma solução

Pode-se criar um processo repetitivo que acumule cada termo da PA e verifique se a soma é superior a 100, contando quantas iterações ("voltas") foram realizadas.

O termo geral da P.A. é: $a_n = a_1 + (n - 1) * \text{razão}$

O esboço da solução é, dados a_1 e a razão:

Iníciar
- Zerar Soma
- Zerar contador de termos (quantidade de voltas)

Repetir
- Somar 1 no contador de termos
- Calcular Termo (usar contador de termos na fórmula do termo geral)

(até que Soma > 100)
- Acumular Termo em Soma

Finalizar
- A resposta é o contador de termos.

A estrutura de repetição usada nessa solução é com o teste no final, ou seja, o bloco de repetição é feito pelo menos uma vez, e será repetido até que a condição seja verdadeira.

```
PROGRESSÃO ARITMÉTICA
Inteiro A1, Razão                    // Definição da P.A.
Inteiro N                            // Quantidade de termos
Triangulares ( )
Objetivo:  Obter a quantidade de termos para que a soma de uma PA seja > 100
Inteiro Termo
Soma  ← 0
N     ← 0
Repita
    N     ← N + 1
    Termo ← A1 + (N – 1) * Razão
    Soma  ← Soma + Termo
Até que Soma > 100
Imprima (N)
```

5.11. Seqüência com aviso de fim

Uma situação muito comum é o tratamento arquivos, em que não se sabe antecipadamente a quantidade de registros. Nesse caso, é comum testar o fim da seqüência usando uma variável que muda de estado quando ele ocorre. Esta variável é chamada de sentinela.

No caso de arquivos existe uma função, uma variável ou uma propriedade chamada EOF (**E**nd **O**f **F**ile [5]), que se torna verdadeira quando se tenta ler o próximo registro e o arquivo terminou. O seguinte trecho de algoritmo é comum no tratamento seqüencial de arquivos:

```
Ler 1º registro
Enquanto não for o fim do arquivo faça
      Tratar registro lido
      Ler o próximo registro
Fim-Enquanto
```

5.12. Alfabeto

Imprimir as letras maiúsculas do alfabeto.

Uma solução

Podemos imprimir cada uma das letras, uma por uma, o que não parece uma boa solução.

Pode-se, também, utilizar operadores/funções para tratamento de caracteres e *strings* (cadeias de caracteres), para gerar uma letra a partir do valor da anterior.

[5] **EOF** não significa **É O F**im, em português.

As principais funções e operadores para tratamento de caracteres e *strings* são:

FUNÇÃO/ OPERADOR	**DESCRIÇÃO**	**EXPRESSÃO**	**VALOR**
+	Concatenação	"AB–" + "C"	"AB–C"
Val (*string*)	Valor numérico do *string*	Val ("45.3")	45.3
Str (Expr.Numérica)	Converte números em *string*	Str (45.3)	"45.3"
Len (String)	Tamanho de um *string*	Len ("TESTE")	5
SubStr (*string*, N1, N2)	Parte do *string*, começando em N1, com tamanho N2	SubStr ("TESTE",3, 2)	"ST"
Chr (Expr. Numérica)	Caractere referente ao número	Chr (67)	"C"
Asc (Caractere)	Código ASCII do caractere	Asc ("F")	70

A letra "A" tem o código 65 na tabela ASCII. "B" tem o código 66, e assim por diante. Assim, podemos iniciar uma variável com 65 e, usando a função CHR, gerar a letra "A", mudar esta variável para 66 e gerar "B", até o valor 90, equivalente à letra "Z".

```
ALFABETO
Caractere Letra
GerarAlfa ( )
Objetivo: Imprimir alfabeto maiúsculo
Inteiro L
Para  L de 65 até  90  repita
      Letra ← Chr ( L )
      Imprimir (Letra)
Fim-Para
```

Pode-se, também, iniciar uma variável com o valor "A", e calcular as próximas somando 1 no valor da letra na tabela ASCII e converter esse valor novamente em caractere.

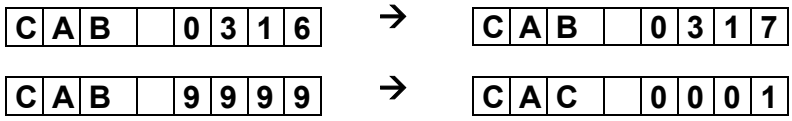
```
ALFABETO
Caractere Letra
GerarAlfa ( )
Objetivo: Imprimir alfabeto maiúsculo
Letra ← "A"
Enquanto Letra ≤ "Z" faça
      Imprimir (Letra)
      Letra ← Chr(Asc(Letra) + 1)          // Próxima letra
Fim-Enquanto
```

5.13. Próxima Placa

Dada uma placa de automóvel, com 3 letras, um espaço e 4 dígitos, armazenada em uma variável do tipo string, obter a placa que sucede a placa dada.

Uma solução

Exemplo:



Não se pode somar 1 direto num dado do tipo *string*. A idéia é separar a parte numérica da placa, somar 1 e juntá-la novamente com as letras. Quando a parte numérica ficar > 9999, voltamos para 0001 e avança-se a terceira letra. Quando a terceira ou a segunda letra ficar > "Z", volta-se essa posição para "A" e avança-se a letra anterior.

```
AUTOMÓVEL
String Placa, Prox                    // Placa atual e placa seguinte
Proxima_Placa( )
Objetivo: Calcular a próxima placa de automóvel
String Letra1, Letra2, Letra3, Num
Inteiro Valor
Letra1 ← SubStr (Placa, 1 , 1)
Letra2 ← SubStr (Placa, 2 , 1)
Letra3 ← SubStr (Placa, 3 , 1)
Num  ← SubStr (Placa, 5 , 4)       // Parte numérica da placa (tipo string)
Valor ← Val (Num) + 1               // Valor numérico de Num, + 1
Se   Valor > 9999 então
     Valor ← 1
     Letra3 ← Chr (Asc (Letra3) + 1)          // Próxima letra
     Se   Letra3 > "Z" então
          Letra3 ← "A"
          Letra2 ← Chr (Asc (Letra2) + 1)
          Se   Letra2 > "Z" então
               Letra2 ← "A"
               Letra1 ← Chr (Asc (Letra1) + 1)
               Se   Letra1 > "Z" então
                    Letra1 ← "A"          // Voltar ao início: "ZZZ" para "AAA"
               Fim-se
          Fim-se
     Fim-se
Fim-se
Num ← Str (Valor)
Prox ← Letra1 + Letra2 + Letra3 + SubStr ("000", 1, 4 – len ( Num)) + Num
```

5.14. Engrenagem

Obter números de 0 a 999, separados em Centenas, Dezenas e Unidades.

Uma solução

As Centenas, Dezenas e Unidades se comportam como engrenagens, ou seja, a cada 10 giros da unidade a dezena gira uma vez (vai um) e a cada 10 giros da dezena a centena gira uma vez. Nota-se que a engrenagem da unidade trabalha mais que a dezena, que trabalha mais que a da centena.

A estrutura que implementa esse mecanismo é um *loop* dentro do outro, tendo a unidade como o *loop* mais interno. A quantidade de voltas é 10 X 10 X 10 = 1000.

Pode-se usar uma variável para cada posição e variá-la de 0 a 9.

```
ENGRENAGEM
Inteiro C, D, U
GerarNúmeros ( )
Objetivo : Obter números de 0 a 999 no formato Centena, Dezena e Unidade
Para  C de 0 até 9
      Para  D de 0 até 9
            Para  U de 0 até 9
                  Imprima (C, D, U)
            Fim-Para
      Fim-Para
Fim-Para
```

5.15. Prestações

Dada a data inicial de uma prestação e um nº inteiro N, representando uma quantidade de meses, calcular os próximos N – 1 vencimentos mensais, sempre no mesmo dia.

Uma solução

Ex.: Data inicial = 16/11/03 , N = 4

As próximas datas seriam: 16/12/03, 16/01/04, 16/02/04

Uma estratégia seria:

- Separar dia, mês e ano da data inicial
- Gerar a próxima data, repetindo o dia e somamos 1 no mês
- Se o resultado for 13, mudar o mês para 1 e somar 1 no ano

Existe um problema para datas com dia > 28, pois fevereiro pode não ter 29 dias nos anos não bissextos, e muito menos 30 ou 31 dias. Também, nem todos os meses têm 31 dias. Podemos testar as exceções, que são os meses de Abril, Junho, Setembro e Novembro, que têm 30 dias, e o mês de fevereiro que, se for ano bissexto tem 29 dias e se não, tem 28.

Um ano é bissexto for múltiplo de 4 e não for múltiplo de 100, exceto quando for múltiplo de 400 (1600 e 2000 foram bissextos; 1900 não foi).

Quando se lida com o tipo "data", pode-se contar com algumas funções de manipulação de datas, tais como:

Dia (data) ➔ Retorna o dia de uma data, no tipo inteiro.
Mês (data) ➔ Retorna o mês
Ano (data) ➔ Retorna o ano, com 4 dígitos
Data (dia, mês, ano) ➔ Converte os 3 números para tipo data

```
CONTRATO
Data DataInicial          // Data da 1ª prestação
Inteiro N                 // Quantidade de prestações
Data DataPrest            // Datas de vencimento das próximas prestações
CalcularDatas ( )
Inteiro D1, D2, M, A
D1 ← Dia  ( DataInicial )
M  ← Mês ( DataInicial )
A  ← Ano ( DataInicial )
Para  K de 1 até N – 1                    // Calcular as próximas N - 1 datas
      M ← M + 1
      Se  M  = 13 então
          M ← 1
          A ← A + 1
      Fim-se
      Se  (M = 4  ou  M = 6  ou  M = 9  ou  M = 11)  e  D1 = 31 então
          D2 ← 30
      Senão
          Se  M = 2 então
              Se  D1 > 28 então
                  Se  A mod 4 = 0 e (A mod 100 ≠ 0 ou A mod 400 = 0)
                      D2 ← 29             // ano bissexto
                  Senão
                      D2 ← 28
                  Fim-se
              Fim-se
          Fim-se
      Senão
          D2 ← D1
      Fim-se
      DataPrest ← Data ( D2, M, A )
      Imprima (DataPrest)
Fim-Para
```

Um recurso interessante para se calcular o último dia de cada mês ou a quantidade de dias de um mês, é subtrair 1 do 1º dia do mês seguinte. O tipo DATA permite esta operação.Exemplo: Para se saber quantos dias tem o mês de fevereiro de 2000, calcula-se Dia (Data (01, 03, 2000) – 1), sem a preocupação com o fato de este ano ser bissexto, e a resposta será 29.

5.16. Exercícios propostos

a) Loops

Quantas vezes serão executados os blocos de instruções sob o controle dos seguintes comandos de repetição?

- Para I de 2 até 20
- Para K de 8 até 2 passo -1
- Para J de 1 até 10 passo 2
- Para I de 10 até 2
- Para K de 3 até N / 2
- Para J de 10 até 3 passo –3
- Para K de N até 2 passo –2

b) Simulações

Mostrar, na forma de uma tabela, a evolução das variáveis usadas nos algoritmos deste capítulo.

c) Seqüências

Dado um número N, inteiro e positivo, obter os N primeiros termos das seguintes seqüências:

- 1 2 4 8 16 ...
- 1 4 7 10 13 ...
- $\frac{1}{2}$ $\frac{3}{4}$ $\frac{5}{6}$ $\frac{7}{8}$
- 1 4 9 16 ...
- 2 4 7 11 ...
- 1 –1 1 –1 1 ...

Dado um número N, inteiro e positivo, obter o N-ésimo termo das seguintes seqüências:

- 1 –1 3 –5 11 –21 43 ...
- 1 3 7 13 21 ...
- 1 2 –1 5 –5 10 –11 ...
- $Y_{K+1} = 2. Y_K + 1, Y_1 = 1, K = 1, 2, 3, ...$
- $Y_{K+1} = Y_K + K, Y_1 = 1, K = 1, 2, 3, ...$
- 2 0 3 –1 4 –2 5 ...
- 2 4 1 5 0 6 –1 ...

Dado um número N, inteiro e positivo, obter a soma dos N primeiros termo das seguintes seqüências:

- 1 2 6 24 ...
- 1 3 7 15 31 ...
- 2 5 11 23 47 95
- 2 3 6 11 18 27
- $Y_K = \frac{(K + 1) K}{2}, \quad K = 1, 2, 3, ...$

d) Soma de dígitos

Obter todos os números inteiros, maiores que 100 e menores que 1000, cuja soma de dígitos seja maior que 10.

Exemplo: Os primeiros números obtidos seriam: {119, 128, 129, 137, 138, 139, ...}

e) Quantidade de dígitos

Desenvolver um algoritmo iterativo para obter quantos dígitos tem um número inteiro dado.
Exemplo: Dado 31762, obter 5.

f) MDC por subtrações sucessivas

Dados 2 nºs, pertencentes a N*, calcular o MDC pelo método das subtrações sucessivas do maior.

Exemplo: Dados A = 20 e B = 12

A	B
20	12
8	4
4	

A > B → A – B = 20 – 12 = 8
B > A → B – A = 12 – 8 = 4
A > B → A – B = 8 – 4 = 4
A = B → MDC = 4

Ou seja, quando A é maior → A – B; quando B é maior → B – A

g) Simulação

Verificar se o algoritmo abaixo cumpre seu objetivo, para N ≥ 1

```
SEQÜÊNCIA
Inteiro N, Termo
GerarSeqüência ( )
Objetivo: Obter os N primeiros termos da seqüência  0  2  6  12  20 ...
Inteiro A, B, C, K
A ← 0
B ← 2
Imprima (A, B)
Para  K de 3 até N repita
      A ← A + 2
      B ← B * 2
      Termo ← A + B
      Imprima (Termo)
Fim-Para
```

h) Séries convergentes

Dado um número N, inteiro e positivo, calcular o valor das seguintes séries, com N termos:

$\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \ldots$ (→ 1) (no limite, o valor dessa série converge para 1)

$2 * \frac{2}{3} * \frac{4}{3} * \frac{4}{5} * \frac{6}{5} * \frac{6}{7} * \frac{8}{7} \ldots$ ($\rightarrow \frac{\Pi}{2}$)

$X - \frac{X^2}{2} + \frac{X^3}{3} - \frac{X^4}{4} + \ldots$ (→ ln (1 + X), X > 1)

$$X - \frac{X^3}{3!} + \frac{X^5}{5!} - \frac{X^7}{7!} + \ldots \quad (\rightarrow \text{sen (X), X em radianos})$$

$$1 - \frac{X^2}{2!} + \frac{X^4}{4!} - \frac{X^6}{6!} + \ldots \quad (\rightarrow \cos (X), X \text{ em radianos})$$

$$1 + X - \frac{X^2}{2!} + \frac{X^4}{4!} - \ldots \quad (\rightarrow e^X)$$

i) Primo

Dado um número N, inteiro e positivo, verificar se N é primo, ou seja, se N tiver apenas dois divisores: ele mesmo e a unidade.

Obs: Para verificar se um número N é primo pode-se limitar o teste de divisibilidade até o valor de $\sqrt{N}$

j) MMC

Dados 2 números, inteiros e positivos, calcular o Mínimo Múltiplo Comum (MMC) entre eles.

Exemplo: Dados 6 e 10, obter 30.

k) Múltiplos

Dados N, A e B, números inteiros e positivos, obter os N primeiros múltiplos de A, de B ou de ambos, em ordem crescente, intercalados e sem repetições.

Exemplo: Dados N = 10, A = 3 e B = 4 , mostrar: 3, 4, 6, 8, 9, 12, 15, 16, 18, 20
Dados N = 6, A = 3 e B = 13, mostrar: 3, 6, 9, 12, 13, 15

l) Camundongos [6]

Uma determinada população de camundongos do tipo A triplica a cada ano. A população de camundongos do tipo B duplica a cada ano. Sabendo-se que a população de camundongos do tipo A é atualmente menor que a do tipo B, escrever um algoritmo iterativo para obter em quantos anos, mantendo-se o mesmo ritmo de crescimento, os camundongos do tipo A serão mais numerosos que os do tipo B.

m) Divisores

Dado um número N, inteiro e positivo, obter todos os divisores pares de N.

Exemplo: Dado o número 36, obter {2, 4, 6, 12, 18, 36}

[6] Baseado no problema das populações, apresentado em Salvetti & Barbosa (1998, p. 32)

n) CGC / CPF

Calcular o dígito de controle do CGC e CPF, de acordo com as seguintes instruções:

1. Cadastro Geral de Contribuintes – CGC

1.1. Formato Geral: $X_1X_2X_3X_4X_5X_6X_7V_1 / O_1O_2O_3O_4 – C_1C_2$

1.2. Composição:

a) Os 7 primeiros algarismos constituem o número básico de inscrição da empresa no CGC (X_1 a X_7);
b) O oitavo algarismo (V_1), constitui o dígito verificador do número formado pelos 7 primeiros algarismos, calculado pelo módulo 10;
c) Os 4 algarismos após a barra (O_1 a O_4) constituem o número de ordem do estabelecimento na empresa cadastrada;
d) O primeiro algarismo após o número de ordem (C_1) representa o dígito verificador para o número formado pelos 12 algarismos anteriores, calculado pelo módulo 11;
e) O último algarismo (C_2) representa o dígito verificador para o número formado pelos 13 algarismos anteriores, calculado também pelo módulo 11.

1.3. Cálculo de V1

Cada algarismo do número básico, a partir da direita, é multiplicado por 2, 1, 2, 1, 2, 1, e 2, sucessivamente; somam-se, a seguir, os algarismos resultantes de cada produto. O valor de V_1 será a diferença entre a soma assim obtida e a dezena imediatamente superior.
Exemplo: seja o número básico 1 2 3 4 5 6 7. O cálculo será efetuado da seguinte forma:

1	2	3	4	5	6	7
X	X	X	X	X	X	X
2	1	2	1	2	1	2
=	=	=	=	=	=	=
2	2	6	4	10	6	14

A soma dos algarismos será: 2 + 2 + 6 + 4 + 1 + 0 + 6 + 1 + 4 = 26
O valor de V1 será dado pela diferença da soma 26 para a dezena imediatamente superior, ou seja, 30. Portanto: V1 = 30 –26 = 4
Como resultado, o número básico de inscrição da empresa no CGC será 12345674.

1.4. Cálculo de C1

Para o cálculo de C_1 é considerado o CGC do estabelecimento, compostos de 12 algarismos e formado pelo número básico seguido do número de ordem. Cada algarismo, da direita para a esquerda, é multiplicado sucessivamente por 2, 3, 4, até 9,

quando o multiplicador volta a 2, reiniciando a série. Os produtos resultantes são então somados.

Para obtenção do valor de C_1 divide-se a soma resultante por 11 e toma-se o seu resto. Quando o resto for igual a 0 (zero), C_1 será também igual a 0 (zero). Nos demais casos o valor de C_1 será igual ao complemento do resto para 11, sendo que, quando o complemento for igual a 10 toma-se o valor 0 (zero).

Exemplo: Seja o CGC do estabelecimento 12345674 / 1234

1	2	3	4	5	6	7	4	/	1	2	3	4
X	X	X	X	X	X	X	X		X	X	X	X
5	4	3	2	9	8	7	6		5	4	3	2
=	=	=	=	=	=	=	=		=	=	=	=
5	8	9	8	45	48	49	24		5	8	9	8

A soma dos produtos será: 5+8+9+8+45+48+49+24+5+8+9+8 = 226
A divisão de 226 por 11 nos dá um quociente 20 com resto 6.

Logo C1 = 11 – 6 = 5

1.5. Cálculo de C2

O valor de C2 é obtido tendo por base os 13 algarismos anteriores, correspondentes ao CGC do estabelecimento e do primeiro dos dígitos de controle (C1).
O procedimento de cálculo é idêntico ao de C1, iniciando a partir da direita, a multiplicação de cada algarismo por 2, 3, 4, até 9, retornando o multiplicador a 2, e somando depois os produtos obtidos.

Exemplo: Seja o CGC do estabelecimento 12345674 / 1234

1	2	3	4	5	6	7	4	/	1	2	3	4	-	5
X	X	X	X	X	X	X	X		X	X	X	X		X
6	5	4	3	2	9	8	7		6	5	4	3		2
=	=	=	=	=	=	=	=		=	=	=	=		=
6	10	12	12	10	54	56	28		6	10	12	12		10

A soma será: 6+10+12+12+10+54+56+28+6+10+12+12+10 = 238.
A divisão de 238 por 11 nos dá um quociente 21 com resto 7.
Logo $C_1 = 11 – 7 = 4$.
Assim, para o exemplo dado, o CGC completo será: 12345674 / 1234 - 54

2. Cadastro de Pessoas Físicas – CPF

2.1. Formato Geral: $X_1X_2X_3X_4X_5X_6X_7X_8R - C_1C_2$

2.2. Composição:

a) Os 8 primeiros algarismos constituem o número básico de inscrição da pessoa física no CPF (X_1 a X_8);
b) O nono algarismo – R – é o indicativo da Região Fiscal onde foi efetuada a inscrição no cadastro;
c) O primeiro algarismo após o indicativo da Região Fiscal – C_1 – representa o dígito verificador para o número formado pelos 9 algarismos anteriores, calculado pelo módulo 11;
d) O último algarismo – C_2 - representa o dígito verificador para o número formado pelos 10 algarismos anteriores, calculado também pelo módulo 11;

2.3. **Cálculo de C_1**

Cada um dos 9 algarismos do CPF, a partir da direita, é multiplicado sucessivamente por 2, 3, 4, 5, 6, 7, 8, 9 e 10 e os produtos resultantes são somados. A soma obtida é dividida por 11 e C_1 será igual ao complemento para 11 do resto da divisão; quando o complemento for maior ou igual a 10, toma-se o valor 0 (zero).

Exemplo: seja o CPF 123456785. O cálculo de C_1 será efetuado da seguinte forma:

1	2	3	4	5	6	7	8	5
X	X	X	X	X	X	X	X	X
10	9	8	7	6	5	4	3	2
=	=	=	=	=	=	=	=	=
10	18	24	28	30	30	28	24	10

A soma dos produtos será: 10+18+24+28+30+30+28+24+10 = 202
A divisão de 202 por 11 nos fornece um quociente 18 com resto 4.
Logo $C_1 = 11 - 4 = 7$

2.4. Cálculo de C_2

O cálculo de C_2 toma por base o número de 10 algarismos formado pelo CPF seguido do primeiro dígito de controle.

Cada um dos algarismos a partir da direita, é multiplicado sucessivamente por 2, 3, 4, 5,6, 7, 8, 9, 10 e 11 e os produtos resultantes são somados.
C_2 é então obtido de maneira análoga a C_1.

Exemplo: Seja o CPF 123456785 com o primeiro dígito verificador igual a 7. C2 será calculado para o número 12345678 – 7 como segue:

1	2	3	4	5	6	7	8	5	-	7
X	X	X	X	X	X	X	X	X		X
11	10	9	8	7	6	5	4	3		2
=	=	=	=	=	=	=	=	=		=
11	20	27	32	35	36	35	32	15		14

A soma dos produtos será: 11+20+27+32+35+36+35+32+15+14 = 257
A divisão de 257 por 11 nos dá um quociente 23 com resto 4. Logo $C_1 = 11 - 4 = 7$
Assim, para o exemplo dado, o CPF completo será : 123456785 – 77

o) Imposto

O pagamento de certo imposto deve ser feito na quarta-feira da semana seguinte ao fato gerador. Dada a data do fator gerador, calcular a data do pagamento.

Considerar a existência de uma função chamada DayOfWeek, que recebe uma data no formato DD/MM/AA, e retorna o nome do dia da semana (‘SUNDAY’, ‘MONDAY’, ‘TUESDAY’, ‘WEDNESDAY’, ‘THURSDAY’, ‘FRIDAY’, ‘SATURDAY’).

p) Repetições

Dado um número N, inteiro e positivo, obter os N primeiros termos da seguinte seqüência: 1 2 2 3 3 3 4 4 4 4 5 ...

q) Fatoração

Dado um número inteiro e positivo, obter sua decomposição em fatores primos.

Ex. Dado: 84 , obter

2
2
3
7

Dado: 75, obter

3
5
5

Alterar o programa anterior, para mostrar o fator primo e seu respectivo expoente:

Ex. Dado: 84 , obter

2	2
3	1
7	1

Dado: 75, obter

3	1
5	2

r) Calculadora financeira

Fazer um projeto para simular as funções de uma calculadora financeira.

As variáveis envolvidas nos cálculos financeiros são:

PMT = prestação (pagamento periódico)
PV = valor atual (valor presente) da anuidade
n = nº de intervalos de pagamento
FV = montante da anuidade (valor futuro)
i = taxa de juros do período, expressa em decimais (5% = 0,05)

Se uma das variáveis não estiver preenchida o programa deve calcular seu valor. Deve, também, refazer o cálculo para a última alteração de uma das variáveis. Deve prever também uma função para limpar tudo.

As fórmulas a seguir são para pagamento no final do período de amortização ou capitalização.

$$FV = PMT \left[\frac{(1+i)^{n-1}}{i} \right]$$

$$PV = PMT \left[\frac{1-(1+i)^{-n}}{i} \right]$$

$$PMT = i \cdot PV \left[\frac{(1+i)^{n}}{(1+i)^{n-1}} \right]$$

$$n = \frac{\log\left(1 - \frac{i\,PV}{PMT}\right)}{\log\left(\frac{1}{1+i}\right)}$$

O cálculo da taxa (i) é feito por interpolação, a partir de uma taxa inicial sugerida e um intervalo para pesquisa da taxa aproximada.

Exemplo: Seja n = 12, PV = 1000 e PMT = 106,55

Usando a fórmula para cálculo de PV temos que encontrar duas taxas, uma acima e outra abaixo da taxa procurada, que aproxime o valor de PV / PMT. Começamos por uma taxa que achamos ser próxima do valor procurado.

$$\frac{PV}{PMT} = \left[\frac{1-(1+i)^{-n}}{i} \right]$$

No exemplo temos: $\frac{PV}{PMT} = \frac{1000}{106,55} = 9,38527$

Se usarmos como taxa inicial 3 %, a relação PV / PMT é:

$$\left[\frac{1 - (1 + 0{,}03)^{-12}}{0{,}03} \right] = 9{,}954$$ (valor acima de 9,38527 → taxa abaixo da real)

Para uma taxa de 5 %, a relação PV / PMT é:

$$\left[\frac{1 - (1 + 0{,}05)^{-12}}{0{,}05} \right] = 8{,}86325$$ (taxa acima da real)

Interpolando os valores temos:

$$\begin{bmatrix} 0{,}03 \\ i \\ 0{,}05 \end{bmatrix} \quad \begin{bmatrix} 9{,}954 \\ 9{,}38527 \\ 8{,}86325 \end{bmatrix}$$

$$\frac{i - 0{,}03}{0{,}05 - 0{,}03} = \frac{9{,}38527 - 9{,}954}{8{,}86325 - 9{,}954} \qquad i = 4{,}04\ \%$$

Se a interpolação fosse feita entre 3,5 % e 4,5 %, a taxa seria 4,01%.
Se as taxas fossem 3,9% e 4,1%, a taxa encontrada seria 4%.

s) Tabela de amortização

Dado uma quantidade de períodos, uma dívida original e uma taxa de juros, preencher a tabela de amortização dessa dívida.

Período	Saldo devedor no inicio do período (SD)	Juro devido no fim do período (J)	Pagamento (PMT)	Capital amortizado (CA)
1	Dívida original	i% x SD	Prestação	PMT – J
2	SD – CA			

t) Depreciação

Calcular a depreciação anual de um bem, dado o valor de compra e sua vida útil, para os seguintes métodos: Linear, Progressiva, Soma dos dígitos.

Ex.: Calcular as depreciações anuais de um bem comprado por $10.000, com vida útil de 10 anos.

Linear

$$\text{Depreciação anual} = \frac{\text{Valor original}}{\text{Vida útil}} = \frac{10.000}{10} = 1.000 \text{ por ano}$$

O valor da depreciação é constante

Progressiva

O valor da depreciação de cada período é obtido aplicando-se um fator sobre o valor do bem no início do período, já descontadas as depreciações dos períodos anteriores. O fator pode ser, por exemplo, 2 / Vida útil = 2 / 10 = 0.2

Ano	Valor inicial do período	Depreciação
1°	10.000	10.000 x 0.2 = 2.000
2°	8.000	8.000 x 0.2 = 1.600
3°	6.400	6.400 x 0.2 = 1.280

O valor final (início do 11o ano), é denominado valor residual ou depreciado.

Soma dos dígitos dos anos

Aplica-se, sobre o valor original, um fator diferente para cada ano. Esse fator é uma fração cujo numerador é o nº do ano de vida, em ordem decrescente (10, 9, 8, ..., 1). O denominador é a soma dos dígitos dos anos (1+2+3+ ...+ 10).

Ano	Depreciação
1°	10.000 x 10 / 55 = 1.818,18
2°	10.000 x 9 / 55 = 1.636,36
3°	10.000 x 8 / 55 = 1.454,55

u) Binário para decimal

Dado um string de bits (seqüência de dígitos 0 ou 1), representando um número na base 2, converte-lo para a base 10.

Exemplo: Dado o *string* "10110", obter 22.

v) Dominó

Criar uma lista com os valores das 28 peças do jogo de dominó:
0 – 0, 0 – 1, 0 – 2, 0 – 3, 0 – 4, 0 – 5, 0 – 6, 1 – 1, 1 – 2, ..., 5 – 6, 6 – 6.

w) Placas de automóvel

Obter uma lista de placas de automóvel, de acordo com a solicitação, por faixa, ou por quantidade.

- Por faixa: mostrar placas entre 2 placas informadas.
  Ex.: Mostrar as placas entre BAC 0001 e BCZ 9999.
- Por quantidade: mostrar uma quantidade de placas a partir de uma placa inicial.
  Ex.: Mostrar 1000 placas a partir de CAA 0358.

Este tipo de programa pode ser utilizado para emissão de etiquetas de placas em despachantes, códigos de contas correntes, números de matrícula etc.

x) Números amigos entre si

(Malba Tahan, 1966) Dois números inteiros e positivos são *amigos entre si* quando a soma dos divisores próprios do primeiro número for igual ao segundo, e a soma dos divisores próprios do segundo for igual ao primeiro. Dados dois números, verificar se são amigos entre si.

Ex.: 220 tem como divisores próprios: 1, 2, 4, 5 ,10, 11, 20, 22, 44, 55 e 110, cuja soma é 284
284 tem como divisores próprios: 1, 2, 4, 71, 142, cuja soma é 220
Portanto, 220 e 284 são amigos entre si.

Verificar o programa para os números (18416 e 17296) e (9.434.056 e 9.363.584).

Alguns números são amigos de si próprio: 6 = 1 + 2 + 3; 28 = 1 + 2 + 4 + 7 + 14

y) Planilha eletrônica

Criar uma lista com os códigos de uma faixa de colunas de uma planilha eletrônica, de "AA" até "IV", ou seja: {AA, AB, AC, ..., AZ, BA, BB, BC, ..., IV)

z) Grãos de trigo

(Malba Tahan, 1966) Conta-se que um poderoso rei da Índia resolveu dar uma rica recompensa ao inventor do jogo de xadrez, tal foi seu encanto pelo jogo. O inventor pediu que o pagamento fosse feito em grãos de trigo, um grão como prêmio pela 1ª casa, dois pela 2ª , 4 pela 3ª, e sempre assim, dobrando a cada casa, até a 64ª.

Escrever um algoritmo para calcular a quantidade de grãos, considerando que o resultado, que possui 20 dígitos, não pode ser armazenado em uma única variável, por falta de suporte para esta quantidade de dígitos (precisão).

REFERÊNCIAS E BIBLIOGRAFIA

BAASE, Sara, GELDER, Allen Van. *Computer Algorithms: Introduction to Design and Analysis*. 3. ed., Addison-Wesley, 2000.

BARBOSA, Lisbete Madsen. *Ensino de algoritmos em cursos de computação*. Dissertação de mestrado. PUC – São Paulo. 1999.

BOYER, Carl. *História da Matemática*. São Paulo: Edgar Blücher e Ed. da USP, 1974.

FARRER, H. et alli. *Algoritmos estruturados*. Rio de Janeiro: Ed. Guanabara, 1989.

GARDNER, Martin. *Ah, Descobri*!. Lisboa: Gadiva, 1990.

KNUTH, Donald E. *The art of computer programming*. Massachusetts: Addison-Wesley. Volume 1 – Fundamental Algorithms; Volume 3 – Sorting and Searching. 1973.

LAWLOR, Robert. *Geometria Sagrada*. Madri: Edições del Prado, 1996.

MALBA TAHAN. *Diabruras da Matemática*. 2.ed. São Paulo: Saraiva, 1966.

MEC/SESu/CEEInf – Diretrizes Curriculares de Cursos da Área de Computação e Informática. Disponível em http://www.mec.gov.br/sesu. Acesso em 30/01/2006.

OLIVEIRA, Ariovaldo Dias de. Notas de aula da disciplina de Algoritmos. Fundação Santo André, 2005.

SALVETTI, Dirceu Douglas, BARBOSA, Lisbete Madsen. *Algoritmos*. São Paulo: Makron Books, 1998.

SEBESTA, Robert W. Conceitos de linguagens de programação. 4. ed. Porto Alegre: Bookman, 2000.

SETUBAL, João Carlos. Uma proposta de Plano Pedagógico para a matéria de Computação e Algoritmos, in. *Qualidade de cursos de graduação da área de Computação e Informática*. SBC – WEI 2000 – Curitiba: Ed. Universitária Champagnat, 2000

TREMBLAY, J., BUNT, R. *Introdução à ciência dos computadores* - Uma abordagem algorítmica. São Paulo: McGraw-Hill, 1983.

VELOSO, Paulo, et. alli . *Estruturas de dados*. Rio de Janeiro: Campus. 1983.

WIRTH, Niklaus. *Programação sistemática em Pascal*. 6ª ed. Rio de Janeiro: Campus,1987.