

Conteúdo

Introdução.....	1
Necessidade do uso da lógica	1
Como usar a apostila.....	1
Conceitos Básicos de Computação	3
O que é um computador?	3
Dados	5
Primeiros sistemas operacionais	6
Conceitos de Lógica	8
Conjuntos.....	8
Expressões.....	8
Interruptor	9
Lógica Formal	10
Proposições.....	10
Princípios fundamentais da lógica	11
Exercícios Complementares.....	16
Programa.....	18
Linguagem de Alto Nível e Baixo Nível.....	18
Programação	19
Linguagens de máquina, Linguagens Assembly	19
Linear	20
Estruturada	20
Modular.....	21
Orientação a Objeto.....	21
Tipos de tradutores.....	23
Antes de começar a programar.	24
Algoritmos.....	25
Técnicas de Programação	30

Fluxogramas.....	30
Pseudocódigo.....	31
Atribuição.....	32
Tipos de informação	32
Operadores	33
Operadores de Aritméticos.....	33
Operadores Relacionais	34
Operador de Atribuição	34
Operadores Lógicos.....	35
Uso do fluxograma para entrada, leitura e atribuição.....	35
Exercícios de Aprendizagem:	36
Exercícios extra	46
Estruturas de Controle – Condicionais.....	48
Desvios Condicionais Simples	48
Condicionais Compostos.....	48
Condicionais Encadeados.....	50
Operadores lógicos – uma revisão.....	52
Condicionais Seletivos (Escolha)	53
Exercícios de Aprendizagem:	54
Estruturas de Controle – Repetições	66
“Enquanto” – Teste lógico no início do looping.....	66
“Repita até” – Teste lógico no final do looping	67
“Para” – Teste lógico controlado por variável – contador.....	68
Repetição Controlada por sentinela	69
Exercícios de Aprendizagem:	70
Estruturas de Dados Simples –Vetores.....	81
Exercícios de aprendizagem.....	82
Estruturas de Dados Simples –Matrizes	91
Programação Modular – Sub-rotinas e Procedimentos.....	93

Estruturas em pseudocódigo	94
Mini manual do Visualg.....	96
Funções do Visualg.....	100
Funções Numéricas, algébricas e trigonométricas	100
Funções para manipulação de cadeias de caracteres (strings).....	101

Introdução

Muitas pessoas gostam de falar ou julgar que possuem e sabe usar o raciocínio lógico, porém, quando questionadas direta ou indiretamente, perde esta linha de raciocínio, pois este depende de inúmeros fatores para completá-lo, tais como: calma, conhecimento, vivência, versatilidade, experiência, criatividade, ponderação, responsabilidade, entre outros.

Para usar a lógica, é necessário ter domínio sobre o pensamento, bem como saber pensar, ou seja, possuir a “Arte de Pensar”. Alguns definem o raciocínio lógico como um conjunto de estudos que visa determinar os processos intelectuais que são as condições gerais do conhecimento verdadeiro. Outros preferem dizer que é a sequência coerente, regular e necessária de acontecimentos, de coisas ou fatos, ou até mesmo, que é a maneira do raciocínio particular que cabe a um indivíduo ou a um grupo.

Para concluir todas essas definições, pode-se dizer que lógica é a ciência que estuda as leis e critérios de validade que regem o pensamento e a demonstração, ou seja, ciência dos princípios formais do raciocínio.

Necessidade do uso da lógica

Usar a lógica é um fator a ser considerado por todos, principalmente pelos profissionais da área da tecnologia de Informação (programadores, analistas de sistemas e suporte), pois seu dia a dia dentro das organizações é solucionar problemas e atingir os objetivos apresentados por seus usuários com eficiência e eficácia, utilizando recursos computacionais e/ou automatizados mecanicamente. Saber lidar com problemas de ordem administrativa, de controle, de planejamento e de estratégia requer atenção e bom desempenho de conhecimento de nosso raciocínio. Porém é necessário considerar que ninguém ensina ninguém a pensar, pois todas as pessoas normais possuem esse “dom”.

Como usar a apostila

A apostila é um apoio para nossos encontros, de forma que, é indispensável cumprir os exercícios propostos durante o desenvolvimento de nossas atividades, fazer anotações, observações, anotar lembretes e dicas que são dados nos treinamentos.

O importante é que esta apostila é apenas parte de nosso trabalho e não trará todas as respostas possíveis, assim como não o tornará um profissional completo. Para isso é necessário um trabalho em conjunto. Seu esforço pessoal em ler, buscar o conhecimento, seja em textos, na internet ou em livros, seja questionando, ou ainda desafiando a si mesmo na procura de solucionar problemas, é parte importantíssima. Os treinamentos complementam o conteúdo aqui presente, buscam orientar de forma a possibilitar a resolução de futuras dificuldades que possam e certamente aparecerão.

Dessa maneira, a leitura simples e inocente da apostila pouco o ajudará. É necessária uma leitura interessada e inquisitiva, a atenção no conteúdo exposto para complementação do material aqui presente, além da feitura de exercícios e busca em outras fontes de informação apresentadas no decorrer do treinamento.

Como apoio, você pode ainda, entrar em contato com o instrutor, através do email: biasonlucky@hotmail.com.

Conceitos Básicos de Computação

Antes de entender a lógica de programação precisamos ter em mente alguns conceitos fundamentais. Não se pode imaginar alguém que afirme que sabe dirigir um carro e ao mesmo tempo afirme não saber o que vem a ser um carro.

A programação de computadores, de uma maneira bem fácil de entender, e como veremos com mais detalhes a frente, nada mais é do que passar uma série de comandos ao computador para que ele possa executar tarefas. Como, entretanto, imaginar que pretendemos dar comandos as máquinas sem ter em mente o que são essas máquinas e como funcionam.

Como se trata de um treinamento que não exige conhecimentos prévios, embora seja de bom tom que o interessado já tenha em mente esses conceitos, passaremos, de forma breve, apresentando os conceitos fundamentais sem muitos detalhes, apenas para que possamos, de maneira uniforme, dar prosseguimento ao treinamento de lógica.

O que é um computador?

Um computador é um dispositivo capaz de fazer cálculos e tomar decisões lógicas a velocidades milhões (até bilhões) de vezes mais rápido que os seres humanos. Por exemplo, muitos computadores pessoais podem realizar um bilhão de somas por segundo. Uma pessoa operando uma calculadora de mesa pode levar anos e anos para fazer cálculos e ainda não concluir a mesma quantidade de cálculos que um poderoso computador pessoal pode realizar em um segundo. (Questões que devem ser levadas em conta: Como saber se a pessoa somou os números corretamente? Como saber se o computador somou os números corretamente?) Os mais rápidos supercomputadores podem realizar centenas de bilhões de somas por segundo -, hoje, computadores de um trilhão de instruções por segundo já estão funcionando em laboratórios de pesquisa.

Os computadores processam dados sob o controle de conjuntos de instruções denominados programas de computador. Esses programas orientam o computador por meio de conjuntos ordenados de ações especificadas pelos programadores de computador .

Os vários dispositivos que compõem um computador são chamados de hardware (como: teclado, tela, mouse, discos, memória, unidades de DVD, CD-ROM e de processamento); já os programas que executam em um computador

são chamados de software. Os custos com hardware caíram significativamente nos últimos anos, a ponto de transformar os computadores pessoais em bem de consumo popular; mas, infelizmente, os custos com o desenvolvimento de software aumentam constantemente ao mesmo tempo em que os programadores desenvolvem aplicativos cada vez mais poderosos e complexos. Independentemente das diferenças na aparência física, em geral, todos os computadores podem estar divididos em seis unidades lógicas ou seções:

Unidade de Entrada: Esta é a seção 'receptora' do computador, obtém informações (dados e programas de computador) de dispositivos de entrada e coloca essas informações à disposição de outras unidades para serem processadas. A maioria das informações é inserida em computadores por meio de dispositivos de entrada, como teclado e mouse. As informações também podem ser inseridas de várias maneiras, inclusive falando com seu computador, digitalizando imagens e fazendo com que ele receba informações de uma rede, como a Internet.

Unidade de saída: Esta é a seção de 'envio' do computador. Coleta as informações que o computador processou, coloca-as em vários dispositivos de saída, tornando-as disponíveis para serem utilizadas fora do computador. Muitas das informações enviadas para a saída de computadores são exibidas em telas, impressas em papel ou utilizadas para controlar outros dispositivos. Os computadores também podem gerar saída de suas informações para redes, como a Internet.

Unidade de memória: Esta é a seção de 'armazenamento' relativamente de baixa capacidade e de acesso rápido do computador. Retém informações que foram inseridas pela unidade de entrada, para se tornarem imediatamente disponíveis para processamento quando necessário. A unidade de memória também retém informações processadas até que elas possam ser colocadas em dispositivos de saída pela unidade de saída. As informações na unidade de memória são, em geral, perdidas quando o computador é desligado. A unidade de memória costuma ser chamada de memória ou memória principal.

Unidade Lógica e Aritmética (ALU): Esta é a seção de 'fabricação' do computador. São responsáveis pela realização de cálculos, como adição, subtração, multiplicação e divisão. Esta seção contém os mecanismos de decisão

que permitem o computador, por exemplo, comparar dois itens da unidade de memória para determinar se são iguais ou não.

Unidade de Processamento (CPU): Esta é a seção 'administrativa' do computador. Ela coordena e supervisiona a operação das outras seções. A CPU diz à unidade de entrada quando as informações devem ser lidas e transferidas para a unidade de memória, informa à ALU quando as informações da unidade de memória devem ser utilizadas em cálculos e instrui a unidade de saída sobre quando enviar as informações da unidade de memória para certos dispositivos de saída. Muitos computadores atuais têm múltiplas CPUs e, portanto, podem realizar muitas operações simultaneamente - esses computadores são chamados de multiprocessadores.

Unidade de armazenamento secundária: Esta é a seção de 'armazenamento' de alta capacidade e de longo prazo do computador. Programas ou dados que não são utilizados ativamente pelas outras unidades, em geral, são colocados em dispositivos de armazenamento secundário (por exemplo, unidades de disco) até que sejam novamente necessários, talvez horas, dias, meses ou até mesmo anos mais tarde. As informações no armazenamento secundário exigem muito mais tempo para serem acessadas do que aquelas da memória principal, mas o custo por unidade de armazenamento secundário é muito menor que o da memória principal. Os exemplos de dispositivos de armazenamento secundário incluem CDs e DVDs, que podem armazenar até centenas de milhões de caracteres e bilhões de caracteres, respectivamente.

Dados

Cada unidade mínima de dado que pode ser manipulado pelo PC é chamada de bit (**Binary Digit**), e corresponde a um pulso elétrico. O bit pode assumir apenas um dos dois valores: 0 ou 1 (binário). O conjunto de 8 bits forma 1 byte, um conjunto de 1024 bytes (portanto $1024 \times 8 \text{ bits} = 8192 \text{ bits}$) formam 1KB e assim por diante conforme apresenta a tabela abaixo:

Unidade	Em bits		
bit (b)	1	1 b	2
Byte (B)	8	8 B	2^3
Kilobyte (kB)	8192	1024 KB	2^10
Megabyte (MB)	8388608	1024MB	2^20
Gigabyte (GB)	8589934592	1024GB	2^30
Terabyte (TB)	8796093022208	1024TB	2^40
Exabyte (EB)	9007199254740990	1024EB	2^50
Zettabyte (ZB)	9223372036854780000	1024ZB	2^60
Yottabyte (YB)	94447329657392900000000	1024YB	2^70
Xentabyte (XB)	96714065569170300000000000	1024XB	2^80
Wektabyte (WB)	99035203142830400000000000000	1024WB	2^90

É através desses pulsos elétricos que a informação trafega e é manipulada pelo computador.

Primeiros sistemas operacionais

Os primeiros computadores realizavam apenas um trabalho ou tarefa por vez. Isso costuma ser chamado de processamento em lotes de um único usuário. O computador executa um único programa por vez enquanto está processando dados em grupos ou lotes. Nesses antigos sistemas, os usuários geralmente enviavam os trabalhos para um centro de processamento de dados em unidades de cartões perfurados e, com frequência, tinham de esperar horas ou até dias antes de as impressões retornarem para suas mesas.

Os sistemas de software, chamados de sistemas operacionais, foram desenvolvidos para tornar a utilização de computadores mais conveniente. Os primeiros sistemas operacionais tornaram a transição entre trabalhos mais suave e mais rápida e, portanto, aumentaram a quantidade de trabalho, ou throughput, que os computadores poderiam processar.

Como os computadores tornaram-se mais poderosos, ficou evidente que o processamento em lotes de um único usuário era ineficiente, porque uma grande quantidade de tempo era gasta esperando dispositivos de entrada/saída lentos completarem suas tarefas.

Também se imaginou que muitos trabalhos ou tarefas poderiam compartilhar os recursos do computador para alcançar melhor utilização. Isso é chamado de multiprogramação. A multiprogramação envolve a operação simultânea de muitos trabalhos que competem para compartilhar os recursos do

computador. Com os primeiros sistemas operacionais baseados em multiprogramação, os usuários ainda submetiam trabalhos em unidades de cartões perfurados e horas de espera ou dias para resultados.

Na década de 1960, vários grupos empresariais e universitários foram os pioneiros dos sistemas operacionais de compartilhamento de tempo. O compartilhamento de tempo é um caso especial da multiprogramação em que usuários acessam o computador por meio de terminais, em geral, dispositivos com teclados e telas. Dúzias ou centenas de usuários compartilham o computador de uma vez. Na verdade, o computador não executa todos eles simultaneamente. Em vez disso, ele executa uma pequena parte do trabalho de um usuário, e, em seguida, atende o próximo usuário, talvez fornecendo serviço para cada usuário, várias vezes por segundo. Portanto, os programas dos usuários parecem estar executando simultaneamente. Uma vantagem do compartilhamento de tempo é que as solicitações de usuário recebem respostas quase imediatas.

"Quem pensa segundo a opinião dos outros, está muito longe de ser um homem livre." - Anônimo

Conceitos de Lógica

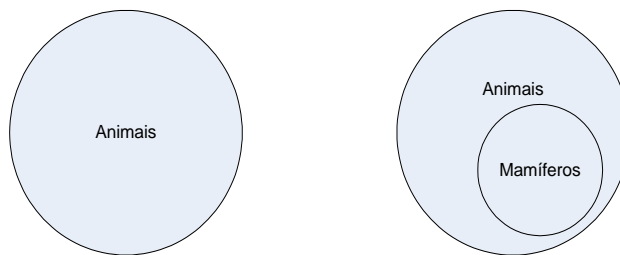
Conjuntos

Conjunto é uma coleção (ou classe) de objetos (elementos ou membros).

Ex. Conjunto dos animais.

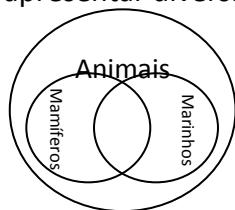
Subconjunto são conjuntos menores dentro de conjunto maiores.

Ex. Conjunto dos animais mamíferos.



Conjunto dos Animais Conjunto dos Animais Mamíferos

Os elementos são reunidos em conjuntos por algumas características que apresentam e que se queira estudar. Para facilitar o estudo desses elementos, o conjunto pode ser subdividido em diversos subconjuntos que acabem por apresentar diversas relações.



No conjunto dos animais temos o subconjunto dos Animais marinhos e o subconjunto dos animais mamíferos. Existe, entretanto animais marinhos que são mamíferos, as baleias.

Expressões

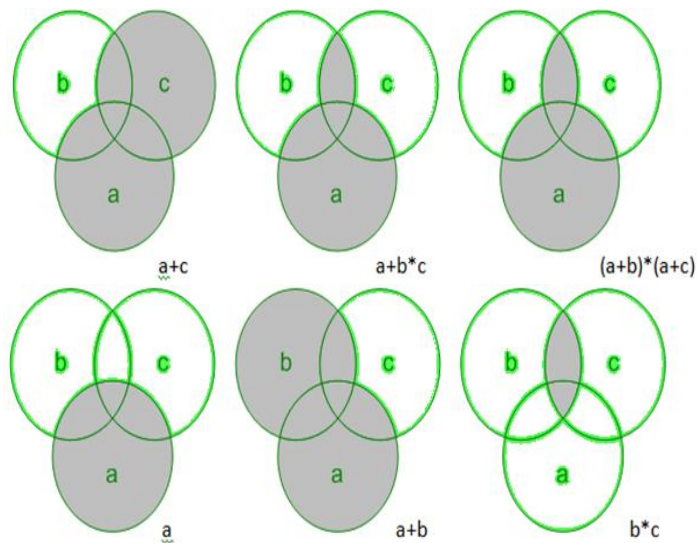
Para representar relações mais complexas também usamos expressões.

OBSERVAÇÃO: Em conjuntos $a+b*c = (a+b)*(a+c)$ diferentemente do que ocorre com a aritmética:

Observe. Supondo que $a=1$, $b=2$ e $c=3$, temos:

$a+b*c = 1+2*3 = 1+6 = 7$	$(a+b)*(a+c) = (1+2)*(1+3) = (3)*(4) = 12$
---------------------------	--

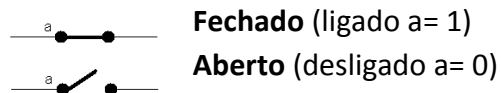
Portanto a relação $7=12$ **não é verdadeira** aritmeticamente.



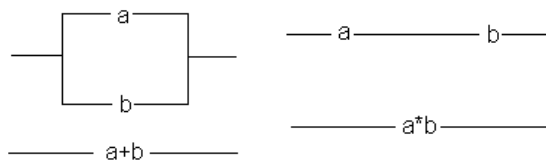
Vejamos a mesma relação em termos de conjuntos: O que deixa claro que, em conjuntos, é perfeitamente correta a igualdade: $a+b*c = (a+b)*(a+c)$.

Interruptor

É um dispositivo ligado a um ponto do circuito elétrico podendo assumir dois estados, conforme apresentado abaixo:

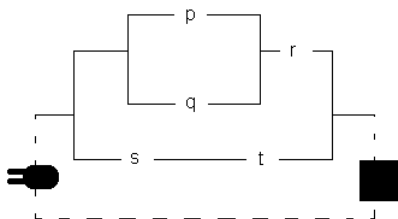


As ligações de interruptores podem ser feitas em **paralelo** ou em **série**:



Ligação em Paralelo Ligação em Série

De acordo com o interesse, uma combinação dessas ligações é possível, gerando dessa forma os mais diversos circuitos e, portanto, as mais diversas equações.



Vamos a um exemplo:

$$(p + q) * r + s * t$$

Com uma única variável **p** é possível dois valores (0 ou 1). No caso do circuito

acima com 5 variáveis, temos 2^5 , ou 32 combinações possíveis. Se acrescentássemos mais uma única variável esse número dobraria, passando para 64 conforme mostra a tabela abaixo:

Numero de Variáveis	Combinações Possíveis
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024

Lógica Formal

A lógica formal é a lógica de Boole, que trata de proposições e suas relações.

Proposições

São conceitos primitivos, sentenças afirmativas que devem exprimir um pensamento de sentido completo, podendo ser escrita na forma simbólica ou na linguagem usual.

As proposições podem assumir um dos dois valores lógicos:

VERDADEIRA. Representamos o valor lógico da proposição **p**, $v(p) = 1$

FALSA. Representamos o valor lógico da proposição **p**, $v(p) = 0$

As proposições podem ser simples, representada por letra minúscula, quando não há nenhuma outra proposição adicional em si :

p: Pedro é rico.

q: Estudar faz bem.

As proposições podem ser compostas, representadas por letras maiúsculas, formadas por duas ou mais proposições relacionadas por conectivos lógicos (E ou OU):

a) $p = 1 + 2 = 3$

b) $q = 2 \neq 1$

P: $p * q$ $1 + 2 = 3$ E $2 \neq 1$

Exemplo:

Q: $p+q$ $1 + 2 = 3$ **OU** $2 \neq 1$

A melhor maneira de organizar variáveis lógicas, ou **proposições**, é utilizando a **Tabela Verdade**. O número de linhas da tabela é proporcional ao número de proposições. Para uma proposição são necessárias duas linhas, para duas, são quatro linhas e assim por diante.

p	q
0	0
0	1
1	0
1	1

Princípios fundamentais da lógica

1 – **Princípio da não contradição**: Uma proposição não pode ser ao mesmo tempo verdadeira e falsa.

2 – **Princípio do terceiro excluído**: A Proposição é só verdadeira, ou só falsa, não ocorrendo um terceiro caso.

A **Tabela Verdade** nos permite entender o comportamento da relação entre as proposições quando aplicadas operações lógicas entre elas. Cada proposição é uma coluna e as linhas são as possíveis combinações:

- **Negação** (não, not, '): Representa o complemento da proposição.
- **Conjunção** (E, AND, *): Só é Verdadeira quando ambas forem verdadeiras.
- **Disjunção Inclusiva** (OU, OR, +): Só é Falsa quando todas forem Falsas.
- **Disjunção Exclusiva** (Exclusive Or, Ou Exclusivo, XOR, \oplus): O $p \oplus q$ é lido “p OU q mas NÃO ambas”
- **Condiciona** (\rightarrow): Falsa quando $v(p) = 1$ E $v(q) = 0$ sendo verdadeira nos demais casos. **p** é antecedente e **q** é o conseqüente. $p \rightarrow q$ lê – se: Se p então q.
- **Bi condicional** – coincidência ($p \leftrightarrow q$): Verdadeira quando $v(p)=v(q)$ e Falsa quando $v(p) \neq v(q)$. Lê – se $p \leftrightarrow q$ “p se e somente se q”.

p	p'
0	1
1	0

Negação

p	q	$p+q$	$p*q$	$p \oplus q$	$p \rightarrow q$	$p \leftrightarrow q$
0	0	0	0	0	1	1
0	1	1	0	1	1	0
1	0	1	0	1	0	0
1	1	1	1	0	1	1

Aplicadas a uma equação, as operações apresentadas seguem a seguinte ordem de precedência:

- 1) Negação (')
- 2) Conjunção (*)
- 3) Disjunção (+ \oplus)
- 4) Condicional (\rightarrow)
- 5) Bicondicional (\leftrightarrow)

Exemplo: $P(p,q) = (p * q')'$

P	q	q'	$p * q'$	$(p * q')'$
0	0	1	0	1
0	1	0	0	1
1	0	1	1	0
1	1	0	0	1

De outra maneira podemos dizer que:

$P(00, 01, 10, 11) = 1101$, $P(00) = 1$, $P(01) = 1$, $P(10) = 0$, $P(11) = 1$

Chamadas:

Tautologia \Rightarrow $(P(00, 01, 10, 11) = 1111)$

Contradição \Rightarrow $(P(00, 01, 10, 11) = 0000)$

Contingência \Rightarrow Nos demais casos (também chamada Indeterminação)

Proposições Independentes: Todas as combinações possíveis na TV

Proposições Dependentes: Quando uma (relação simples) ou duas (relação dupla) não ocorrem.

P	Q
0	0
0	1
1	0
1	1

Independente

P	Q	$q \rightarrow p$
0	0	1
0	1	0
1	0	1
1	1	1

Dependente: Não ocorre 10 nesta combinação.

Relação de Implicação: Dizemos que uma proposição **p** implica em uma proposição **q** quando em suas TVs não ocorre 10, nesta ordem, e representa – se da seguinte forma: $p \Rightarrow q$ (lê – se: p implica q). Observe que $p \rightarrow q$ é uma operação $p \Rightarrow q$ é a relação, portanto $p \rightarrow q \neq p \Rightarrow q$.

Relação de Equivalência: p é equivalente a q quando não ocorrer nem 10 nem 01. $p \Leftrightarrow q$ (p é equivalente a q) e da mesma forma devemos lembrar que $p \leftrightarrow q \neq p \Leftrightarrow q$.

p	Q	$q \rightarrow p$
0	0	1
0	1	0
1	0	1
1	1	1

Implicação: $p \Rightarrow q \rightarrow p$

p	q	$p * q$	$(p' + q')'$
0	0	0	0
0	1	0	0
1	0	0	0
1	1	1	1

Equivalência: $p * q \Leftrightarrow (p' + q')'$

Equivalências Notáveis.

- Dupla Negação: a) $(p')' \Leftrightarrow p$
- Lei Idempotentes: a) $p+p \Leftrightarrow p$ b) $p*p \Leftrightarrow p$
- Leis Comutativas: a) $p+q \Leftrightarrow q+p$ b) $p*q \Leftrightarrow q*p$
- Leis Associativas: a) $p+(q+r) \Leftrightarrow (p+q)+r$ b) $p*(q*r) \Leftrightarrow (p*q)*r$
- Lei de De Morgan: a) $(p*q)' \Leftrightarrow p'+q'$ b) $(p+q)' \Leftrightarrow p'*q'$
- Leis Distributivas: a) $p*(q+r) \Leftrightarrow p*q+p*r$ b) $p+q*r \Leftrightarrow (p+q)*(p+r)$
- Condicionais: a) $p \rightarrow q \Leftrightarrow q' \rightarrow p'$ b) $q \rightarrow p \Leftrightarrow p' \rightarrow q'$
- Bicondicional: a) $p \Leftrightarrow q \Leftrightarrow (p \rightarrow q)*(q \rightarrow p)$

Demonstração de equivalências nas Tabelas Verdade:

DUPLA NEGAÇÃO		
P	p'	(p')'
0	1	0
1	0	1

LEIS IDEMPOTENTES		
p	p+p	p*p
0	0	0
1	1	1

LEIS COMUTATIVAS					
P	q	p+q	q+p	p*q	q*p
0	0	0	0	0	0
0	1	1	1	0	0
1	0	1	1	0	0
1	1	1	1	1	1

LEIS ASSOCIATIVAS - Primeiro caso						
P	Q	r	q+r	p+(q+r)	(p+q)	(p+q)+r
0	0	0	0	0	0	0
0	0	1	1	1	0	1
0	1	0	1	1	1	1
0	1	1	1	1	1	1
1	0	0	0	1	1	1
1	0	1	1	1	1	1
1	1	0	1	1	1	1

1	1	1	1	1	1	1	
LEIS ASSOCIATIVAS – Segundo caso							
P	Q	r	$q*r$	$p*(q*r)$	$(p*q)$	$(p*q)*r$	
0	0	0	0	0	0	0	
0	0	1	0	0	0	0	
0	1	0	0	0	0	0	
0	1	1	1	0	0	0	
1	0	0	0	0	0	0	
1	0	1	0	0	0	0	
1	1	0	0	0	1	0	
1	1	1	1	1	1	1	
LEIS DISTRIBUTIVAS – Primeiro caso							
P	Q	r	$q+r$	$p*(q+r)$	$p*q$	$p*r$	$p*q+p*r$
0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	1	0	0	0	0
0	1	1	1	0	0	0	0
1	0	0	0	0	0	0	0
1	0	1	1	1	0	1	1
1	1	0	1	1	1	0	1
1	1	1	1	1	1	1	1
LEIS DISTRIBUTIVAS – Segundo caso							
P	q	R	$q*r$	$p+(q*r)$	$p+q$	$p+r$	$(p+q)*(p+r)$
0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0
0	1	0	0	0	1	0	0
0	1	1	1	1	1	1	1
1	0	0	0	1	1	1	1
1	0	1	0	1	1	1	1
1	1	0	0	1	1	1	1
1	1	1	1	1	1	1	1
LEI DE DE MORGAN – Primeiro caso							

P	q	p'	q'	p*q	(p*q)'	p'+q'
0	0	1	1	0	1	1
0	1	1	0	0	1	1
1	0	0	1	0	1	1
1	1	0	0	1	0	0
LEI DE DE MORGAN – Segundo caso						
p	q	p'	q'	p+q	(p+q)'	p'*q'
0	0	1	1	0	1	1
0	1	1	0	1	0	0
1	0	0	1	1	0	0
1	1	0	0	1	0	0

OBSERVAÇÃO: A Lei de De Morgan também é válida para os casos:

a) $p'*q \Leftrightarrow (p+q)'$; b) $p'+q \Leftrightarrow (p*q)'$; c) $(p'*q)' \Leftrightarrow p+q'$ d) $(p'+q)' \Leftrightarrow p*q'$

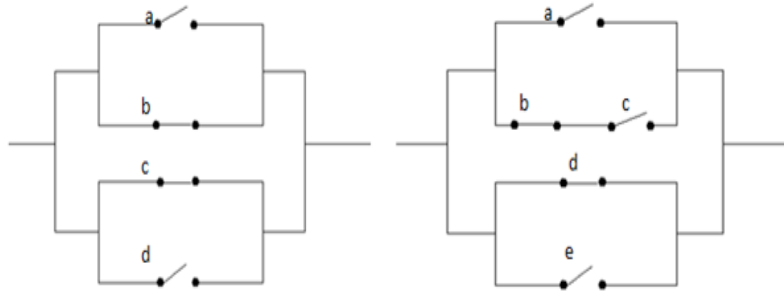
BICONDICIONAL							
P	Q	$p \leftrightarrow q$	$p \rightarrow q$	$q \rightarrow p$	$(p \rightarrow q) * (q \rightarrow p)$		
0	0	1	1	1	1		
0	1	0	1	0	0		
1	0	0	0	1	0		
1	1	1	1	1	1		
CONDICIONAIS							
P	q	p'	q'	$p \rightarrow q$	$q' \rightarrow p'$	$q \rightarrow p$	$p' \rightarrow q'$
0	0	1	1	1	1	1	1
0	1	1	0	1	0	0	0
1	0	0	1	0	1	1	1
1	1	0	0	1	1	1	1

Exercícios Complementares

1 - Dadas as equações abaixo, apresente os circuitos correspondentes:

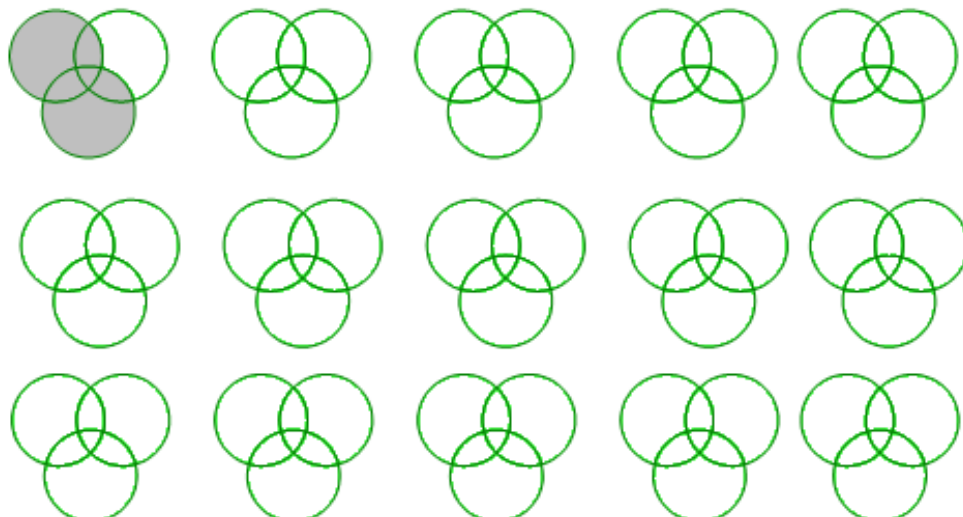
a) $a*b*c$ b) $a*b+c'$ c) $(b*c')*(a+d')$ d) $a+b*c$ e) $a*(b+c)+a'*b$

2 – Dados os circuitos abaixo, apresente as equações correspondentes:



3 - Apresente o resultado das seguintes equações. Use os diagramas abaixo conforme o exemplo:

	1	2	3	4	5
A	$a+b$	$a+c$	$a*a'$	$a*b$	$a*c$
B	$b*c$	$a+b+c$	$((a*c'))'*b$	$a+a'$	$b'+(c*a)$
C	$a'+b'+c'$	$a'+b'*c$	$(b*c')*(a+b')$	$a+b*c$	$(a+b)'*(a+c)'$



4 - Resolva as equações abaixo, utilizando a Tabela Verdade:

a) $p + (q * r)$

b) $p' + (r * q)$

c) $p' * (q' + r)$

d) $p' + r + q'$

e) $p * q * r * (r + q)'$

f) $p + (p' * q) * q'$

g) $p + r * q$

h) $q' + p' * r$

i) $p + (q + (r + (p * q * r)))$

j) $p + q + r + p * q * r$

k) $(p * (q + r))'$

l) $r \rightarrow (p + q) * (q \rightarrow p)'$

m) $(p \rightarrow (q \leftrightarrow r) \rightarrow q' \rightarrow (p \leftrightarrow r))'$

n) $p + (q \rightarrow r)$

o) $(p * q) \leftrightarrow (p * r) \leftrightarrow (p * q)' \leftrightarrow (p * r)'$

p) $((p * r)' \rightarrow (p * q)')'$

q) $(p * r)' \rightarrow (p' * q)'$

r) $(p' * r)' \rightarrow (p * q)'$

s) $(p + r)' * (p + q)' + (p * r)'$

t) $((((p + q) + (p * r)' \rightarrow (p * s) + (q * r))' \leftrightarrow (p * s)))'$

"Falar é fácil. Mostre-me o código." - *Linus Torvalds*

Programa

Um programa é uma sequência de passos ordenados com o objetivo de resolver um problema computacional que obedecem as regras sintáticas de uma linguagem de programação. Em outras palavras, é um algoritmo que respeita as regras de uma linguagem de programação.

Programar é, portanto, organizar passos com clareza e de maneira correta, pensando em cada detalhe e possibilidade de falha, escrevendo o programa de maneira que o computador consiga executar exatamente aquilo que se espera, sem apresentar erros, ainda que por culpa do uso incorreto.

Linguagem de Alto Nível e Baixo Nível.

A **Linguagem de baixo nível** está muito próxima da linguagem de máquina e é, portanto, difícil de compreender, difícil de programar, e facilmente se cometer erros difíceis de encontrar.

As **Linguagens de alto nível** estão mais próximas da linguagem humana, seguem uma estrutura própria de programação, são fáceis de entender, mais simples para o desenvolvimento e com maior facilidade se localiza falhas no algoritmo.

BAIXO NÍVEL (ARQUITETURA INTEL)	ALTO NÍVEL (Linguagem Java)
7C90EC28 mov word ptr [eax+000000C8h] 7C90EC2E mov dword ptr [eax],10007h 7C90EC34 push 1 7C90EC36 push eax 7C90EC37 push dword ptr [ebp+8] 7C90EC3A call 7C90E252 7C90EC3F sub esp,20h 7C90EC42 mov dword ptr [esp],eax 7C90EC45 mov dword ptr [esp+4],1 7C90EC4D mov dword ptr [esp+10h],0 7C90EC55 mov eax,dword ptr [ebp+8] 7C90EC58 mov dword ptr [esp+8],eax 7C90EC5C mov eax,esp 7C90EC5E push eax 7C90EC5F call 7C90EBAC	public class Numero { public static void main(String args[]) { double x,y; x = 2; y = 3.0; y = y + x; System.out.println("x+y = " + (x+y)); } }

Cada linguagem apresenta suas próprias regras sintáticas e suas palavras reservadas. Isso varia de linguagem para linguagem e é necessário um conhecimento desses detalhes antes de começar a programar. Depois disso é necessário praticar. Somente com a prática se aprende a programar.

Programação

Linear	Estruturada	Orientada a Objeto
<p>Sequência de passos executados consecutivamente com início e fim específicos. Um bom exemplo é a linguagem BASIC, com linhas numeradas e uso de desvios "GOTO".</p> <p>A linguagem linear é muito complexa.</p> <p>Os programas extensos tornam – se difíceis de entender e desenvolver.</p>	<p>Modularização: "Dividir para conquistar".</p> <p>O módulo deve executar tarefas específicas e de forma independente.</p> <p>Procedimento e função: bloco para executar tarefa específica, podendo ambos receber valores, mas apenas a função retorna.</p> <p>Problema: Como dividir e gerenciar a tarefa do desenvolvimento?</p>	<p>Mudança de enfoque: Modelo tradicional: Conjunto de programas.</p> <p>Modelo Orientado a Objeto: O mundo é um conjunto de objetos que interagem e apresentam características e comportamentos próprios representados por seus atributos (dados) e suas operações (processo que o objeto executa).</p>

Um exemplo da diferença entre a programação linear e estruturada.

Exemplo de Programação Linear	Exemplo de programação estruturada
<pre> 10 REM RESOLVE EQUACAO DO SEGUNDO GRAU 20 READ A,B,C 25 IF A=0 THEN GOTO 410 30 LET D=B*B-4*A*C 40 IF D<0 THEN GOTO 430 50 PRINT "SOLUCAO" 60 IF D=0 THEN GOTO 100 70 PRINT "PRIMEIRA SOLUCAO",(-B+SQR(D))/(2*A) 80 PRINT "SEGUNDA SOLUCAO",(-B-SQR(D))/(2*A) 85 GOTO 500 90 GOTO 20 100 PRINT "SOLUCAO UNICA",(-B)/(2*A) 150 GOTO 500 200 GOTO 20 410 PRINT "A DEVE SER DIFERENTE DE ZERO" 420 GOTO 20 430 PRINT "NAO HA SOLUCOES REAIS" 440 GOTO 20 490 DATA 10,20,1241,123,22,-1 500 END </pre>	<pre> Leia (A, B, C) Se (A = 0) Entao Escreva("A deve ser diferente de 0") Senao D<-B*B-4*A*C Se (D=0) Entao Escreva("Solução única: ", -B/(2*A)) Senao Escreva("Primeira Solução: ",(-B+RaizQ(D))/(2*A)) Escreva("Segunda Solução: ", (-B-RaizQ(D))/(2*A)) FimSe FimSe </pre>

Linguagens de máquina, Linguagens Assembly

Qualquer computador pode entender diretamente somente sua própria linguagem de máquina . A linguagem de máquina é a 'linguagem natural' de um computador e é definida pelo seu projeto de hardware. As linguagens de máquina consistem geralmente em strings de números (em última instância reduzidas a 1s e Os) que instruem os computadores a realizar suas operações

mais elementares uma de cada vez. As linguagens de máquina são dependentes de máquina (isto é, uma linguagem particular de máquina pode ser utilizada apenas em um tipo de computador). Essas linguagens são muito complexas para os seres humanos.

+1300042774

+1400593419

+1200274027

A programação de linguagem de máquina era simplesmente muito lenta e tediosa para a maioria dos programadores. Em vez de utilizar as strings de números que os computadores poderiam entender diretamente, os programadores começaram a utilizar abreviações em inglês para representar operações elementares. Essas abreviações formaram a base de linguagens assembly. Programas tradutores assemblers foram desenvolvidos para converter os primeiros programas de linguagem assembly em linguagem de máquina a velocidades de computador.

load basepay
add overpay
store grosspay

Linear

A técnica lógica linear é conhecida como um modelo tradicional de desenvolvimento e resolução de um problema. Não está ligada a regras de hierarquia ou de estruturas de linguagens específicas de programação de computadores. Devemos entender que este tipo de procedimento está voltado à técnica matemática, a qual permite determinar a atribuição de recursos limitados, utilizando uma coleção de elementos organizados ou ordenados por uma só propriedade, de tal forma que cada um deles seja executado passo a passo de cima para baixo, em que tenha um só predecessor e um só sucessor.

Estruturada

A técnica da lógica estruturada é a mais usada pelos profissionais de processamento eletrônico de dados. Possui características e padrões particulares, os quais diferem dos modelos das linguagens elaboradas por seus

fabricantes. Tem como pontos fortes para elaboração futura de um programa produzem-no com alta qualidade e baixo custo.

Modular

A técnica da lógica modular deve ser elaborada como uma estrutura de partes independentes, denominada de módulos, cujo procedimento é controlado por um conjunto de regras. Segundo James Martin, suas metas são as seguintes:

- Decompor um diagrama em partes independentes;
- Dividir um problema complexo em problemas menores e mais simples;
- Verificar a correção de um módulo de blocos, independentemente de sua utilização como uma unidade em um processo maior.

A modularização deve ser desenvolvida, se possível, em diferentes níveis. Poderá ser utilizada para separar um problema em sistemas, um sistema em programas e um programa em módulos.

Orientação a Objeto

As Linguagens Orientadas a Objeto trazem consigo uma série de conceitos importantes que veremos de forma geral a seguir:

OBJETO: Podemos dividir os objetos como, tangíveis e intangíveis. Tangíveis são objetos do mundo real nos quais podemos tocar, tais como carros, prédios, pessoas, brinquedos, entre outros. Intangíveis são os que não podemos tocar, mesmo sabendo que existem, tais como transação bancária, compra, venda, e inúmeros outros. Ambos são objetos porque possuem atributos (forma cor peso, tamanho, etc.) e comportamentos (um carro anda, freia, acelera, etc.).

INSTÂNCIA DE OBJETO: A instância é criação do objeto na memória.

CLASSE: Classe é o modelo do objeto. Em outras palavras é na classe que se define atributos e comportamentos dos objetos. Pode – se criar diversos objetos a partir de uma única classe.

GENERALIZAÇÃO: Classe com atributos e comportamentos mais gerais. Por exemplo, a classe “Telefone” define um aparelho para fazer e receber

chamadas, composto por microfone, saída de áudio e teclas. Não define, entretanto se é um orelhão, um celular ou um telefone fixo.

ESPECIALIZAÇÃO: A especialização trata de casos mais específicos, no caso do exemplo acima, seria, por exemplo, a classe “Telefone_Fixo”.

HERANÇA: A criação de uma classe mais específica a partir de uma classe mais genérica chama – se herança. A classe mais específica (subclasse ou classe Filha) herda da classe mais genérica (Superclasse ou classe Pai) todos os atributos e comportamentos, e recebe alguns atributos e comportamentos específicos.

POLIMORFISMO: É a possibilidade de executar a mesma tarefa (método) de diversas formas. Quem decide a maneira como o método será executado é o objeto e este comportamento só se define em tempo de execução.

SOBRECARGA: Em uma mesma classe, dois ou mais métodos podem ter o mesmo nome, desde que sua assinatura seja diferente. Assinatura, são os parâmetros de entrada do método.

REDEFINIÇÃO DE MÉTODO: Acontece quando um método, cuja assinatura já tenha sido especificada, recebe nova definição (novo corpo) em uma classe derivada.

ENCAPSULAMENTO: O encapsulamento ou ocultamento (empacotamento) protege os atributos e operações dos objetos sendo possível sua utilização por meio de uma interface bem definida, sem ser possível, entretanto, alterá-lo. Um excelente exemplo é o carro: Você pode dirigir o carro (objeto) mesmo sem saber como funcionam os sistemas de freio, aceleração, etc. (atributos e operações) através dos pedais, volante e cambio (interface).

Tipos de tradutores

Os códigos escritos em linguagem de alto nível, não são compreendidos pela máquina e precisam passar por uma tradução. É dessa forma que o computador entende os códigos e executa de maneira correta os comandos.

Há duas formas para acontecer essa tradução.

A Compilação cria uma dependência de máquina, processador e sistema operacional. O programa é traduzido para a máquina e o sistema do computador onde ele é compilado, podendo ser usado em outros com as mesmas características, mas podendo apresentar diversos problemas em computadores com processador e sistema operacional diferente.

A Interpretação é outra forma que acontece em tempo de execução. O código escrito em linguagem de programação de alto nível, é lido linha a linha e interpretado em linguagem de máquina, durante a execução do programa. Isso garante que o programa funcione em qualquer máquina, entretanto o programa perde em desempenho. Existe ainda a forma híbrida, como é o caso da linguagem Java. Os programas escritos em Java, passam pelos dois processos, de forma que o melhor de cada um deles é aproveitado.

Os programas são escritos em alto nível e são compilados em bytecode, código de máquina para a máquina virtual Java. Uma vez compilado, os bytecodes podem ser interpretados por qualquer máquina real que possua a máquina virtual Java. A máquina virtual interpreta os bytecodes para a linguagem de máquina que aquele processador e sistema operacional entendam. Isso garante portabilidade, ou seja, uma vez escrito o código, funcionará em qualquer computador. A prévia compilação em Bytecode também garante um desempenho melhor dos programas.

Antes de começar a programar.

Antes de sentar e fazer os seus programas na linguagem de programação escolhida, é importante que alguns passos sejam seguidos a fim de obter melhores resultados:

- 1º Tenha uma ideia clara do que pretende fazer.
- 2º Escreva a ideia de modo narrativo para corrigir possíveis imperfeições.
- 3º Reescreva e altere enquanto for necessário.
- 4º Divida o problema em partes menores.
- 5º Elabore soluções para cada uma das partes menores.
- 6º Construa fluxogramas para as soluções elaboradas.
- 7º Verifique se as soluções são válidas e corrija imperfeições.
- 8º Transcreva os fluxos em Pseudocódigos.
- 9º Verifique novamente se é necessária, alguma alteração.
- 10º Reescreva o que for necessário.

Embora pareça complexo e demorado, a primeira vista, o desenvolvimento dessas atividades é rápido, e permite economia de tempo no futuro, uma vez que todos os erros são mais facilmente encontrados e mais rapidamente corrigidos. Uma vez feito todos os passos acima é hora de passar seu código de português estruturado para uma linguagem de programação. Para evitar erros, é interessante que você observe alguns pontos sobre a linguagem escolhida antes de começar a codificar:

- 1º As palavras reservadas da linguagem.
- 2º Os operadores, relacionais, aritméticos, lógicos e de atribuição.
- 3º As estruturas de decisão e laço.
- 4º Como declarar uma variável (Local e Global)
- 5º Se a linguagem diferencia letras maiúsculas e minúsculas.
- 6º A API (Application Programming Interface) da linguagem.
- 7º Um IDE (Integrated Development Environment)

Algoritmos

Para um programador, é de sua importância e sobrevivência saber organizar seus pensamentos a fim de resolver os problemas básicos do cotidiano. O programador pensa na solução e aplica na vida real.

O Curso de Lógica de Programação tem como objetivo, fazer com que o aluno possa sistematizar uma solução de forma simples, direta e que qualquer pessoa entenda. Ele é a base para se aprender qualquer linguagem de programação.

O “projeto básico” de uma solução de problemas é o algoritmo, a partir dele, construímos o diagrama de blocos para sistematizar a solução e caso haja importância para a resolução passamos para a linguagem de Programação, ou seja, programamos o nosso algoritmo de forma computacional.

Primeiramente, O que é um problema?

➔ Uma situação que exige que seja feito algo

Como assim? Como resolvemos o problema? A informática veio para facilitar a nossa vida, transformando a resolução do problema em passos finitos, discretos e simples, o chamado algoritmo, isso para que se entenda melhor o que se deve fazer para que se obtenha o resultado.

O algoritmo tende a pegar a realidade, colocá-la no papel e nos mostrar um meio de resolver o problema, uma vez resolvido, é a hora de aplicar o algoritmo na realidade.



(É óbvio que o algoritmo não age sozinho, é preciso alguém para desenvolvê-lo. Ele é apenas as anotações de alguém que mostra uma “receita” da resolução.)

Ex: Maria quer suco de laranja.

Para isso seguimos uma lógica básica:

- 1) Pegar as laranjas,
- 2) Lavar,
- 3) Cortar,
- 4) Espremer as laranjas,
- 5) Servir.

Essa lógica nada mais é que o algoritmo feito para o problema “Maria quer sucos”, fácil de fazer, não? Vamos aprender a fazer...

Devemos primeiramente nos focar em três perguntas básicas:

O que quero?

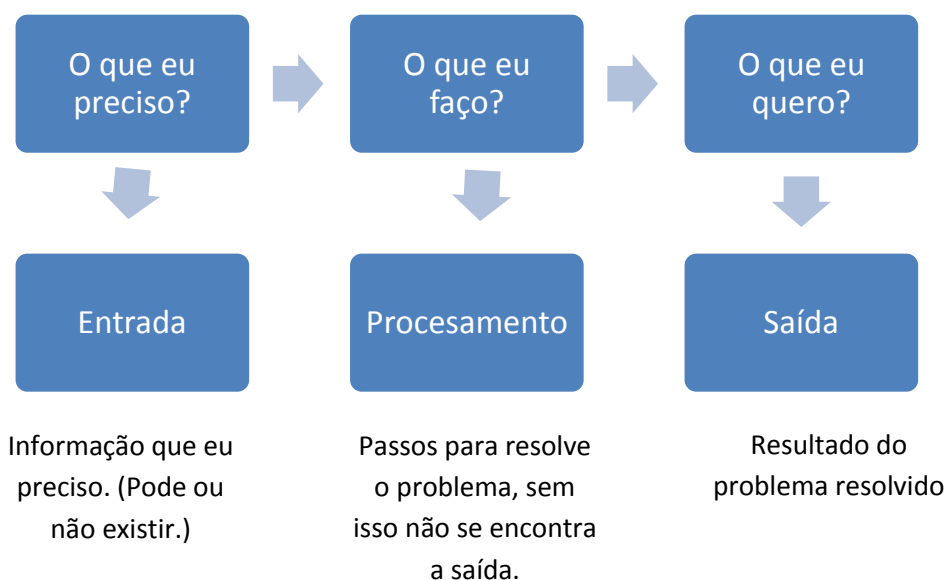
O que eu preciso?

O que eu faço?

No exemplo de Maria, vemos que o problema é basicamente o seguinte: O que Maria quer? – Suco de Laranja; Responder essa pergunta é fundamental, pois explicita o que se quer no problema, ou seja, a isso chamamos de Saída.

A saída é o nosso objetivo, é a resolução do nosso problema e a razão de viver daquele problema. Para se chegar a saída é preciso processar o problema, ou seja, encontrar/descrever os caminhos que, seguidos rigorosamente, nos dão a resposta, isso chamamos de Processamento.

Para se encontrar esses caminhos é preciso conhecer o problema e muitas vezes analisar, armazenar e usar informações dele, por exemplo, no caso de Maria é fundamental saber, a quantidade de laranjas, para se saber o quanto de água e açúcar utilizar, precisamos ter essa informação, ou seja, “o que eu preciso?”. De uma informação, a qual chamamos de Entrada.



A informática surgiu para salvar a humanidade nos ajudando com seu poder milagroso a resolver os problemas da vida... (há gente que fala que o

computador veio para resolver os problemas que antes não tínhamos...). E tudo isso graças a algo chamado *Programa*. O que é essa coisa graciosa?

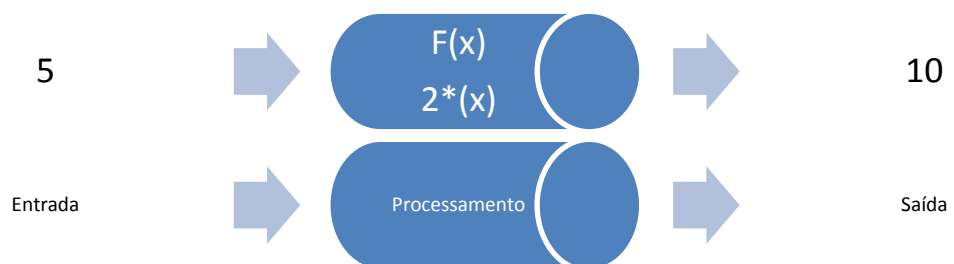
É um artifício da informática para resolver determinadas questões da realidade a fim de facilitar a vida mundana, ou seja, não passa de uma simples função que é criada a partir da transformação de um problema real em expressões Algébricas, ou seja, realidade em matemática.

Você já percebeu que funções que aprendemos na escola estão intimamente ligadas aos programas que estamos começamos a ver? Não? Um programa nada mais é que uma função mais complexa.

Ex: Dada uma função $f(x)$

$F(x)=2X$, se colocarmos entradas (números) nela, ela irá resolver um processo ($2*X$) e nos dará um resultado ($2x$).

Veja:



Ou seja algoritmo é uma sequência ordenada de passos que devem ser cumpridos com o fim de resolver um problema. Esta sequência de passos ou instruções pode ser apresentada de maneira empírica ou estruturada.

Há diversas maneiras de se estruturar um algoritmo, ou seja, descrever os passos que devem ser dados com o fim de solucionar um problema:

Descritivo – Narrativo: os passos são descritos em sequência de forma narrativa, sem preocupação com formalidades. Geralmente é usado como etapa inicial para se descrever de maneira informal a idéia de solução que se tem para o problema.

Diagrama de Chapin: os passos são apresentados em uma única caixa, tornando mais concisa a descrição e mais fácil visualiza - la.

Fluxograma: utiliza formas geométricas que representam cada um dos passos, tornando muito mais simples a visualização da sequência. O fluxograma é uma ótima ferramenta para estruturar a solução do problema e visualizar possíveis imperfeições na solução proposta.

Pseudocódigo: é a forma que mais se aproxima das linguagens de programação. Utilizando o português estruturado, codifica a sequência de passos como é feito em um programa real, tornando muito mais simples a tarefa de gerar o código para uma linguagem de programação. O programador, independente da linguagem com a qual trabalha, consegue entender e reescrever um código do português estruturado para a linguagem de programação com a qual trabalha.

Há diversas maneiras de solucionar um mesmo problema e uma modelagem bem feita pode resultar em uma solução melhor.

Temos um **Programa** quando as ações do algoritmo obedecem a uma sintaxe (regras de escrita) de uma linguagem de programação.

A criação de uma solução algorítmica deve atentar por resolver o problema proposto e os demais relacionados que possam surgir.

Por exemplo: Supondo que se queira somar dois números inteiros, deve – se ler os valores do dispositivo de entrada, armazená-los em variáveis, efetuar o processamento (soma) e disponibilizar o resultado no dispositivo de saída. Vamos supor, entretanto, que a pessoa que utiliza o programa, ao entrar com os valores, não respeite a condição de soma de inteiros e tenta somar valores decimais, ou ainda pior, entra com valores literais.

São problemas que devem ser previstos e tratados buscando a melhor qualidade e segurança no resultado final.

Para tanto o respeito – se um fluxo de desenvolvimento do algoritmo buscando evitar problemas, erros ou falhas que possam prejudicar o desempenho do programa ou mesmo impedir o seu correto funcionamento. O fluxo de criação é o seguinte:

- 1 – Definir perfeitamente o algoritmo, deixando – o claro e completo.
- 2 – Não permitir ambiguidades: não pode haver dúvidas sobre o que deve ser feito.
- 3 – Deve primar pela eficácia, resolvendo os problemas e não criando - os.
- 4 – Deve buscar sempre a eficiência, solucionando o problema com o mínimo de passos e, portanto, mais rapidamente possível.

[illegible]

"A inteligência é o farol que nos guia, mas é a vontade que nos faz caminhar." - Anônimo


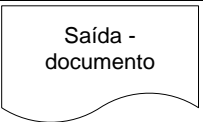
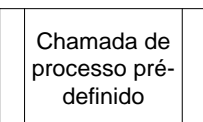

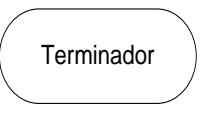
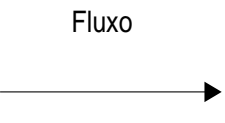

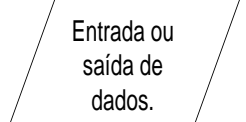
Técnicas de Programação



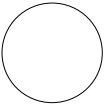

A seguir, começaremos a aprender técnicas para estruturar soluções de problemas em programas que computador. Não se preocupe, treine muito e logo você será fluente nessas técnicas.

Fluxogramas

Para auxiliar facilitar a visualização, agilizando o processo de compreensão do algoritmo, fazemos uso de símbolos geométricos (retângulos, losangos, etc.). Ao uso dos símbolos geométricos, ligados através de setas, para descrição dos algoritmos, damos o nome de **Fluxograma**.

Vários símbolos podem ser utilizados e muitos deles são bem específicos. Não cabe aqui, apresentar todos os símbolos possíveis, dada sua enorme variedade. Vários softwares que permitem a criação de fluxogramas, como o Microsoft® Visio, por exemplo, além de apresentar alguns símbolos de uso específico, apresentam ainda uma rápida e simplificada explicação da funcionalidade de cada um. Apresentamos os símbolos mais usuais abaixo:

	Qualquer processo (ações).		Indica a saída de documento.
	Chamada a códigos descritos fora do código atual.		Saída de informação no monitor.
	Indicador de início e término do fluxograma.		Indica o fluxo de dados e ações ligando os símbolos.
	Tomada de decisão cuja condição retorna uma resposta lógica.		Pode indicar a entrada/saída de dados por quaisquer dispositivos.

	Referência a Dados seqüenciais, por exemplo, gravados em fita magnética.		Entrada de dados via teclado.
Referência 	Usado quando o fluxograma é maior que o espaço disponível, indicando continuidade.		Texto explicativo que pode aparecer no código mas que será ignorado pelo compilador.

Pseudocódigo

O Pseudocódigo não é uma linguagem de programação, é uma maneira de apresentar o algoritmo de forma estruturada (Português Estruturado), o que fica muito próximo das verdadeiras linguagens de programação.

Utiliza – se a língua nativa para escrever o Pseudocódigo, no nosso caso, o Português.

As regras para a estrutura de um pseudocódigo podem ser um pouco flexíveis, entretanto, de maneira geral, segue – se o padrão apresentado abaixo:

Na primeira linha do Pseudocódigo, é informado o nome do programa.

Abaixo do nome do programa, algumas informações complementares na forma de **comentário**, tais como autor do programa, data, etc

Em seguida a declaração das variáveis globais na forma:
Nome_da_variavel:Tipo

De maneira geral, não é obrigatória a preocupação com o chamado “*case sensitive*”, ou seja, letras em maiúsculo e minúsculo. Aconselha – se, entretanto, manter um padrão. Uma vez que uma variável seja criada utilizando se letra maiúscula, cuide para sempre escreva – la da mesma forma, isso porque algumas linguagens são “*case sensitive*”. A prática previne erros.

Utiliza – se os delimitadores INICIO e FIM , no inicio e final do código.

Para receber dados, usamos o comando “Leia”.

Para imprimir dados, usamos o comando “Escreva”

Valores do tipo Literal são sempre apresentados entre aspas “ ”, para atribuição a variáveis e para impressão na tela.

Os textos colocados na mesma linha, após barras invertidas duplas representam comentários, e são, portanto, ignorados pelo interpretador.

São três as estruturas de controle de fluxos que podem ser utilizadas para resolução de problemas: **atribuição, condição e repetição**.

Atribuição

Atribuição é o ato de atribuir um valor a uma determinada variável respeitando seu domínio. É a mais básica das estruturas. Denotada pelo fluxograma como um retângulo. Uma simples expressão algébrica, ela é uma função comum, como estamos acostumados. É constituída basicamente por contas, termos, variáveis, constantes, entrada e saída de dados.

Tipos de informação

Os programas de computador manipulam dados através de variáveis.

As **Variáveis** são espaços que o programa reserva na memória para armazenar dados que são usados pelo programa durante a execução, como veremos mais a frente.

Estas variáveis, com relação ao conteúdo que armazenam, podem ser:

Literais: Representam os caracteres alfanuméricos e símbolos.

Numéricas: *Inteiro:* Representa os números sem casas decimais, positivos ou negativos. *Real:* Representam os números positivos ou negativos com casas decimais.

Lógico: Representam os valores lógicos: VERDADEIRO ou FALSO.

Há ainda, os **Tipos Construídos**. Com eles pode se organizar uma série de informações e tratá-la como uma única variável.

Observação: *Os valores numéricos literais não são compatíveis com operações matemáticas, pois são tratados apenas como símbolos. Para operações matemáticas deve – se usar as variáveis Inteiras ou Reais.*

As variáveis podem ainda ser classificadas de acordo com a forma como o dado estará armazenado:

Constante: O dado armazenado não é e nem pode ser alterado durante a execução do programa.

Variável Simples: São dados únicos que podem sofrer alteração de valor durante o decorrer do programa.

Variáveis Indexadas: Variáveis que armazenam vários dados diferentes do mesmo tipo. As variáveis indexadas podem ser:

Vetores: Variáveis que tem apenas um índice de posição do dado, ou uma dimensão. Equivale a posição do dado na coluna.

Matrizes: Variáveis que mantém dois índices, ou duas dimensões. Equivale a linha e coluna.

Para declaração de variáveis, segue – se sempre, independente da linguagem utilizada, o seguinte padrão:

- 1 – Ser coerente, utilizando nomes que ajudem a lembrar a utilidade da variável.
- 2 - Devem iniciar com letra.
- 3 – Devem conter apenas letra, algarismos e os símbolos _ e \$.
- 4 – Não deve conter espaço no nome.
- 5 – Não devem utilizar palavras reservadas.
- 6 – Observar as especificações da linguagem.

Operadores

Os operadores são utilizados para indicar qual a ação deve ser tomada, qual operação deve ser efetuada ou ainda para fazer uma comparação. Um exemplo disso são os sinais que utilizamos para indicar as operações fundamentais aritméticas; soma, subtração, multiplicação e divisão:

Operadores de Aritméticos

Símbolo	Explicação	Exemplo
++	Adiciona 1 ao valor atual da variável, muito utilizado em laços junto de contadores.	a++ ou ++a
Incremento		
--	Subtrai 1 ao valor atual da variável, muito utilizado em laços junto de contadores.	a-- ou --a
Decremento		
*	Multiplica os valores.	2 * 3
Multiplicação		
/	Divide os valores	2 / 3
Divisão		
^ ou **	Eleva um valor por outro.	2^3

Exponenciação		ou $2^{**}3$
mod ou %	Apresenta o resto de uma divisão. No caso do exemplo ao lado o valor apresentado será 1, já que $7 / 2 = 2$ e sobra "1"	7 % 3 ou 7 mod 3
Módulo		
+	Soma dois valores	2 + 2
Soma		
-	Subtrai um valor de outro	3 - 1
Subtração		

Operadores Relacionais

Operador	Explicação	Exemplo
>	Na comparação entre um e outro número, retorna <i>verdadeiro</i> se o primeiro é maior senão retorna <i>falso</i>	2 > 1
Maior		
>=	Na comparação entre um e outro número, retorna <i>verdadeiro</i> se o primeiro é maior ou igual ao segundo senão retorna <i>falso</i>	2 >= 2
Maior ou igual		
<	Na comparação entre um e outro número, retorna <i>verdadeiro</i> se o primeiro é menor senão retorna <i>falso</i>	2 < 3
Menor		
<=	Na comparação entre um e outro número, retorna <i>verdadeiro</i> se o primeiro é menor ou igual ao segundo senão retorna <i>falso</i>	2 <= 2
Menor ou igual		
= ou ==	Retorna <i>verdadeiro</i> somente se ambos os valores forem iguais	2 = 2
Igual		
<> ou !=	Retorna <i>verdadeiro</i> somente se ambos os valores forem diferentes	2 <> 5
Diferente		

Operador de Atribuição

Símbolo	Explicação	Exemplo
<- ou =	Indica a atribuição de um valor a uma variável. O Símbolo utilizado varia de acordo com a linguagem utilizada. Entretanto os mais utilizados são os apresentados ao lado.	Nome <- "José" ou Nome = "José"

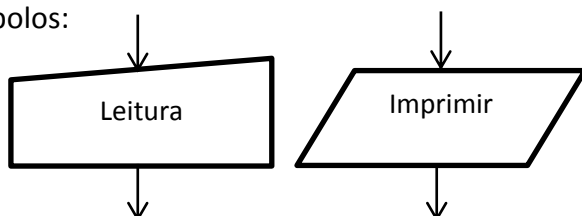
Operadores Lógicos

Símbolo	Explicação	Exemplo
& ou e	Duas condições devem ser verdadeiras para que retorne <i>verdadeiro</i> , caso contrário retornará <i>falso</i> .	10>5 & 1<2 ou 10>5 e 1<2
ou ou	Pelo menos uma condição deve ser verdadeira para que retorne <i>verdadeiro</i> , se ambas forem falsas retornará <i>falso</i> .	1>3 5=5 ou 1>3 ou 5=5
! ou nao	Nega o valor atual. No caso do exemplo ao lado, para x maior que 5 o retorno será <i>falso</i> , caso contrário <i>verdadeiro</i> .	não x > 5 ou !x > 5

Uso do fluxograma para entrada, leitura e atribuição.

Para as instruções **leia** e **escreva** serão utilizados respectivamente os

símbolos:

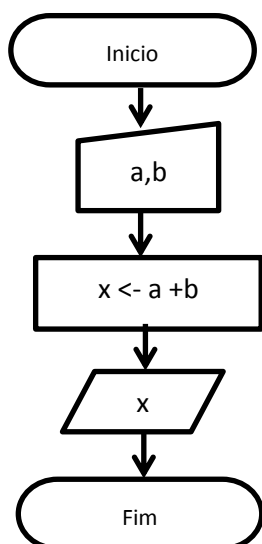


E o português estruturado:

Leia (<dado>)

Escreva(<informação>)

Exemplo: “Crie um programa que efetue a leitura de dois valores numéricos. Faça a operação de soma entre os dois valores e apresente o resultado obtido”. Construa o diagrama de blocos e depois o português estruturado.



Algoritmo:

- 1- Ler os dados;
- 2- Efetuar a soma e colocar o resultado em x;
- 3- Apresentar o valor

Português Estruturado:

Programa soma

var

x,a,b : inteiro

algoritmo

leia(a)

leia(b)

x <- a + b

escreva(x)

fimalgoritmo

Exercícios de Aprendizagem:

- 1) Indique com um X quais dos dados abaixo são do tipo Inteiro:
 - a. () 1000
 - b. () "0"
 - c. () "-900"
 - d. () .Verdadeiro.
 - e. () -456
 - f. () 34
 - g. () "Casa 8"
 - h. () 0
 - i. () .Falso.
 - j. () -1.56
- 2) Indique com um X quais dos dados abaixo são do tipo Real:
 - a. () -678
 - b. () "0.87"
 - c. () "-9.12"
 - d. () .Verdadeiro.
 - e. () -456
 - f. () -99.8
 - g. () "cinco"
 - h. () 45.8976
 - i. () .Falso.
 - j. () -1.56
- 3) Indique com um x quais dos dados abaixo são do tipo Literal.
 - a. () 678
 - b. () "0.87"
 - c. () "-9.12"
 - d. () "Verdadeiro"
 - e. () -456
 - f. () -99.8
 - g. () "cinco"
 - h. () 45.8976
 - i. () .Falso.
 - j. () 1.56

4) Desenvolver a lógica para um programa que efetue o cálculo da área de uma circunferência, apresentando a medida da área calculada.

5) Ler uma temperatura em graus Celsius e apresenta-la convertida em graus Fahrenheit. A fórmula de conversão é: $F \leftarrow (9 * C + 160) / 5$, sendo F a temperatura em Fahrenheit e C a temperatura em Celsius.

6) Ler uma temperatura em graus fahrenheit e apresenta-la convertida em graus Celsius. A fórmula de conversão é: **$C \leftarrow (F - 32) * (5/9)$** , sendo F a temperatura em Fahrenheit e C a temperatura em Celsius.

7) Calcular o valor do volume de uma lata de óleo, utilizando a fórmula: **volume** <- **3.14159* r^2 * altura**.

8) Efetuar o calculo da quantidade de litros de combustível gastos em uma viagem, utilizando-se um automóvel que faz 12km por litros. Para obter o calculo, o usuário deverá fornecer o tempo gasto e a velocidade média durante viagem. Desta forma, será possível obter a distancia com a fórmula **distancia <- tempo*velocidade**. Tendo o valor da distancia, basta calcular a quantidade de litros de combustível utilizada na viagem com a fórmula **litros_usados <- distancia/12**. O programa deverá apresentar os valores da velocidade média, tempo gasto na viagem, a distância percorrida e a quantidade de litros utilizada na viagem.

9) Efetuar o cálculo e a apresentação do valor de uma prestação em atraso, utilizando a formula: **$\text{prestação} \leftarrow \text{valor} + (\text{valor} * (\text{taxa} / 100) * \text{tempo})$** .

10) Ler dois valores para as variáveis a e b, e efetuar a troca dos valores de forma que a variável a passe a possuir o valor da variável b e a variável b passe a possuir o valor da variável a. Apresentar os valores trocados.

11) Construir um programa que efetue o cálculo do salário líquido de um professor. Para fazer este programa, você deverá possuir alguns dados, tais como: valor da hora aula, número de horas trabalhadas no mês e percentual de desconto do INSS. Em primeiro lugar, deve-se estabelecer qual será o seu salario bruto para efetuar o desconto e ter o valor do salario liquido.

12) Faça um programa que efetue a média de três notas de um aluno e mostre na tela.

Exercícios extra

1. Ler quatro valores numéricos inteiros e apresentar o resultado das adições e das multiplicações utilizando-se o conceito de propriedade distributiva para máxima combinação possível entre as quatro variáveis. Considerando-se o uso das variáveis a,b,c e d, deverá ser efetuada seis adições e seis multiplicações, ou seja, de forma geral, deverá ser combinada a variável a com a variável b, a variável a com a variável c, a variável a com a variável d. depois será necessário combinar a variável b com a variável c e a variável b com a variável d e por fim a variável c será combinada com a variável d.
2. Elaborar um programa que calcule e apresente o volume de uma caixa retangular, por meio da formula **volume <- comprimento * largura*altura**.
3. Efetuar a leitura de um número inteiro e apresentar o resultado do quadrado desse número.
4. Ler dois valores inteiros (variáveis a e b) e apresentar o resultado do quadrado da diferença do primeiro valor (variável a) pelo segundo valor (variável b).
5. Elaborar um programa que efetue a apresentação do valor da conversão em real(R\$) de um valor lido em dólar (US\$). O programa deverá solicitar o valor da cotação do dólar e também a quantidade de dólares disponíveis com o usuário.
6. Elaborar um programa que efetue a apresentação do valor da conversão em dólar (US\$) de um valor lido em real(R\$). O programa deverá solicitar o valor da cotação do dólar e também a quantidade de reais disponíveis com o usuário.
7. Elaborar um programa que efetue a leitura de três valores inteiros (representados pelas variáveis a,b e c) e apresente como resultado final o valor da soma dos quadrados dos três valores lidos.
8. Elaborar um programa que efetue a leitura de três valores inteiros (representados pelas variáveis a,b e c) e apresente como resultado final o valor do quadrado da soma dos três valores lidos.
9. Elaborar um programa de computador que efetue a leitura de quatro valores inteiros (variáveis a,b,c e d). Ao final o programa deve apresentar o resultado do produto (variável p) do primeiro com o terceiro valor, e o resultado da soma (variável s) do segundo com o quarto valor.
10. Ler o valor correspondente ao salario mensal (variável sm) de um trabalhador e também o valor do percentual de reajuste (variável pr) a ser atribuído. Apresentar o valor do novo salario (variável ns).
11. Elaborar um programa de computador que calcule e apresente o valor da área de uma circunferência (variável a). para tanto, o programa deve

solicitar o valor do raio (variável r) e fazer uso da formula de calculo: $a < -3.14159 * r^2$.

12. Em uma eleição sindical concorreram ao cargo de presidente três candidatos (a,b e c). Durante a apuração dos votos foram computados votos nulos e votos em branco, além dos votos válidos para cada candidato. Deve ser criado um programa de computador que efetue a leitura da quantidade de votos válidos para cada candidato, além de efetuar também a leitura da quantidade de votos nulos e votos em branco. Ao final o programa deve apresentar o numero total de eleitores, considerando votos válidos, nulos e em branco; o percentual correspondente de votos válidos em relação á quantidade de eleitores; o percentual correspondente de votos válidos do candidato a em relação á quantidade de eleitores; o percentual correspondente de votos válidos do candidato b em relação a quantidade de eleitores; o percentual correspondente de votos válidos do candidato c em relação a quantidade de eleitores; o percentual correspondente de votos nulos em relação a quantidade de eleitores; e por ultimo o percentual correspondente de votos em branco em relação a quantidade de eleitores.

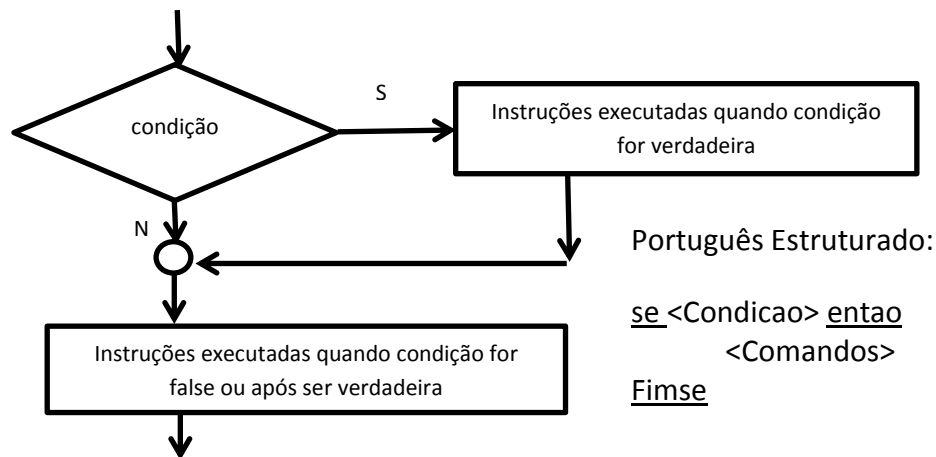
"A pior decisão é a indecisão." (Benjamin Franklin)

Estruturas de Controle – Condicionais

Condição é o desvio condicionado do fluxo das ações do algoritmo, ou seja, como o próprio nome diz é uma condição, uma pergunta que exige um tipo de resposta: *Sim* ou *Não*. É comparada a uma função definida por partes, onde cada parte é executada de acordo com a resposta atribuída à condição.

Desvios Condicionais Simples

Considera-se apenas um caminho a ser tomado. Caso a pergunta seja verdadeira (sim), uma ação é tomada. Não necessitando de ações caso ela seja falsa.

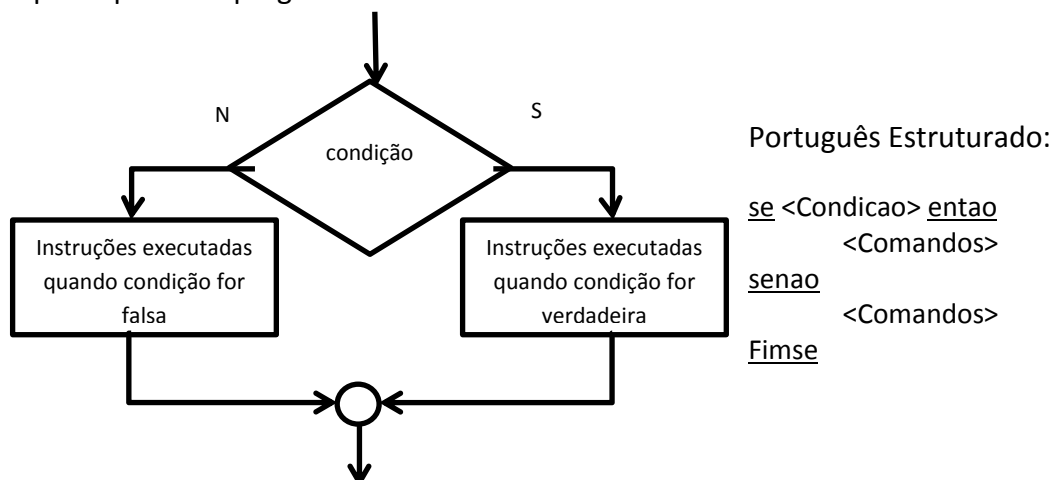


Português Estruturado:

```
se <Condicao> entao  
    <Comandos>  
Fimse
```

Condicionais Compostos

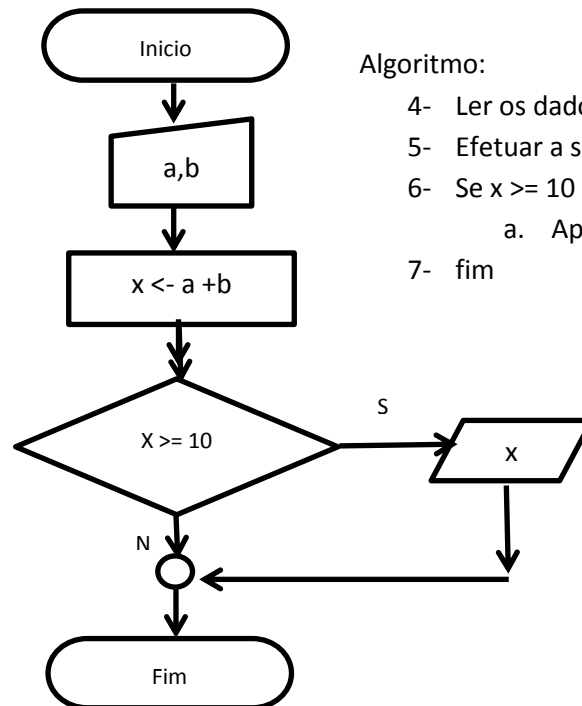
Consideram-se dois caminhos: aquele quando a pergunta é verdadeira e aquele quando a pergunta é falsa.



Português Estruturado:

```
se <Condicao> entao  
    <Comandos>  
senao  
    <Comandos>  
Fimse
```

Exemplo 1: “ler dois valores numéricos. Efetuar a adição e apresentar o seu resultado caso o valor somado seja maior que 10”.



Algoritmo:

- 4- Ler os dados;
- 5- Efetuar a soma e colocar o resultado em x;
- 6- Se $x \geq 10$
 - a. Apresentar o valor
- 7- fim

Português Estruturado:

Programa soma

var

x, a, b : inteiro

algoritmo

leia(a)

leia(b)

$x \leftarrow a + b$

se $x \geq 10$ **entao**
escreva(x)

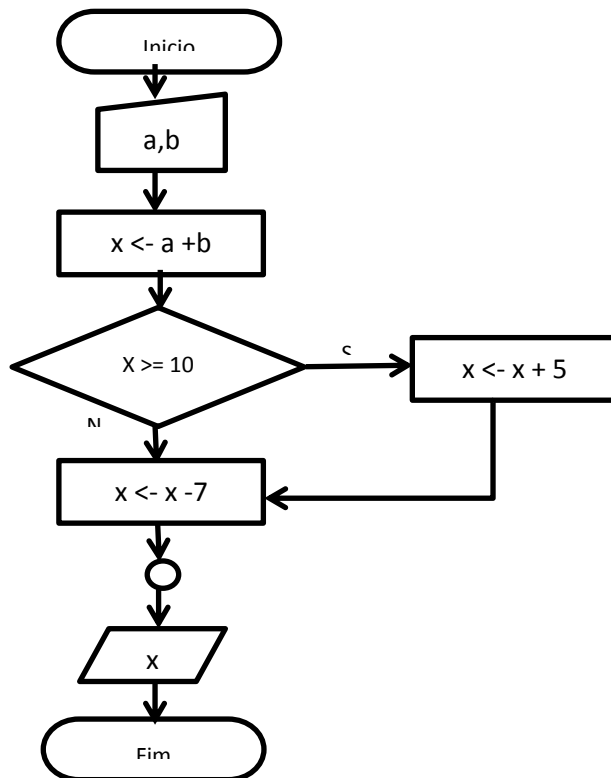
Fimse

fimalgoritmo

Exemplo 2: “Ler dois valores numéricos. Efetuar a adição e apresentar o seu resultado caso o valor somado seja maior que 10, deverá ser apresentado somando a ele mais 5; caso o valor somado não seja maior ou igual a 10, este deverá ser apresentado subtraindo 7”.

Algoritmo:

- 1- Ler os dados;
- 2- Efetuar a soma e colocar o resultado em x;
- 3- Se $x \geq 10$
 - a. Apresentar o valor somado de 5
- 4- Senão
 - a. Apresentar o valor subtraindo 7
- 5- fim



Português Estruturado:

Programa soma

var

x,a,b : inteiro

algoritmo

leia(a)

leia(b)

x <- a + b

se x >= 10 entao

x <- x+5

senão

x <- x-7

Fimse

escreva(x)

fimalgoritmo

Condicionais Encadeados

Existem casos em que é necessário estabelecer verificações de condições sucessivas, em que uma determinada ação poderá ser executada se um conjunto anterior de instruções ou condições for satisfeito. Sendo a ação executada, ela poderá ainda estabelecer novas condições. Isto significa utilizar uma condição dentro de outra. Este tipo de estrutura poderá possuir diversos níveis de condição, sendo chamadas de aninhamentos ou encadeamentos.

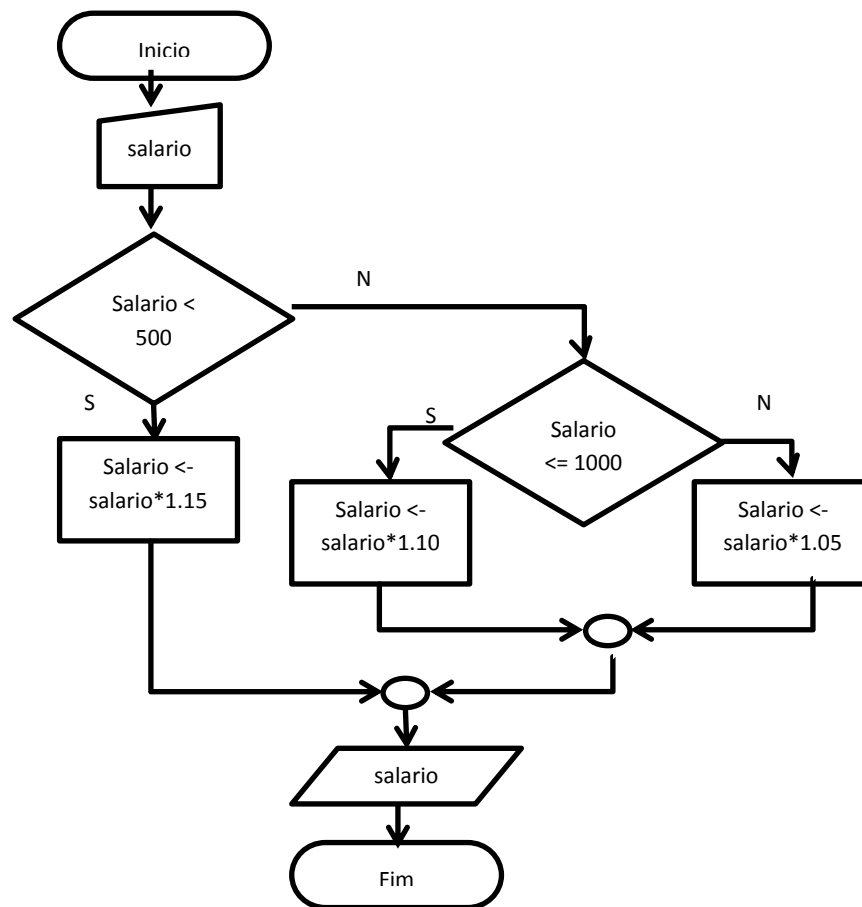
Exemplo: “Elaborar um programa que efetue o calculo do reajuste de salario de um funcionário. Considere que o funcionário deverá receber um reajuste de 15% caso seu salario seja menor que 500. Se o salario for mais ou igual a 500, mas menor ou igual a 1000, seu reajuste será de 10%; caso seja ainda maior que 1000, o reajuste deverá ser de 5%”.

Algoritmo:

Salario < 500, reajuste de 15%

Salario >= 500, mas <= 1000, reajuste de 10%

Salario > 1000, reajuste de 5%



Português Estruturado:

Programa reajuste

var

salario : real

algoritmo

leia(salario)

se salario < 500 entao

salario <- salario * 1.15

senão

se salario <= 1000 entao

salario <- salario * 1.10

senão

salario <- salario * 1.05

fimse

Fimse

escreva(salario)

fimalgoritmo

Operadores lógicos – uma revisão

Às vezes você precisará trabalhar com mais de uma condição ao mesmo tempo na mesma instrução **se**. Para esses casos, utilizamos operadores booleanos ou **lógicos**. Eles são: **e**, **ou** e **não**. Em muitos casos, eles evitam a utilização de instruções encadeadas, e são os mesmo operadores que você viu na Lógica Formal, no início desse material.

O operador **e** é utilizado quando dois ou mais relacionamentos lógicos de uma determinada condição necessitam ser verdadeiros. Exemplo de português estruturado:

Se (numero >= 20) e (numero <= 90) então

Escreva (“O numero está na faixa de 20 a 90”)

Senão

Escreva(“O numero está fora da faixa de 20 a 90”)

Fimse

O operador **ou** é utilizado quando pelo menos um dos relacionamentos lógicos de uma condição necessita ser verdadeiro. Apenas um basta, mas se ambos forem, é lucro.

Se (sexo = “masculino”) ou (sexo= “feminino”) então

Escreva (“Seu sexo é valido”)

Senão

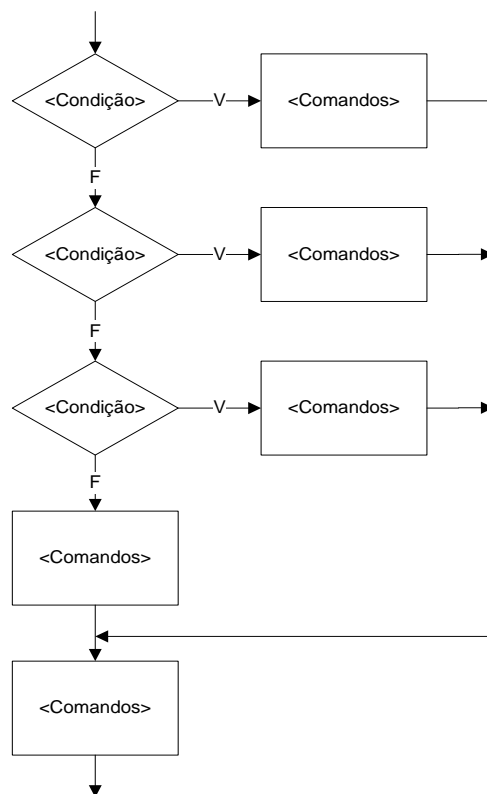
Escreva(“Seu sexo é invalido”)

Fimse

O operador **não** é utilizado quando houver a necessidade de estabelecer a inversão do resultado lógico de uma determinada condição. Se a condição for verdadeiro, será considerada falsa. Se a condição for falsa, será considerada verdadeira.

Condicionais Seletivos (Escolha)

Quando existe a necessidade de verificar valores de uma variável, normalmente é utilizado condições encadeadas, porém dependendo da situação, essa forma torna se trabalhosa. A estrutura escolha é mais prática para esses casos. Nela, uma variável é escolhida e para cada **caso**, uma ação é feita. Um caso a parte, é utilizado para caso nenhum dos casos anteriores tenha acontecida, esse caso chama se **outrocaso**.



Português estruturado:

Escolha (variável)

Caso valor

//instruções

Caso valor2

//instruções

Caso valor3

//instruções

Outrocaso

//instruções

fimescolha

Exercícios de Aprendizagem:

1) Determine o resultado lógico das expressões mencionadas, assinalando se são verdadeiras ou falsas. Considere para as respostas os seguintes valores: $x=1$, $a=3$, $b=5$, $c=8$ e $d=7$.

- a. Não ($x>3$)
- b. ($x<1$) e não($b>d$)
- c. Não($d<0$) e ($c>5$)
- d. Não($x>3$) ou($c<7$)
- e. ($a>b$) ou ($c>b$)
- f. ($x\geq 2$)
- g. ($x<1$) e ($b\geq d$)
- h. ($d<0$) ou ($c>5$)
- i. Não ($d>3$) ou não($b<7$)
- j. ($a>b$) ou não($c>b$)

2) Indique a saída dos trechos de programa em português estruturado, mostrado abaixo. Para as saídas considere os seguintes valores: $a=2$, $b=3$, $c=5$ e $d=9$. Não é necessário calcular os valores de x , marque apenas a formula que será executada.

- a. Se não($d>5$) então
 $X \leftarrow (a+b) * d$
Senão
 $X \leftarrow (a-b) / c$
Fimse
- b. Se ($a>2$) e ($b<7$) então
 $X \leftarrow (a+2) * (b-2)$
Senão
 $X \leftarrow (a+b)/d * (c+d)$
Fimse

- c. Se $(a=2)$ ou $(b<7)$ então
 $X \leftarrow (a+2) * (b-2)$
 Senão
 $X \leftarrow (a+b) / d * (c+d)$
 Fimse
- d. Se $(a>2)$ ou não $(b<7)$ entao
 $X \leftarrow a+b-2$
 Senão
 $X \leftarrow a- b$
 Fimse
- e. Se não $(a>2)$ ou não $(b<7)$ então
 $X \leftarrow a+b$
 Senão
 $X \leftarrow a/b$
 Fimse
- f. Se $(a>3)$ e não $(b<5)$ então
 $X \leftarrow a+d$
 Senão
 $X \leftarrow d/b$
 Fimse
- g. Se $(c \geq 2)$ e $(b \leq 7)$ entao
 $X \leftarrow (a+d) / 2$
 Senão
 $X \leftarrow d*c$
 Fimse
- h. Se $(a \geq 2)$ ou $(c \leq 1)$ então
 $X \leftarrow (a+d) / 2$
 Senão
 $X \leftarrow d*c$
 Fimse

3) Ler dois valores numéricos inteiros e apresentar o resultado da diferença do maior valor pelo menor valor.

4) Ler um valor numérico inteiro positivo ou negativo e apresentar o valor lido como sendo um valor positivo, ou seja, se o valor lido for menor ou igual a zero, ele deve ser multiplicado por -1.

5) Ler os valores de quatro notas escolares de um aluno. Calcular a média aritmética e apresentar a mensagem “aprovado” se a media obtida for maior ou igual a 7; caso contrário, o programa deve solicitar a nota de exame do aluno e calcular uma nova média aritmética entre a nota de exame e a primeira média aritmética. Se o valor da nova média for maior ou igual a cinco, apresentar a mensagem “aprovado em exame”; caso contrário, apresentar a mensagem “reprovado”. Informar junto de cada mensagem o valor da média obtida.

6) Desenvolver um programa que efetue a leitura de um valor numérico inteiro e apresente-o caso este valor seja divisível por 4 e 5. Não sendo divisível por 4 e 5 o programa deverá apresentar a mensagem “não é divisível por 4 e 5”.

7) Ler três valores numéricos (a,b e c) e efetuar o calculo da equação completa de segundo grau, utilizando a formula de Baskara (considerar todas as possíveis condições para $\Delta < 0$, $\Delta > 0$ e $\Delta = 0$). Lembre-se de que é completa a equação de segundo grau que possui simultaneamente as variáveis a, b e c diferentes de zero.

8) Ler três valores para os lados de um triângulo, considerando lados como: A, B e C. verificar se os lados fornecidos formam realmente um triângulo, e se for esta condição verdadeira, deverá ser indicado qual tipo de triângulo foi formado: isósceles, escaleno ou equilátero.

9) Ler quatro valores numéricos inteiros e apresentar os valores que são divisíveis por 2 **e** 3.

10) Ler quatro valores numéricos inteiros e apresentar os valores que são divisíveis por 2 **ou** 3.

11) Ler um valor numérico inteiro que esteja na faixa de valores de 1 até 9. O programa deve apresentar a mensagem: “o valor está na faixa permitida”, caso o valor informado esteja entre 1 e 9. Se o valor estiver fora da faixa, o programa deve apresentar a mensagem “o valor está fora da faixa permitida”.

12) O cardápio de uma lancharia é o seguinte:

Especificação	Código	Preço
Cachorro quente	100	2,00
Bauru simples	101	2,50
Americano	102	2,80
Hambúrguer	103	1,90
Cheeseburger	104	2,00
Refrigerante	105	1,50
Água	106	1,00

Escrever um algoritmo que leia o código dos itens pedidos, as quantidades e calcule o valor a ser pago.

13) Elabore um algoritmo que, dada a idade de um nadador, classifique-o em uma das seguintes categorias:

- a. infantil A = 5 - 7 anos
- b. infantil B = 8-10 anos
- c. juvenil A = 11-13 anos
- d. juvenil B = 14-17 anos
- e. adulto = maiores de 18 anos

14) Ler um valor numérico inteiro qualquer e fazer a sua apresentação caso o valor não seja maior que três. Utilize apenas o operador lógico **não** para a solução deste problema.

15) Ler o nome e o sexo de uma pessoa e apresentar como saída uma das seguintes mensagens: "ilmo. Sr.", caso seja informado o sexo como masculino, ou "ilma. Sra.", caso seja informado o sexo como feminino. Apresentar também junto de cada mensagem de saudação o nome previamente informado.

16) Tendo como dados de entrada a altura e o sexo de uma pessoa construa um algoritmo que calcule seu peso ideal, utilizando as seguintes fórmulas:

- para homens: $(72.7 * \text{altura}) - 58$
- para mulheres: $(62.1 * \text{altura}) - 44.7$

17) Faça um algoritmo que leia um nº inteiro e mostre uma mensagem indicando se este número é par ou ímpar, e se é positivo ou negativo.

18) Uma empresa concederá um aumento de salário aos seus funcionários, variável de acordo com o cargo, conforme a tabela abaixo. Faça um algoritmo que leia o salário e o cargo de um funcionário e calcule o novo salário. Se o cargo do funcionário não estiver na tabela, ele deverá, então, receber 40% de aumento. Mostre o salário antigo, o novo salário e a diferença.

Código	Cargo	Percentual
101	Gerente	10%
102	Engenheiro	20%
103	Técnico	30%

19) Ler cinco valores numéricos inteiros, identificar e apresentar o maior e o menor entre eles. Não executa a ordenação dos valores.

20) Ler um valor numérico inteiro e apresentar uma mensagem informando se o valor numérico informado é par ou ímpar.

21) Ler três valores e apresentá-los dispostos em ordem crescente.

"Há repetição em todos os lugares, e nada é encontrado apenas uma vez no mundo."

Goethe

Estruturas de Controle – Repetições

Repetição é um conjunto de instruções que se repetem enquanto uma determinada condição for verdadeira.

Imagine que você entrou num restaurante, comeu, porem esqueceu o cartão, ou seja, você não tem dinheiro. O que acontece? Você vai lavar pratos ate que complete o dinheiro que deve.

Exemplo: supondo que você tenha gastado 20 reais. E o dinheiro por prato limpo é 1 real, você começa com zero pratos e vai lavando um por um ate que você consiga lavar 20 pratos e pagar sua divida.

Ou seja, vamos analisar: você tem um *valor de partida* (zero pratos), um *destino/meta* a ser cumprido (20 pratos), onde se tem que repetir o processamento (lavar os pratos) ate cumpri-lo e existe algo que chamamos de passos, ou seja, de quanto em quanto o algoritmo ira contar quantas vezes será repetida ação, nesse caso será lavado um prato por vez, por tanto o passo é um: 0 ..1..2..3. Já que será ganho 1 real por prato, dando um total de 20, caso fosse 2 reais, o passo seria dois: 0..2..4..6 sendo um total de 10 pratos. E assim por diante.

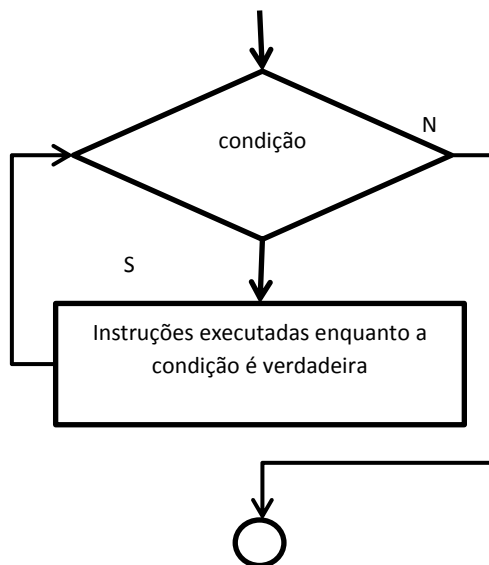
A repetição é basicamente isso... Imagine q ela é uma grande bolha que prende o programa e faz com q ele faça as mesmas ações ate que a condição que ela impôs seja satisfeita.

A principal vantagem desse recurso é que o programa passa a ter um tamanho menor, podendo sua amplitude de processamento ser aumentada sem alterar o tamanho do código de programação.

“Enquanto” – Teste lógico no inicio do looping

É uma estrutura que efetua um teste lógico no inicio de um looping, verificando se é permitido executar o trecho de instruções subordinado a esse looping. A estrutura em questão é denominada de **enquanto**.

A estrutura enquanto tem seu funcionamento controlado por decisão. Sendo assim, poderá executar um determinado conjunto de instruções enquanto a condição verificada for verdadeira. No momento em que esta condição se torna falsa, as instruções contidas no looping são ignoradas.



Português estruturado:

Enquanto (condição) faça

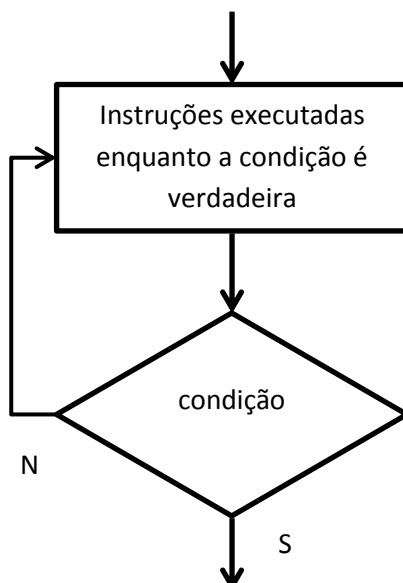
//instruções

fimenquanto

“Repita até” – Teste lógico no final do looping

Caracteriza-se por uma estrutura que efetua um teste lógico no fim de um looping. Esta estrutura é parecida com a **enquanto**. A estrutura em questão é denominada de **repita**.

A estrutura repita também tem seu comportamento controlado por decisão. Porém, irá efetuar a execução de um conjunto de instruções **pelo menos uma vez antes de verificar a validade da condição estabelecida**. Diferente da estrutura **enquanto** que executa somente em conjunto de instruções, enquanto a condição é verdadeira.



Português estruturado:

Repita

//instruções

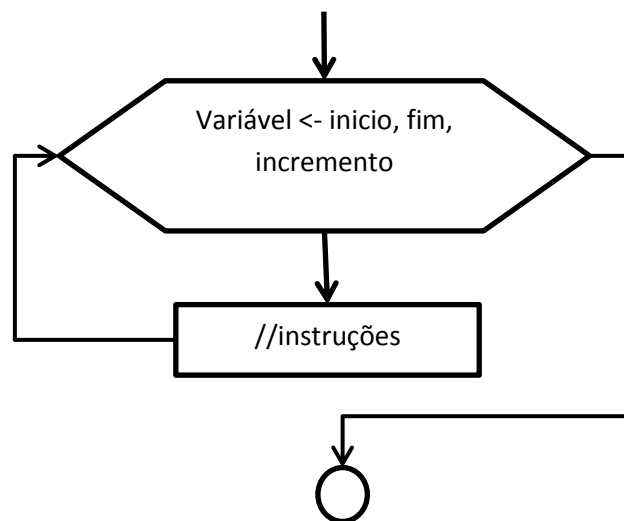
ate (condição)

Dessa forma repita tem seu funcionamento em sentido contrário a enquanto, pois sempre irá processar um conjunto de instruções no mínimo uma vez ate que a condição se torne **verdadeira**. Para a estrutura repita um conjunto de instruções é executado enquanto a condição se mantem falsa e ate que ela seja verdadeira.

“Para” – Teste lógico controlado por variável – contador

As repetições muitas vezes são controlados por um “contador” que é verificado na condição. Sempre que seu valor não alcança uma condição o looping continua. Esse contador inicia-se em um valor inicial, passa pelo looping e no final o “estoura”. Esse é o principio básico das repetições controladas por contador.

Existe uma possibilidade de facilitar o uso de contadores finitos por meio da instrução **para**. A instrução para tem o seu funcionamento controlado por uma variável denominadas contador. Sendo assim, poderá executar um determinado conjunto de instruções em determinado numero de vezes.



para <Variável> de <Condicao_Inicial> ate <Condicao_Final> passo <Incremento> faca
 <Comandos>
fimpara

Repetição Controlada por sentinela

Das estruturas anteriores podemos utilizar a repita para uma técnica chamada “repetição controlada por sentinela”. Nela o contador utilizado, assume um valor inicial, e conforme o usuário entra com um dado o looping continua a se repetir até que o usuário digite a condição de parada. Por exemplo: imagine que estamos fazendo várias transações bancárias e o programa mostra na tela um menu de operações. O usuário tem a opção de entrar com o numero da operação desejada ou um numero não válido para terminar a operação. Caso ele entre com um numero válido (uma opção), o programa executa normalmente. E no final, o menu é mostrado novamente, esse processo continua ate que o usuário digite o código para a opção de sair.

Ou seja, nosso contador (sentinela) ficou “vigiando” até aparecer o código de saída, uma vez que ele apareceu, ele termina o looping.

As três estruturas não são separadas, você pode utiliza-las como bem entender, para adaptar a resolução do programa, o melhor possível. É permitido misturara-las, como condição dentro de repetição, repetição dentro de condição. E nem se fala à atribuição que é basicamente 90% do programa. O segredo é pensar, e descobrir, treinar e testar para ver se sua lógica está correta.

Exercícios de Aprendizagem:

- 1) Conte de 0 até 100.

- 2) Apresentar os quadrados dos números inteiros de 15 a 200.

3) Mostre todos os números pares entre 0 e n. (Peça ao usuário o valor de n)

- 4) Mostre todos os números ímpares entre 0 e n. (Peça ao usuário o valor de n)

5) Apresentar os resultados de uma tabuada de um numero qualquer, a qual deve ser impressa no seguinte formato:

Considerando como exemplo o fornecimento do numero 2 para o numero qualquer:

$$2 \times 1 = 2$$

$$2 \times 2 = 4$$

$$2 \times 3 = 6$$

...

$$2 \times 10 = 20$$

6) Apresentar o total da soma obtida dos cem primeiros números inteiros.

7) Elaborar um programa que apresente no final o somatório dos valores pares existentes na faixa de 1 até n.

8) Apresentar os resultados da potencia de b elevado a e . (Peça o valor da base (b) e do expoente (e)).

9) Elabore um programa que mostre o fatorial dos números de 1 até n. (peça o valor de n ao usuário)

10) Elaborar um programa que efetue a leitura sucessiva de valores numéricos e apresente no final o total do somatório, a media e o total de valores lidos. O programa deve fazer as leituras dos valores enquanto o usuário estiver fornecendo valores positivos. Ou seja, o programa deve parar quando o usuário fornecer um valor negativo.

11) Pedir a leitura de um valor para a variável x , multiplicar esse valor por 3, implicando-o á variável de resposta R , e apresentar o valor obtido, repetindo esta sequencia por cinco vezes.

12) Apresentar todos os números divisíveis por 4 que sejam menores que 200.

13) Elaborar um programa que efetue a leitura de 10 valores numéricos e apresente no final o total do somatório e a media dos valores lidos.

14) Elaborar um programa que efetue a leitura de 15 valores numéricos inteiros e no final apresente o total do somatório da fatorial de cada valor lido.

15) Elaborar um programa que apresente os resultados da soma e da média aritmética dos valores pares situados na faixa de 50 a 70.

16) Elaborar um programa que apresente os resultados da soma e da média aritmética dos valores impares situados na faixa de 50 a 70.

17) Elaborar um programa que possibilite calcular a área total de uma residência (sala, banheiro, quartos, área de serviço, quintal, garagem, etc.). o programa deve solicitar a entrada do nome, a largura e o comprimento de um determinado cômodo. Em seguida, deve apresentar a área do cômodo lido e também uma mensagem solicitando do usuário à confirmação de continuar calculando novos cômodos. Caso o usuário responda “não”, o programa deve apresentar o valor total acumulado da área residencial.

18) Elaborar um programa que efetue o calculo da fatorial do numero 5, 5!.

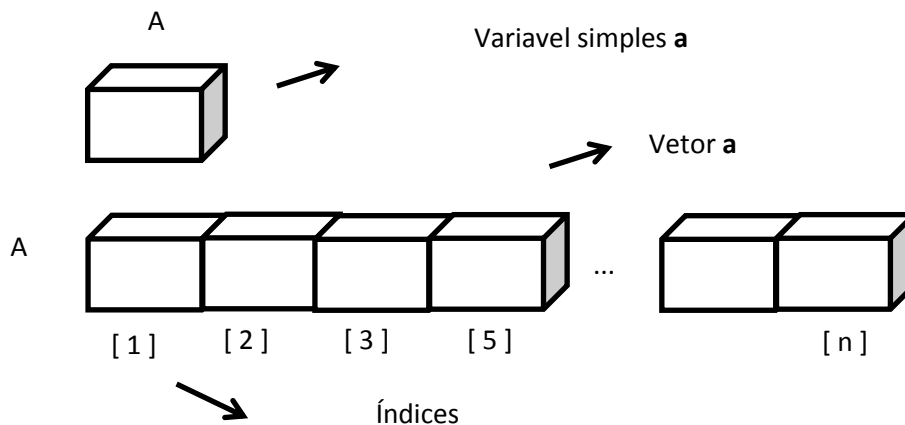
19) Elaborar um programa que efetue a leitura de valores positivos inteiros até que um valor negativo seja informado. Ao final devem ser apresentados o maior e o menor valor informados pelo usuário.

Estruturas de Dados Simples – Vetores

Ate agora utilizamos variáveis simples, que armazenavam um valor qualquer de um tipo predeterminado. Muitas vezes precisaremos de um conjunto de variáveis para armazenar muitas informações de um mesmo tipo. Como em muitos casos não é possível saber exatamente a quantidade dessas informações, não podemos criar um numero certo de variáveis. Para isso vamos aprender uma técnica de programação que permitira trabalhar com o agrupamento de várias informações dentro de uma mesma variável, é claro que sempre obedecera ao mesmo tipo de dado.

A utilização deste tipo de estrutura de dados recebe diversos nomes, como: variáveis indexadas, variáveis compostas, variáveis subscritas, arranjos, vetores, matrizes, tabelas de memoria ou arrays. Para nós utilizaremos vetores.

Uma variável simples é apenas um pequeno bloco na memoria. Uma variável indexada, é um grande conjunto desses pequenos blocos, obedecendo a um mesmo tipo e atendendo a um mesmo nome, sendo diferenciados apenas por um numero que dita sua ordem/posição na estrutura, chamado de índice.

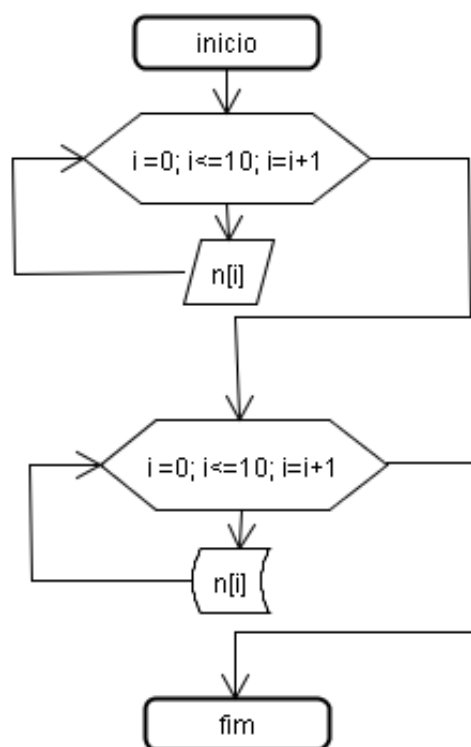


A utilização dessas variáveis segue os mesmo princípios das variáveis simples. Com uma adição: é preciso definir qual “casinha” do vetor você se refere. Desse modo:

Vetor [indice] = valor, Valor = vetor [indice], Soma = vetor [indice] + valor, ... e assim por diante, ou seja, não muda quase nada, você só acrescenta o valor do índice.

Exercícios de aprendizagem

1) Elabore um programa que leia 5 números e coloque em um vetor chamado n. apresente esses 5 números.



2) Desenvolver um programa que efetue a leitura de dez elementos de um vetor A. Construir um vetor b de mesmo tipo, observando a seguinte lei de formação: se o valor de índice for par, o valor deverá ser multiplicado por 5, sendo ímpar, deverá ser somado com 5. Ao final mostrar o conteúdo do vetor B.

3) Ler 8 elementos de um vetor A. Construir um vetor B de mesma dimensão com os elementos do vetor A multiplicados por 3. O elemento $b[1]$ deverá ser implicado pelo elemento $a[1]*3$, o elemento $b[2]$ deverá ser implicado pelo elemento $a[2]*3$ e assim por diante, até 8. Apresente o vetor b.

4) Ler dois vetores A e B com 5 elementos. Construir um vetor C, onde cada elemento de C é a subtração do elemento correspondente de A com B. Apresentar a matriz C.

5) Ler 5 elementos de um vetor a. construir um vetor B de mesmo tipo, observando a seguinte lei de formação: “todo elemento de B deverá ser o quadrado do elemento de A correspondente”. Apresentar os vetores a e b.

6) Desenvolver um programa que efetue a leitura de cinco elementos de um vetor A. No final, apresente o total da soma de todos os elementos que sejam ímpares.

7) Ler 5 elementos inteiros para o vetor A . Construir um vetor B de mesmo tipo e dimensão, observando a seguinte lei de formação: “todo elemento de A que for impar deverá ser multiplicado por 2; caso contrario, o elemento de A deverá permanecer constante”. Apresente a matriz B.

8) Ler 5 elementos reais para um vetor A. construir um vetor B de mesmo tipo e dimensão, observando a seguinte lei de formação: “todo elemento de A que possuir índice par deverá ter seu elemento dividido por 2; caso contrario, o elemento de A deverá ser multiplicado por 1.5”. Apresentar o vetor B.

9) Ler um vetor A com 10 elementos. Construir uma matriz B de mesmo tipo, sendo que cada elemento da matriz B seja o fatorial do elemento correspondente da matriz A. Apresentar as matrizes A e B;

10) Ler um vetor A e B com 5 elementos cada. Construir um vetor C, sendo esta a junção dos dois outros vetores. Desta forma, C deverá ter o dobro de elementos, ou seja, 10. Apresentar a matriz c.

11) Ler dois vetores sendo A com 10 elementos e b com 15 elementos. Construir um vetor C, sendo este a junção dos dois outros vetores. Desta forma, C deverá ter a capacidade de armazenar 25 elementos. Apresentar a matriz c.

12) Ler elementos de um vetor A e construir um vetor B de mesma dimensão com os mesmos elementos do vetor A, sendo que deverá estar invertidos. Ou seja, o primeiro elemento de A passa a ser o ultimo de b, o segundo elemento de a passa a ser o penúltimo de b e assim por diante. Apresentar os vetores a e b.

13) Ler um vetor A com 10 elementos. Construir um vetor B do mesmo tipo, sendo que cada elemento de b seja o somatório do elemento correspondente da matriz A. se o valor do elemento de a[1] for 5, b[1] deverá ser 15 (1+2+3+4+5) e assim por diante. Apresentar o vetor b.

14) Ler um vetor A com 10 elementos positivos. Construir um vetor b de mesmo tipo e dimensão, em que cada elemento do vetor b deverá ser o valor negativo do elemento correspondente do vetor A. desta forma, se em A[1] estiver armazenado o elemento 8, deverá estar em b[1] o valor -8, e assim por diante. Apresentar os elementos do vetor B.

15) Ler um vetor a com 10 elementos. Construir um veto B do mesmo tipo sendo que cada elemento de b deverá ser a metade de cada elemento de a. apresentar os elementos das matrizes A e B.

16) Elaborar um programa que efetue o calculo de uma tabuada de um numero qualquer e armazene os resultados em um vetor A de 10 elementos. Apresentar os valores armazenados na matriz.

Estruturas de Dados Simples – Matrizes

Conforme vimos no modulo anterior, vetores são conjuntos de variáveis simples com o mesmo nome, mesmo tipo que são determinados por um índice.

Existem casos que é necessário uma estrutura mais evoluída, chamada de matriz. Ela é um conjunto de vetores, também chamada de matriz bidimensional. Iremos ver pouco sobre ela, uma vez que vetores são mais importantes e usados.

As matrizes diferentemente dos vetores tem um índice composto por **linha e coluna**. Ao se referir a uma “casinha” da matriz é necessário passar esses dois valores.

Um importante aspecto importante a ser considerado é que na manipulação de uma matriz tipo vetor é utilizada uma única instrução de looping. No caso de matrizes com mais dimensões, deverá ser utilizado o numero de loopings relativo ao tamanho de sua dimensão. Desta forma, uma matriz de duas dimensões deverá ser controlada com dois loopings, de três dimensões deverá ser controlado por três loopings e assim por diante.

A		1	2	3	4	5
1		A[1,1]				
2						
3						
4					A[4,4]	
5			A[5,2]			
6						

do mesmo modo que se manipula um vetor, porem, agora com um índice a mais:

`matriz[linha,coluna] = valor, valor = matriz[linha,coluna],`

Faça um programa que leia uma matriz de 3 por 3 e apresente seus dados.

Programação Modular – Sub-rotinas e Procedimentos

No geral, problemas complexos exigem algoritmos complexos. Mas sempre é possível dividir um problema grande em problemas menores. Desta forma, cada parte menor tem um algoritmo mais simples, e é esse trecho menor que é chamado de sub-rotina. Uma sub-rotina é na verdade um programa, e sendo um programa poderá efetuar diversas operações computacionais (entrada, processamento e saída) e deverá ser tratada como foram os programas projetados até este momento.

As sub-rotinas são utilizadas na divisão de algoritmos complexos, permitindo assim possuir a modularização de um determinado problema, considerado grande e de difícil solução.

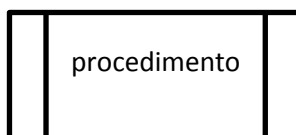
Ao trabalhar com esta técnica, pode-se deparar com a necessidade de dividir uma sub-rotina em outras quantas foram necessárias, buscando uma solução mais simples de uma parte do problema maior. O processo de dividir sub-rotinas em outras é denominado *método de refinamento sucessivo*.

Outro detalhe a ser considerado é que muitas vezes existem em um programa trechos de códigos que são repetidos várias vezes. Esses trechos poderão ser utilizados como sub-rotinas, proporcionando um programa menor e mais fácil de ser alterado num futuro próximo.

Iremos trabalhar com um tipo de sub-rotina conhecido como procedimento. Existe outro tipo conhecido como função que será visto mais a frente. Existe uma diferença entre eles mais o conceito é o mesmo. O importante no uso prático destes dois tipos de sub-rotina é distinguir as diferenças entre elas e como utiliza-las no momento mais adequado.

Um procedimento é um bloco de programa contendo início e fim e será identificado por um nome, por meio do qual será referenciado em qualquer parte do programa principal ou do programa chamador da rotina. Quando uma sub-rotina é chamada por um programa, ela é executada e ao seu termino o controle.

O procedimento nada mais é que um programa qualquer escrito do mesmo modo que vimos ao longo do curso. Para representar no diagrama de blocos utilizamos o símbolo a seguir.



Apenas iremos ver o conceito dessa técnica. Ao se iniciar em uma linguagem de programação, essa já utilizará essa técnica de um modo bem mais visível.

Estruturas em pseudocódigo

Conforme vimos anteriormente, apresentamos agora todas as estruturas básicas na forma de pseudocódigo, é interessante que se faça uma comparação entre as estruturas em fluxograma e o apresentado em Português Estruturado.

1 – Estrutura de decisão Simples

```
se <Condicao> entao
    <Comandos>
Fimse
```

3 – Estrutura de decisão Encadeada

```
se <Condicao> entao
    se <Condicao> entao
        <Comandos>
    senao
        <Comandos>
    fimse
senao
    <Comandos>
Fimse
```

4 – Estrutura de decisão Escolha

```
escolha <Variável>
    caso1 <Condicao>
        <Comandos>
    caso2 <Condicao>
        <Comandos>
    caso3 <Condicao>
        <Comandos>
    outrocaso
        <Comandos>
Fimescolha
```

2 – Estrutura de decisão Composta

```
se <Condicao> entao
    <Comandos>
senao
    <Comandos>
Fimse
```

Entre as variações possíveis temos

```
se <Condicao> entao
    se <Condicao> entao
        <Comandos>
    senao
        <Comandos>
    fimse
fimse
```

Laço se em condição FALSA:

```
se <Condicao> entao
    <Comandos>
senao
    se <Condicao> entao
        <Comandos>
    senao
        se <Condicao> entao
            <Comandos>
        senao
            <Comandos>
        fimse
    fimse
fimse
```

5 – Laço Enquanto (verifica e executa)

```
enquanto <Condicao> faca
    <Comandos>
Fimenquanto
```

6 – Laço Repita (Executa e verifica)

repita

<Comandos>

ate <Condicao>

* Neste caso o laço é executado até que a condição se torne VERDADEIRA.

Observe a condição (depende da linguagem)

repita

<Comandos>

enquanto <Condicao>

* Neste caso o laço é executado enquanto a condição for VERDADEIRA e para quando for FALSA.

7 – Laço Para (numero fixo de iterações)

para <Variável> de <Condicao_Inicial> ate <Condicao_Final> [passo <Incremento>] faca
<Comandos>

fimpara

* A parte apresentada entre colchetes [] não é obrigatória, entretanto sua não presença implica em que o incremento será padrão, de 1 em 1. Em qualquer outro caso, onde o incremento deve ser maior que 1, ou ainda, quando se trata de um decremento, a parte apresentada em colchetes passa a ser obrigatória.

Exemplo de algoritmo em pseudocódigo para ler três valores e apresentar

a média entre eles:

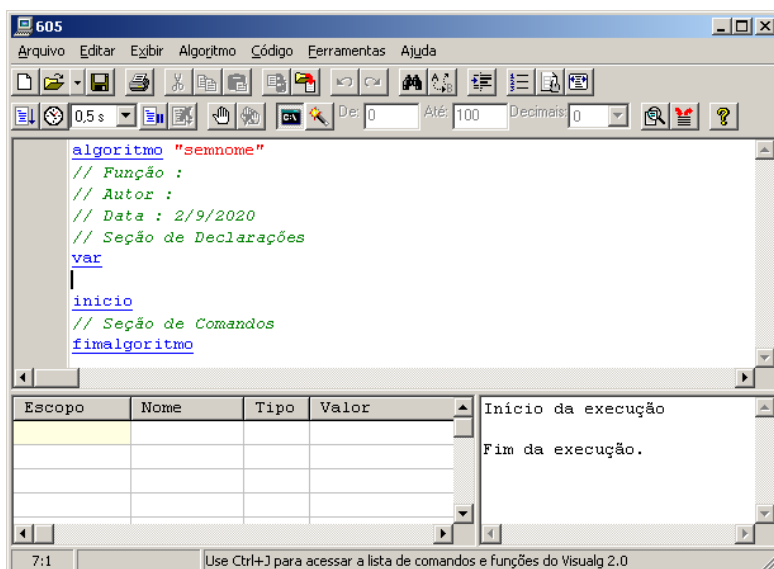
<p>algoritmo "Media"</p> <p>// Declaração de variáveis</p> <p>var</p> <p>Notas: vetor [1..3] de inteiro</p> <p>Media: real</p> <p>Inicio</p> <p>escreval("Entre com três valores inteiros")</p> <p>Leia (Notas[1])</p> <p>Leia (Notas[2])</p> <p>Leia (Notas[3])</p> <p>Media<-(Notas[1]+ Notas[2]+ Notas[3])/3</p> <p>Escreval (Media)</p> <p>FimAlgoritmo</p>	<p>algoritmo "Média"</p> <p>//Declaração de variaveis</p> <p>var</p> <p>Notas: vetor [1..3] de inteiro</p> <p>Media: real</p> <p>Cont :inteiro</p> <p>Inicio</p> <p>Cont <- 3</p> <p>Escreval("Entre com três valores inteiros")</p> <p>Enquanto (Cont>0)faca</p> <p> Leia (Notas[Cont])</p> <p> Cont <- Cont - 1</p> <p>FimEnquanto</p> <p>Media <- (Notas[1]+ Notas[2]+ Notas[3])/3</p> <p>Escreval (Media)</p> <p>FimAlgoritmo</p>
--	---

Mini manual do Visualg

O Software Visualg é um interpretador de português estruturado que torna extremamente fácil a compreensão e o desenvolvimento da aprendizagem de lógica e linguagem de programação. É um ambiente simples, próprio para o desenvolvimento do processo de ensino e aprendizagem, entretanto muito bem estruturado, fornecendo ferramentas suficientes para o desenvolvimento de pseudo – códigos que implementam algoritmos reais.

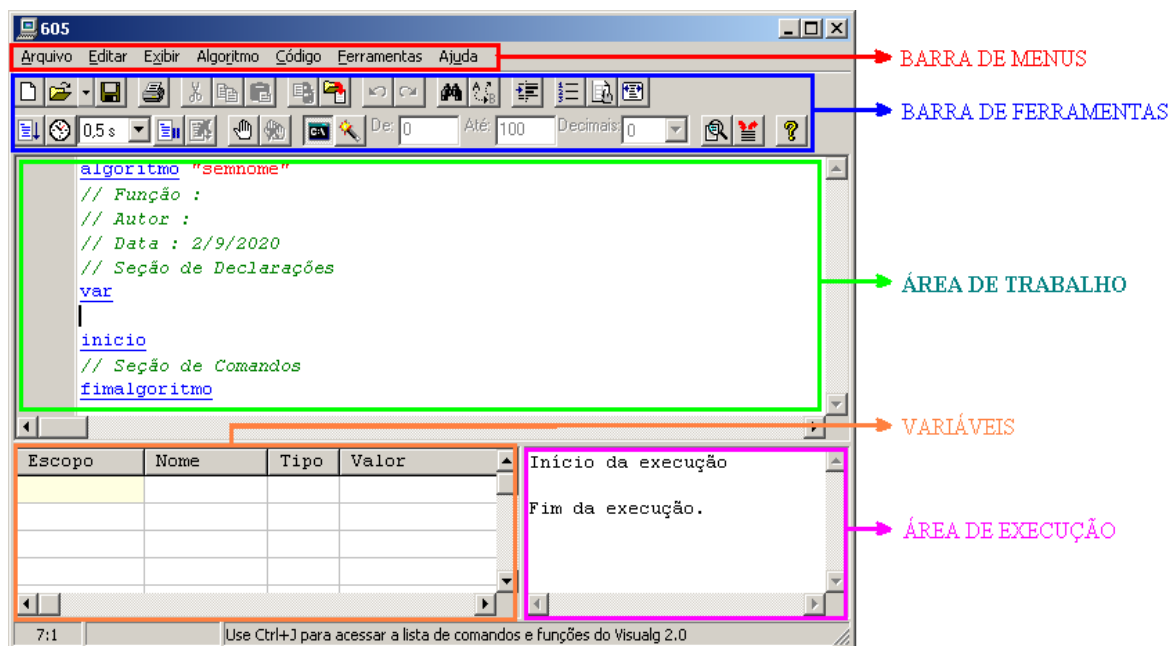
Os desenvolvedores do Software buscaram simplificar a tarefa de desenvolvimento, não dando tanta importância a detalhes como a capitalização de fontes (Diferença entre maiúscula e minúscula) e excluindo todas as acentuações. O que realmente importa é entender como funciona a estrutura de programação, a forma para se desenvolver um algoritmo corretamente que atenda às necessidades para o qual foi desenvolvido, e quanto a isso o Visualg é bem cuidadoso. Uma das vantagens que veremos brevemente, é que o código desenvolvido pode ser diretamente transformado em linhas de código Pascal. Em versões futuras, já é proposta do software, a capacidade de transformar o português estruturado em outras linguagens (C, Clipper e Basic).

Tela inicial do Visualg



Esta é a tela do Visualg, como ela aparecerá quando o programa for iniciado. Note que já é oferecida uma estrutura de algoritmo padrão.

Conheceremos a seguir cada uma das partes do programa e em seguida suas funcionalidades de maneira breve, uma vez que nosso objetivo é ter apenas o conhecimento necessário para usar o Visualg como ferramenta.



Principais Campos do Visualg

ARQUIVO		
Item	Atalho	Descrição
Novo	Ctrl+N	Cria um ambiente para um novo código.
Abrir...	Ctrl+A	Abre arquivo existente.
Salvar	Ctrl+S	Salva código atual.
Salvar Como...	-	Permite escolher nome e local para salvar o código.
Enviar por Email...	-	Permite enviar o código por email.
Imprimir...	Ctrl+P	Imprime o Código atual.
Sair	Alt+F4	Fecha o programa.
EDITAR		
Desfazer	Ctrl+Z	Desfaz a ultima ação.

Refazer	Ctrl+Shift+Z	Refaz a última ação desfeita.
Recortar	Ctrl+X	Recorta bloco selecionado.
Copiar	Ctrl+C	Copia Bloco Selecionado.
Colar	Ctrl+V	Cola Bloco Selecionado.
Corrigir Identação	Ctrl+G	Corrige a indentação.
Selecionar tudo	Ctrl+T	Seleciona o código completo.
Localizar...	Ctrl+L	Localiza palavras no código.
Localizar de novo.	F3	Procura novamente.
Substituir	Ctrl+U	Substitui palavras no código.
Gravar bloco	Ctrl+W	Grava bloco de código selecionado.
Inserir bloco	Ctrl+R	Insere um bloco de código previamente salvo.
EXIBIR		
Número de linhas	-	Apresenta o número de linhas do código
Variáveis Modificadas	-	Apresenta as variáveis que foram modificadas.
Restaurar Tabela	-	
ALGORITMO		
Executar	F9	Executa os códigos.
Passo a Passo	F8	Executa os códigos passo a passo.
Executar com Timer	Shift+F9	Executa os códigos com tempo fixo para cada passo.
Parar	Ctrl+F2	Para a execução do código.
Liga/Desl. Breakpoint	F5	Ativa pontos de parada no código para teste.
Desmarcar Breakpoints	Ctrl+F5	Desmarca todos os pontos de parada.
Executar Modo Dos	-	Ativa tela de DOS para execução do código
Valores Aleatórios	-	Gera valores aleatórios
Perfil	F7	Apresenta linhas de código e quantas vezes foram lidas.
Pilha de Ativação	Ctrl+F3	Apresenta a pilha de memória durante a execução.
CÓDIGO		

Pascal, C, Clipper e Basic, serve para transformar o código de português estruturado para essas linguagens, entretanto, na versão atual apenas Pascal está funcionando.

FERRAMENTAS

Opções	-	Configurações do ambiente de trabalho.
--------	---	--

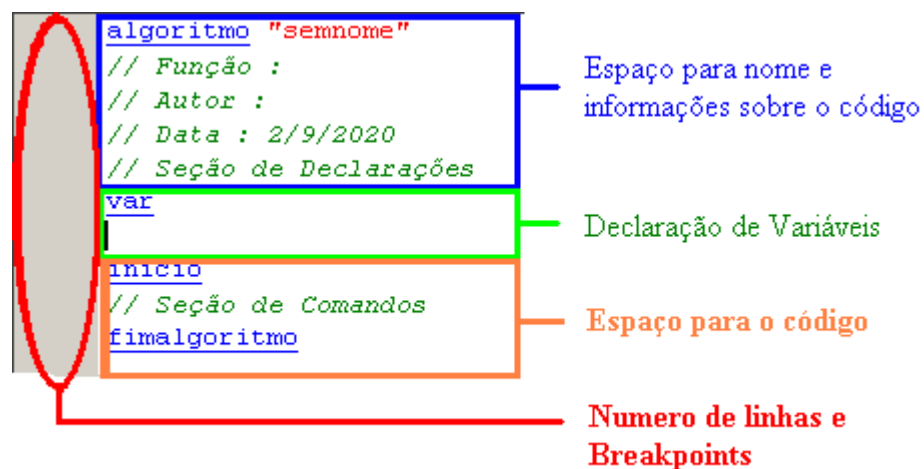
AJUDA

Dica do Dia	-	Apresenta dicas do programa.
A Tela do Visualg	-	Apresenta cada uma das ferramentas disponíveis*
A Linguagem Visualg	-	Apresenta uma visão geral da linguagem visualg
Referências	-	Apresenta uma tela de referências do programa
Sugestão e Comentário	-	Permite enviar email com sugestão ou comentários.
Apoio informática	-	Site do desenvolvedor.
Sobre	-	Informações sobre o programa.

* Como o programa já oferece material sobre cada uma das ferramentas da Barra de Ferramentas, não faremos isso aqui. Também não falaremos da linguagem utilizada o que é facilmente acessível no próprio programa através do menu apresentado acima.

AREA DE TRABALHO

A área de trabalho é onde todo o trabalho deve ser desenvolvido. É nele que você vai codificar seus fluxogramas, programando em português estruturado.



Além das estruturas apresentadas acima, entre a área para declaração de variáveis e o espaço para o código, é possível declarar função. Não apresentaremos a criação de função aqui. Para isso consulte o material disponível no software (Menu Ajuda).

A palavra algoritmo é obrigatória, diante da qual, entre aspas duplas, você deve colocar o nome do pseudo-código.

Logo em seguida é sugerido vários campos de comentário não obrigatórios, mas que permitem organizar melhor os arquivos, sempre documentando os trabalhos desenvolvidos.

Outros comentários podem ser adicionados colocando barras invertidas duplas no início da linha, para que o interpretador saiba que se trata de um comentário.

A palavra var indica que serão declaradas as variáveis globais do programa. Isso deve ser feito entre a palavra reservada var e a palavra reservada inicio.

As palavras reservadas inicio e finalgoritmo delimitam o espaço onde deverão ser acrescentados os códigos do seu pseudo-código.

VARIÁVEIS

É possível acompanhar a criação, alocação de memória, e alterações nas variáveis em tempo de execução. Fica mais fácil de ser utilizado quando estamos utilizando execução passo a passo.

ÁREA DE EXECUÇÃO

Aparece o resultado que apareceria no monitor. Se estiver ativo o modo DOS, aparecerá uma tela preta de DOS, onde serão apresentadas as saídas de seu código, mas mesmo assim, nesta área, ficará registrada a saída de seu código.

Funções do Visualg

Funções Numéricas, algébricas e trigonométricas

Abs(expressão) - Retorna o valor absoluto de uma expressão do tipo inteiro ou real. Equivale a $|expressão|$ na álgebra.

ArcCos(expressão) - Retorna o ângulo (em radianos) cujo co-seno é representado por expressão.

ArcSen(expressão) - Retorna o ângulo (em radianos) cujo seno é representado por expressão.

ArcTan(expressão) - Retorna o ângulo (em radianos) cuja tangente é representada por expressão.

Cos(expressão) - Retorna o co-seno do ângulo (em radianos) representado por expressão.

CoTan(expressão) - Retorna a co-tangente do ângulo (em radianos) representado por expressão.

Exp(base, expoente) - Retorna o valor de base elevado a expoente, sendo ambos expressões do tipo real.

GraupRad(expressão) - Retorna o valor em radianos correspondente ao valor em graus representado por expressão.

Int(expressão) - Retorna a parte inteira do valor representado por expressão.

Log(expressão) - Retorna o logaritmo na base 10 do valor representado por expressão.

LogN(expressão) - Retorna o logaritmo neperiano (base e) do valor representado por expressão.

Pi - Retorna o valor 3.141592.

Quad(expressão) - Retorna quadrado do valor representado por expressão.

RadpGrau(expressão) - Retorna o valor em graus correspondente ao valor em radianos representado por expressão.

RaizQ(expressão) - Retorna a raiz quadrada do valor representado por expressão.

Rand - Retorna um número real gerado aleatoriamente, maior ou igual a zero e menor que um.

RandI(limite) - Retorna um número inteiro gerado aleatoriamente, maior ou igual a zero e menor que limite.

Sen(expressão) - Retorna o seno do ângulo (em radianos) representado por expressão.

Tan(expressão) - Retorna a tangente do ângulo (em radianos) representado por expressão.

Funções para manipulação de cadeias de caracteres (strings)

Asc (s : caracter) : Retorna um inteiro com o código ASCII do primeiro caracter da expressão.

Carac (c : inteiro) : Retorna o caracter cujo código ASCII corresponde à expressão.

Caracpnum (c : character) : Retorna o inteiro ou real representado pela expressão. Corresponde a StrToInt() ou StrToFloat() do Delphi, Val() do Basic ou Clipper, etc.

Compr (c : character) : Retorna um inteiro contendo o comprimento (quantidade de caracteres) da expressão.

Copia (c : character ; p, n : inteiro) : Retorna um valor do tipo character contendo uma cópia parcial da expressão, a partir do character p, contendo n caracteres. Os caracteres são numerados da esquerda para a direita, começando de 1. Corresponde a Copy() do Delphi, Mid\$() do Basic ou Substr () do Clipper.

Maiusc (c : character) : Retorna um valor character contendo a expressão em maiúsculas.

Minusc (c : character) : Retorna um valor character contendo a expressão em minúsculas.

Numpcarac (n : inteiro ou real) : Retorna um valor character contendo a representação de n como uma cadeia de caracteres. Corresponde a IntToStr() ou FloatToStr() do Delphi, Str() do Basic ou Clipper. Pos (subc, c : character) : Retorna um inteiro que indica a posição em que a cadeia subc se encontra em c, ou zero se subc não estiver contida em c. Corresponde funcionalmente a Pos() do Delphi, Instr() do Basic ou At() do Clipper, embora a ordem dos parâmetros possa ser diferente em algumas destas linguagens.