Sumário

1. Lógica de Computadores

1.1. Introdução

O computador é uma máquina que realiza uma variedade de tarefas de acordo com instruções específicas. É uma máquina de processamento de dados, que recebe dados através de um dispositivo de entrada e o processador os manipula de acordo com um programa.

O computador tem dois componentes principais. O primeiro é o Hardware que é a parte palpável (que se pode tocar) do computador. Ele é composto de partes eletrônicas e mecânicas.

O segundo componente principal é o Software que é a parte impalpável (não se pode tocar) do computador. Ele é composto de dados e dos programas de computador.

O software é um programa que o computador usa para funcionar. Ele é armazenado em algum dispositivo de hardware como um disco rígido, mas é em si mesmo intangível. Os dados que o computador usa podem ser qualquer coisa que o programa precise. Os programas agem como instruções para o processador.

Alguns tipos de programas de computador:

1. Programas de Sistemas: Programas necessários para que o hardware e o software funcionem juntos corretamente. Exemplos: Sistemas Operacionais como Linux e outros.
2. Aplicativos: Programas que as pessoas usam para realizar determinado trabalho. Exemplos: Processadores de Textos, Jogos, Planilhas Eletrônicas entre outros.
3. Compiladores: O computador entende apenas uma linguagem: linguagem de máquina. Linguagem de máquina está na forma de zeros e uns. Já que é totalmente impraticável para as pessoas criarem programas usando zeros e uns, é preciso haver uma maneira de traduzir ou converter a linguagem que entendemos em linguagem de máquina, para isto, existem os compiladores.

1.2. Principais elementos de Hardware

Começaremos a falar um pouco sobre o principal para o computador funcionar, ou seja, a parte física dele.

Sem estes elementos físicos o computador não poderia funcionar. Elas interligadas formam o computador.

1.2.1. Memória Principal

A memória, onde se encontram os dados e as instruções que a CPU precisa para realizar suas tarefas, dividida em diversos locais de armazenamento que possuem seus respectivos endereços lógicos. A CPU acessa a memória pelo uso destes endereços.

1.2.2. Processador ou Unidade Central de Processamento (CPU)

A unidade central de processamento (CPU - Central Processing Unit), que comumente é chamada de Processador, é o "cérebro" do computador. Ele possui milhões de partes elétricas muito pequenas. Ele faz as operações fundamentais dentro do sistema. Alguns exemplos de processadores modernos são Pentium 4, Core 2 Duo, Athlon, entre outros.

1.2.3. Unidades de entrada e saída

Os dispositivos de entrada e saída permitem que o computador interaja com o mundo exterior pela movimentação de dados para dentro e para fora do sistema.

Exemplos de dispositivos de entradas são teclados, mouses, microfones, etc. Exemplos de dispositivos de saída são monitores, impressoras, alto-falantes, etc.

1.3. Lógica de programação

A lógica de programação é necessária para pessoas que desejam trabalhar com desenvolvimento de sistemas e programas, ela permite definir a sequência lógica para o desenvolvimento.

Então o que é lógica?

Lógica de programação é a técnica de encadear pensamentos para atingir determinado objetivo.

1.3.1. Sequência lógica (Passos lógicos)

Agora, veremos o que seria a sequência lógica.

Os pensamentos encadeados para atingir determinado objetivo podem ser descritos como uma sequência de instruções, que devem ser seguidas para se cumprir uma determinada tarefa.

Podemos então falar que sequência lógica são passos executados até atingir um objetivo ou solução de um problema.

1.3.2. O que são instruções?

Na linguagem comum, entende-se por instruções "um conjunto de regras ou normas definidas para a realização ou emprego de algo".

Em informática, porém, instrução é a informação que indica a um computador uma ação elementar a executar.

Convém ressaltar que uma ordem isolada não permite realizar o processo completo, para isso é necessário um conjunto de instruções colocadas em ordem sequencial lógica. Por exemplo, se quisermos fazer uma omelete de batatas, precisaremos colocar em prática uma série de instruções: descascar as batatas, bater os ovos, fritar as batatas, etc...

É evidente que essas instruções têm que ser executadas em uma ordem adequada – não se pode descascar as batatas depois de fritá-las.

Dessa maneira, uma instrução tomada em separado não tem muito sentido; para obtermos o resultado, precisamos colocar em prática o conjunto de todas as instruções, na ordem correta.
Instruções são um conjunto de regras ou normas definidas para a realização ou emprego de algo. Em informática, é o que indica a um computador uma ação elementar a executar.

1.4. Definição de algoritmo

Um algoritmo é formalmente uma sequência finita de passos que levam a execução de uma tarefa. Podemos pensar em algoritmo como uma receita, uma sequência de instruções que dão cabo de uma meta específica. Estas tarefas não podem ser redundantes nem subjetivas na sua definição, devem ser claras e precisas.

Como exemplos de algoritmos podemos citar os algoritmos para a realização de operações básicas (adição, multiplicação, divisão e subtração) de números reais decimais. Outros exemplos seriam os manuais de aparelhos eletrônicos, como um videocassete, que explicam passo-a-passo como, por exemplo, gravar um evento.

Até mesmo as coisas mais simples, podem ser descritas por sequências lógicas.

Em outras palavras, podemos falar também que é um processo de cálculo matemático ou de resolução de um grupo de problemas semelhantes, em que se estipulam, com generalidade e sem restrições. Podemos dizer também, que são regras formais para obtenção de um resultado ou da solução de um problema, englobando fórmulas de expressões aritméticas.

1.4.1. Exemplo de um algoritmo (Sequência lógica do dia-a-dia)

Os algoritmos estão presentes no nosso dia-a-dia em coisas simples, como por exemplo, ao escrever, ou abrir uma porta.

Temos como exemplo de um algoritmo:

1. “Abrir uma porta”.
    - Aproximar da porta;
    - Abaixar a maçaneta;
    - Puxar a maçaneta com ela abaixada.
2. “Somar dois números quaisquer”.
    - Escreva o primeiro número no primeiro retângulo;
    - Escreva o segundo número no segundo retângulo;
    - Some o primeiro número com o segundo número e coloque o resultado no terceiro retângulo.

Observe que cada um dos casos, temos 3 ações, que devem ser seguidas passo-a-passo mesmo, pois o não seguimento de uma delas, causará um erro.

Por exemplo, imagine que é construído um braço mecânico de um robô para que toda vez que alguém de aproxime, ele mesmo abra a porta. Se por acaso o passo 2 não seja colocado no

algoritmo, no mínimo o nosso braço mecânico não conseguirá abrir a porta, ou no pior dos casos, ele colocará força que puxará a maçaneta e a quebrará, ou a própria porta.

Por isso notamos uma grande importância os algoritmos e que eles sejam declarados cuidadosamente passo a passo.

É claro que não vamos começar construindo um braço mecânico (nem o vamos fazer) ou outra coisa mais complicada, mas antes de aprender a real utilidade dos algoritmos, iremos construir mais alguns algoritmos do dia-a-dia.

1.4.2. Fazendo algoritmos - Praticando algoritmos do dia-a-dia

Visto o tópico acima vamos tentar resolver e debater alguns tópicos em sala de aula com alguns tópicos do dia-a-dia:

1. Crie uma sequência lógica para tomar banho.
2. Faça um algoritmo para realizar a média entre dois números.
3. Crie uma sequência lógica para trocar um pneu de um carro.
4. Crie uma sequência lógica para fazer um sanduíche de queijo.
5. Faça um algoritmo para trocar uma lâmpada.

1.5. Pseudocódigo (Forma genérica de escrever um algoritmo, utilizando uma linguagem simples)

Os algoritmos são descritos em uma linguagem chamada pseudocódigo. Este nome é uma alusão à posterior implementação em uma linguagem de programação, ou seja, quando formos programar em uma linguagem, por exemplo Python, estaremos gerando código em Python. Por isso os algoritmos são independentes das linguagens de programação. Ao contrário de uma linguagem de programação não existe um formalismo rígido de como deve ser escrito o algoritmo.

O algoritmo deve ser fácil de se interpretar e fácil de codificar. Ou seja, ele deve ser o intermediário entre a linguagem falada e a linguagem de programação.

1.6. Regras para construção do algoritmo

Para escrever um algoritmo precisamos descrever a sequência de instruções, de maneira simples e objetiva. Para isso utilizaremos algumas técnicas:

- Usar somente um verbo por frase;
- Imaginar que você está desenvolvendo um algoritmo para pessoas que não trabalham com informática;
- Usar frases curtas e simples;
- Ser objetivo;
- Procurar usar palavras que não tenham duplo sentido.

1.6.1. Estrutura principal de um algoritmo em Portugol

```
1. programa
2. {
3.      funcao inicio ()
4.      {
5.      }
6. }
```

6.1.1. Fases

Anteriormente vimos que ALGORITMO é uma sequência lógica de instruções que podem ser executadas.

É importante ressaltar que qualquer tarefa que siga determinado padrão pode ser descrita por um algoritmo, como por exemplo:

- Como fazer sanduíche, ou então, como calcular a soma de dois números.

Entretanto ao montar um algoritmo, precisamos primeiro dividir o problema apresentado em três fases fundamentais:

1. ENTRADA: São os dados de entrada do algoritmo;
2. PROCESSAMENTO: São os procedimentos utilizados para chegar ao resultado;
3. SAÍDA: São os dados já processados.

6.2. Tipos de Dados

Os principais tipos de dados são: inteiro, real, caractere, lógico.

- O do tipo **inteiro** é qualquer número inteiro, negativo, nulo ou positivo (-
- 125,0,5,3456);
- O do tipo **real** é qualquer número real, negativo, nulo ou positivo (0.27, -0.01);
- O do tipo **lógico** são conjunto de valores Falso ou verdadeiro (FALSO,
- VERDADEIRO);
- O do tipo **caracter** é qualquer caractere alfanumérico e símbolos colocados entre aspas simples (‘F’, ‘M’, ‘7’).
- O do tipo **cadeia** é qualquer conjunto de caracteres alfanuméricos e símbolos colocados entre aspas duplas (“Instituto Federal”, “E-jovem”, “7”, “FALSO”).

6.3. Identificadores

Os identificadores são os elementos básicos de uma linguagem onde tem como função identificar de forma única variáveis, funções, constantes entre outro

6.3.1. Formação de identificadores

As regras para a formação dos identificadores são:

1. Os caracteres que você pode utilizar são: os números, as letras maiúsculas e minúsculas e o *underline (_);*
2. O primeiro caractere deve ser sempre uma letra;
3. Não são permitidos espaços em branco e caracteres especiais (@, $, +, &, %, !);
4. Não podemos usar palavras reservadas nos identificadores, ou seja, palavras que pertençam a uma linguagem de programação.

Exemplo de identificadores válidos:

A	a	nota
NOTA	a32	NoTa1
MATRICULA	nota_1	IDADE_FILHO

Exemplo de identificadores inválidos:

5b	por começar com número
e 12	por conter espaço em branco
x – y	por conter espaço em branco e caractere especial
prova 2n	por conter espaço em branco
nota(2)	por conter caracteres especiais ()
para	por ser palavra reservada
se	por ser palavra reservada
algoritmo	por ser palavra reservada

6.4. Constantes (ou Literais)

Constante é um determinado valor fixo que não se modifica ao longo do tempo, durante a execução de um programa. Conforme o seu tipo, a constante é classificada como sendo numérica, lógica e literal.

6.5. Variáveis

Variável é a representação simbólica dos elementos de um certo conjunto. Cada variável corresponde a um local, ou seja, a uma posição de memória, onde pode-se armazenar qualquer valor do conjunto de valores possíveis do tipo básico associado, cujo conteúdo pode ser alterado ao longo do tempo durante a execução de um programa. Embora uma variável possa assumir diferentes valores, ela só pode armazenar um valor a cada instante. O nome da variável deve ser um identificador válido.

6.6. Declaração de variáveis

As variáveis só podem armazenar valores de um mesmo tipo, de maneira que também são classificadas como sendo numéricas, lógicas e literais. Corresponde a reserva de locais na memória rotulada com o nome da variável (identificador) e cujo conteúdo será interpretado conforme o tipo declarado.

Exemplo de declaração de variáveis:

`inteiro`	`idade`
`real`	`altura`
`cadeia`	`nome`
`logico`	`filho_unico`
`caracter`	`tipo`

6.7. Atribuição de valores

Para atribuir valores a uma variável, usaremos o operador de atribuição "=" (lê-se "recebe"), que tem um caráter imperativo. Exemplos de atribuição:

```
inteiro idade = 19;
real altura = 1.80;
cadeia nome = "Fulano";
```

Erros comuns nas atribuições:

- Incompatibilidades de tipos - Tentar atribuir um tipo de dado a uma variável de outro tipo;
- Perda de Precisão - Atribuir valores reais a variáveis do tipo inteiro, perdendo a parte decimal do número;
- Atribuição de valor às variáveis não declaradas.

6.8. Comando de Entrada e saída

Comandos de Entrada e saída são instruções que nos permitem, ou receber dados (Entrada) ou informar dados (Saída). O padrão de Entrada para os computadores básicos é o teclado, é dele que nós informamos o que o programa nos solicita. Já o padrão de Saída é o monitor, todas as informações mostradas ao usuário serão passadas por este dispositivo de saída. Em Portugol nós reservamos a palavra ***leia*** para solicitar dados, e a palavra ***escreva*** para enviar uma mensagem ou dados para o monitor, são exemplos de uso correto:

```
escreva ("Digite um número: ")      #Mostra no monitor a mensagem entre aspas
leia numero;                        #Espera o usuário digitar um número
```

6.9. Diagrama de Bloco

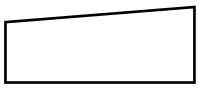
Uma outra maneira de mostrar um bloco de códigos seria por um diagrama, que é uma forma simples e eficaz para demonstrar quais passos lógicos devem ser seguidos pelo programador para chegar a um resultado.

Com o diagrama podemos definir uma sequência de símbolos, com significado bem definido, portanto, sua principal função é a de facilitar a visualização dos passos de um processamento.

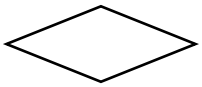
Existem diversos símbolos de um diagrama de bloco. A seguir veremos alguns deles.

TERMINAL — Indica o INÍCIO ou FIM do processamento

PROCESSAMENTO — Processamento em geral



ENTRADA DE DADOS MANUAL — Indica entrada de dados através do teclado

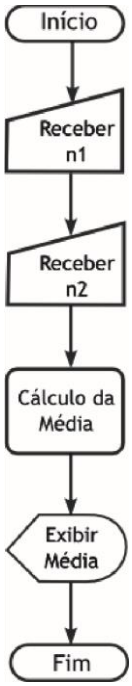


DECISÃO — Indica uma decisão no fluxo do programa

EXIBIR — Mostra informações ou resultados

No interior do símbolo, escrevemos algum tipo de informação, caso contrário os símbolos não nos dizem nada, veja o exemplo a seguir:

Calcular a média de dois números:



Perceba que se seguirmos o fluxo de setas, podemos sem problema algum, ter o resultado, que no caso, é a média das notas.

6.10. Operadores Relacionais

Os operadores relacionais são utilizados para comparar textos e números. Os valores a serem comparados podem ser caracteres ou variáveis.

Estes operadores sempre retornam valores lógicos (verdadeiro ou falso / True ou False). Para estabelecer prioridades no que diz respeito a qual operação executar primeiro, utilize os parênteses.

Os operadores relacionais são:

>	Verifica se um tipo de dado é maior que outro tipo
<	Verifica se um tipo de dado é menor que outro tipo
==	Verifica se um tipo de dado é igual a um outro tipo
>=	Verifica se um tipo de dado é maior ou igual a um outro tipo
<=	Verifica se um tipo de dado é menor ou igual a um outro tipo
! = ou < >	Verifica se um tipo de dado é diferente de um outro tipo

Como exemplo temos:

```
a = 10;
b = 20;
a > b;      # retorna falso
a < b;      # retorna verdadeiro
a == b;     # retorna falso
```

Entre outros testes que o programador pode realizar.

6.11. Operadores Lógicos

Os operadores lógicos servem para combinar resultados e testes de expressões, retornando um valor cujo resultado final é verdadeiro ou falso. Eles são:

and	Chamado também de operador **"e"**. Verifica se ambos os operandos são verdadeiros. Se sim, ele retorna verdadeiro, senão, retorna um valor booleano falso.
or	Chamado também de operador **"ou"**. Verifica se um dos valores testados é verdadeiro. Se sim, ele retorna verdadeiro, senão, retorna um valor booleano falso.
not	Chamado também de operador **"não"**. É usado para negar o valor de uma variável ou expressão

Suponha termos três variáveis:

```
inteiro a = 5;
inteiro b = 8;
inteiro c = 1;
```

Podemos realizar as seguintes comparações e temos como resultado um valor verdadeiro ou falso.

Expressões			Resultados
a == b	and	c < a	FALSO
a != b	and	b > c	VERDADEIRO
	not	a > c	FALSO

Exercício: Sabendo que A=3, B=7 e C=4, informe se as expressões abaixo são verdadeiras ou falsas.

(a) `(A = C) or (B > C)` ( )

(b) `not(B = A) and ((A + 1) = C)` ( )

(c) `(C = (B - A)) or B` ( )

6.12. Estruturas de Decisão e Repetição

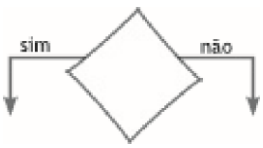
6.12.1. Estrutura *SE...FAÇA ISSO / SENÃO*

Com o conhecimento do tópico acima, nós podemos agora, comentar sobre instruções que realizam testes entre caracteres e variáveis, e sobre laços.
No nosso pseudocódigo a palavra reservada para uma instrução que realiza um teste é o “se”, seu uso é:

```
se <teste> faça isso:
    <bloco de comando>
senão:
    <bloco de comando>
```

NOTA: Perceba que a instrução “senão”, não é obrigatória, pois em alguns caso você pode necessitar fazer somente um teste que não tenha nenhum código a ser executado se o teste for falso.

O símbolo gráfico que representa uma estrutura de decisão simples é mostrado na figura a seguir. Por um lado, o algoritmo segue se a condição testada dentro do símbolo for verdadeira, se for falsa, o algoritmo seguirá por outro caminho, o qual contém um diferente bloco de comando.



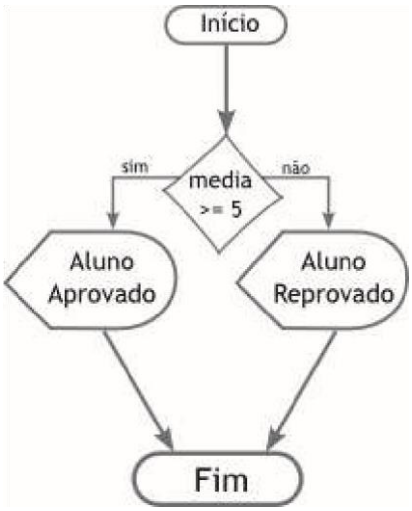
Representa um teste a ser feito

DECISÃO

A estrutura de decisão “se” normalmente vem acompanhada de um comando, ou seja, se determinada condição for satisfeita pelo comando “se” então execute determinado comando, se tal condição não for satisfeita, execute outro determinado comando. Imagine um algoritmo que determinado aluno somente estará aprovado se sua média for maior ou igual a 5.0, veja no exemplo de algoritmo como ficaria.

```
se media >= 5.0 faça isso:
     escrever “Aluno Aprovado”
senão:
     escrever “Aluno Reprovado”
```

No Fluxograma nós temos a seguinte sequência lógica:



Exercício: Elabore um diagrama que satisfaça os seguintes pontos:

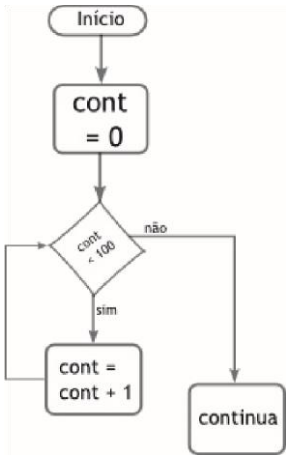
1. Leia 4 (quatro) números;
2. Some o primeiro com o segundo número;
3. Some o terceiro com o quarto número;
4. Se a primeira soma for maior que a segunda soma escreva o resultado e finalize o programa;
5. Senão, imprima a segunda soma e finalize o programa.

6.12.2.Estrutura *ENQUANTO*

Utilizamos os comandos de repetição - neste caso a instrução “enquanto” - quando desejamos que um determinado conjunto de instruções ou comandos sejam executados um número definido ou indefinido de vezes, ou enquanto um determinado estado de coisas prevalecer ou até que seja alcançado. Em pseudocódigo sua sintaxe é:

```
enquanto <condição> faça:
        <bloco de comandos>;
fim enquanto;
```

Neste caso, o bloco de operações será executado enquanto a condição x for verdadeira. O teste da condição será sempre realizado antes de qualquer operação. Enquanto a condição for verdadeira o processo se repete. Podemos utilizar essa estrutura para trabalharmos com contadores. Em diagrama de bloco a estrutura é a seguinte:



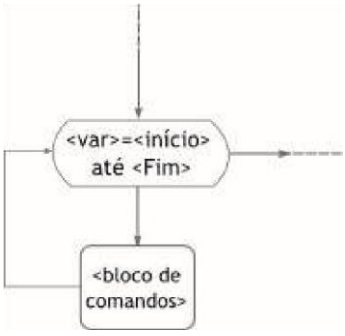
6.12.3. Estrutura *PARA <VAR> DE <início> ATÉ <final> FAÇA*

Os laços contados são úteis quando se conhece previamente o número de vezes que se deseja executar um determinado conjunto de comandos. Então, este tipo de laço nada mais é que uma estrutura dotada de mecanismos para contar o número de vezes que o corpo do laço (ou seja, o bloco de comandos) é executado. A sintaxe usada em pseudocódigo para laços contados é mostrada a seguir:

```
para <var> de <início> até <final> faça:
        <bloco de comandos>
fim para
```

Supomos que nosso PARA, irá fazer um incremento na <var> de 1 uma unidade até que seja atingido o valor <final>.

Em termos de fluxograma, já várias representações possíveis e, até o momento não há consenso quanto à forma mais conveniente.



Algumas observações interessantes a serem comentadas:

- <var> é necessariamente uma variável, uma vez que seu valor é alterado a cada iteração (volta do laço);
- <início> e <fim> podem ser consideradas constantes ou variáveis. No segundo caso (variáveis), algumas linguagens de programação proíbem que seus valores sejam modificados durante a execução do laço;