Fundamentos

Bootcamp: Programador(a) de Software Iniciante

Guilherme Assis

2021

Fundamentos

Bootcamp Programador(a) de Software Iniciante

Guilherme Assis

Sumário

Capítulo 1. Introdução

O objetivo desta apostila é servir de material de apoio para as aulas gravadas do módulo, as quais concentram grande parte do conteúdo teórico e prático. Não é recomendado que o estudo seja feito somente pela apostila, e sim que ela seja utilizada como reforço das aulas gravadas.

Neste módulo, faremos uma introdução à lógica de programação. Porém, antes de começarmos a estudar sobre programação, precisamos entender, mesmo que de forma superficial, como é o funcionamento de um computador, quais são seus principais componentes e como eles interagem.

Após essa compreensão inicial do funcionamento de um computador, começaremos a estudar a programação em si. Como o intuito deste módulo é servir como introdução para a área, o foco maior será na lógica de programação e não tanto em ferramentas e linguagens. Com uma base sólida de lógica de programação e criação de algoritmos, o aprendizado de linguagens de programação e ferramentas utilizadas no mercado fica menos complicado.

Para o estudo da lógica de programação, utilizaremos uma linguagem chamada Portugol, que será executada em um programa chamado VisuAlg. Essa linguagem e o programa são amplamente utilizados em cursos de introdução à lógica de programação no Brasil, pois a escrita da programação se dá com comandos em português, o que caracteriza uma barreira a menos para o aprendizado de quem está iniciando.

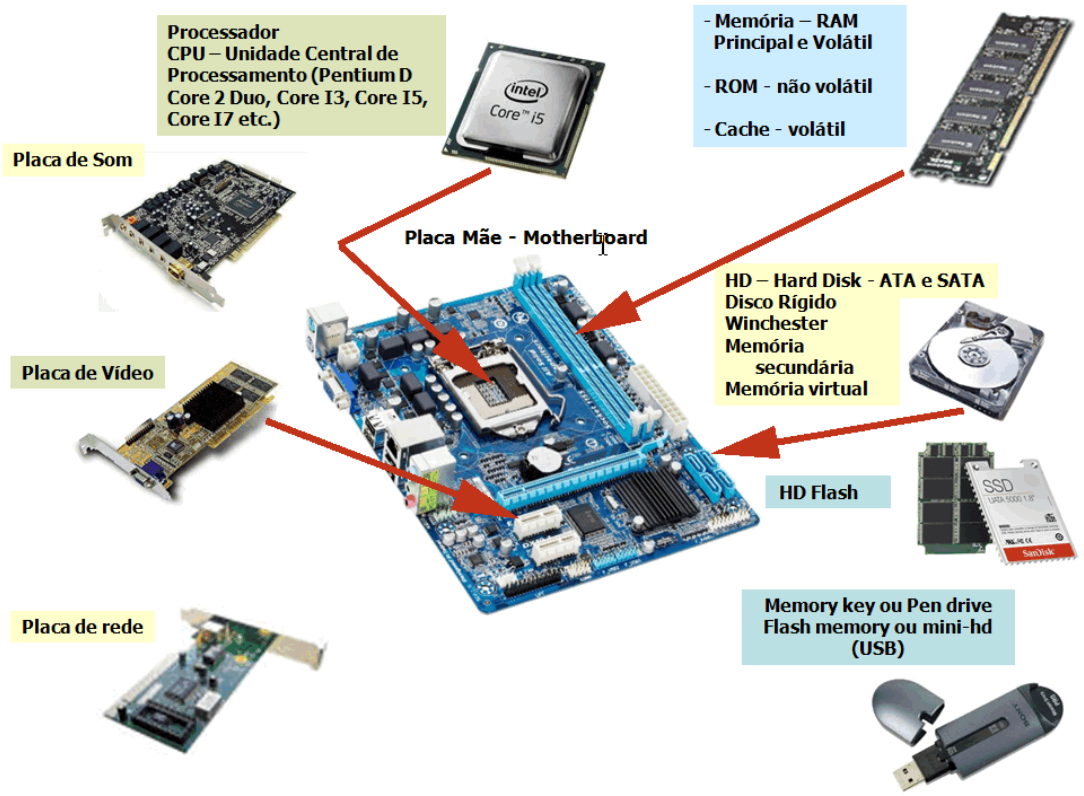
As aulas gravadas são uma combinação de teoria e prática. Inicialmente, o conceito é apresentado de forma teórica e, em seguida, já é praticado com exemplos, facilitando, assim, o entendimento do aluno. É recomendado que o aluno programe juntamente com as aulas no seu próprio computador, dando pausa no vídeo ou voltando em alguns momentos, caso seja necessário, para poder acompanhar. Isso é recomendado, porque a programação é uma atividade

extremamente prática, de forma que, se o estudante apenas assistir a alguém programar, dificilmente se tornará um bom programador. Quanto mais praticar, mais a sua lógica será treinada para a forma de pensar programação, facilitando, assim, a criação de algoritmos para a resolução de problemas. Errar o código ou a lógica elaborada é algo extremamente normal, e mesmo programadores com anos de experiência erram todos os dias. Faz parte do processo de aprendizagem buscar o erro no seu código. Quando você encontra o motivo que fazia aquele trecho de código não funcionar, na próxima vez que você se deparar com o mesmo problema, terá uma facilidade muito maior para a resolução. Ou seja, não tenha medo de tentar e errar, pois faz parte do processo.

1.1. Componentes de um Computador

Um computador é composto por diversos componentes. A imagem a seguir ilustra alguns dos principais. Abaixo, listaremos alguns componentes, com uma explicação do seu papel.

Figura 1 - Componentes de um computador



Fonte: https://www.monolitonimbus.com.br/dicas-de-manutencao-de-computadores/

- **Placa Mãe:** placa na qual todos os demais componentes são encaixados. É por ela que os componentes se comunicam.
- **Processador:** considerado o cérebro do computador, realiza as tarefas e cálculos. Ele é quem designa tarefas aos demais componentes.
- **Memória RAM:** trabalha em conjunto com o processador para o processamento de informações, atuando como uma memória de apoio para que o processador possa armazenar e buscar informações. Possui alta velocidade de leitura e escrita. A memória RAM é volátil, quando o computador é reiniciado, as informações são perdidas.
- **Memória Secundária:** também conhecida como memória de massa, é representada pelos HDs e SSDs. Alta capacidade de armazenamento, porém é mais lenta para leitura e escrita do que a memória RAM.

- **Placa de Rede:** interface de comunicação com a internet, por onde são enviados e recebidos dados.
- **Placa de Vídeo:** realiza a apresentação de informações para o usuário na tela, roda jogos e softwares que demandam processamento visual.
- **Placa de Som:** processa os sons para serem reproduzidos em uma caixa de som ou fone de ouvido, por exemplo, além dos sons obtidos via microfone.

O computador possui diversos dispositivos de entrada, como teclado, mouse, microfone e scanner, por exemplo. Como dispositivos de saída podemos citar o monitor, impressora e os alto-falantes. Quanto aos dispositivos que são ao mesmo tempo de entrada e saída, temos, por exemplo, o modem da internet, que recebe e envia dados, e a memória secundária, que lê e armazena informações.

1.2. CPU e Memória

CPU é a sigla para Central Processing Unit, ou em português, Unidade Central de Processamento. A CPU também é chamada de processador, sendo o principal item do computador, atuando como se fosse o cérebro. Ele é o responsável por realizar cálculos e tarefas, delegando ações para os demais itens do computador. O processador está ligado diretamente à placa mãe do computador.

O processador trabalha em conjunto à memória RAM, executando instruções que estiverem nela e utilizando-a para armazenar dados durante a execução de tarefas. A CPU trabalha com o seguinte fluxo: entrada de dados, processamento das informações e saída de dados. Por exemplo, quando você digita uma tecla no teclado, ela é enviada para a CPU (input), que, ao receber o dado, informa à interface de vídeo que ele precisa exibir aquele caractere na tela (output).

A CPU não trabalha diretamente com a memória secundária do computador (HD). Para trabalhar com dados armazenados no HD, eles, antes, precisam ser

transferidos para a memória RAM para que, assim, a CPU possa trabalhar com eles.

Nós trabalhamos com o sistema decimal, representado pelos números 0, 1, 2, 3, 4, 5, 6, 7, 8 e 9. O computador, por sua vez, não trabalha nesse sistema, pois ele não é adequado para os circuitos internos do computador. Ele trabalha no sistema binário, representado pelos dígitos 0 e 1, também chamados de bits. Bit é a abreviatura de Binary Digit. Para armazenar informações maiores, como letras e algarismos, são utilizados grupos de bits, também chamados de bytes. 1 byte consiste em um grupo de 8 bits. Por exemplo, para armazenar a letra A, é utilizado o seguinte byte: 01000001. Para armazenar a letra B, é utilizando a seguinte sequência de bits: 01000010. Ninguém precisa decorar esses bytes, o computador faz essa interpretação internamente.

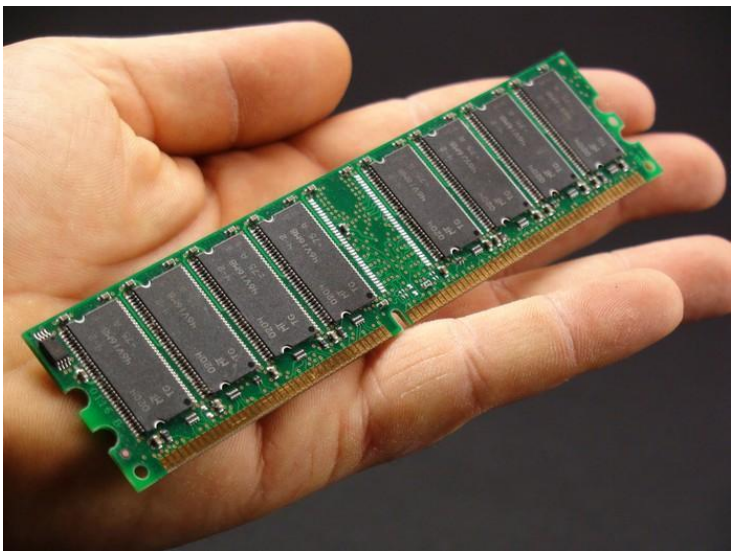
Quando precisamos armazenar um grande volume de informações, são necessários vários bytes. Existem nomenclaturas para podermos falar sobre vários bytes em conjunto. Essas nomenclaturas são muito utilizadas quando queremos falar sobre a capacidade de um HD ou da memória RAM. Segue alguns desses nomes:

- Bit: 0 ou 1
- Bytes: grupo de 8 bits
- KB (kilobyte): 1024 bytes
- MB (megabyte): 1024 KB
- GB (gigabyte): 1024 MB
- TB (terabyte): 1024 GB

A memória RAM, também chamada de memória principal, é a memória de apoio do processador para processamento das informações. Ela é acessada diretamente por ele, sendo utilizada para armazenar e executar programadas. Ela é

uma memória volátil, ou seja, quando o computador é desligado, todos os dados são perdidos. A imagem abaixo ilustra uma memória RAM.

Figura 2 - Memória RAM



Fonte: https://www.techtudo.com.br/dicas-e-tutoriais/noticia/2013/11/como-comprar-uma-memoria-ram-para-o-computador-o-que-levar-em-consideracao.html

Já a memória secundária, também chamada de memória de massa, memória permanente ou HD (Hard-Disk), não perde seus dados quando o computador é reiniciado. Ela possui uma alta capacidade de armazenamento, sendo mais barata que a memória principal. Porém, ela é bem mais lenta para realizar leitura e escrita de informações em relação à memória principal. O processador não trabalha diretamente com a memória secundária, pois, além de serem incompatíveis estruturalmente, a menor velocidade de leitura e de escrita inviabilizaria o trabalho do processador. As imagens abaixo ilustram dois tipos de memória secundária, a

primeira representando um HD e a segunda um SSD, que possui uma velocidade de leitura e escrita maior que o HD tradicional.

Figura 3 - HD

Fonte: https://www.techtudo.com.br/dicas-e-tutoriais/noticia/2016/03/como-instalar-um-hd-sata-extra-no-seu-computador.html

Figura 4 - SSD

Fonte: https://www.kingston.com/br/embedded/design-in-ssd

1.3. Prompt de Comandos

O usuário interage com o computador a partir de um sistema operacional, que é um programa carregado na memória quando o computador é ligado. Ele executa os comandos solicitados pelo usuário. Ele também gerencia os recursos de um computador, evitando, por exemplo, que outros programas entrem em conflito, como ao tentarem acessar uma mesma parte da memória. O sistema operacional mais comum, hoje em dia, é o Windows, mas temos outros também muito utilizados, tais como o Linux e o macOS.

O prompt de comandos é um aplicativo de linha de comando utilizado para executar comandos. Ele também é chamado de terminal. Ele permite um acesso a partir da linha de comandos, sem uma interface gráfica muito robusta, a funcionalidades do computador, como por exemplo acessar, criar e excluir arquivos e pastas. É importante para os programadores saberem trabalhar minimamente com o prompt de comandos, pois, em muitas linguagens de programação, o programador irá executar o programa durante o trabalho diretamente pelo terminal. A imagem abaixo ilustra um prompt de comandos.

Figura 5 - Prompt de comandos

Fonte: Elaborada pelo próprio professor

Existem diversos comandos que podem ser utilizados para trabalhar a partir do prompt de comandos, segue alguns deles:

- cd: trocar de pasta
- mkdir: criar uma pasta
- dir: listar os arquivos e pastas no diretório atual
- cls: limpar a tela
- echo: imprimir frase na tela
- copy: copiar arquivo
- del: excluir arquivo
- rmdir: apagar pasta

Capítulo 2. Lógica de Programação

2.1. Introdução à Programação

Um programa de computador é uma sequência de passos definidos por um programador para alcançar um objetivo. No nosso dia a dia, utilizamos diversos programas em nosso computador, como por exemplo Word, Excel, entre outros.

Esses programas também podem ser chamados de software. Software não é palpável e sim instalável no computador, enquanto hardware é algo palpável, como por exemplo o notebook, mouse, teclado etc.

Os computadores só entendem bits (0 e 1), eles não entendem palavras formadas por letras, por exemplo. Para representar letras, números, símbolos etc., são utilizados um conjunto de bits, também chamado de byte. Por exemplo, para representar a letra “a”, o conjunto de bits seria 01100001. Seria inviável programar dessa forma, por isso são utilizadas as linguagens de programação.

Com as linguagens de programação, o programador pode, então, escrever os comandos em uma linguagem de alto nível, e depois uma outra ferramenta fica responsável por traduzir esse código para a linguagem que o computador entende. Existem várias linguagens de programação, como por exemplo JavaScript, Java e Python.

Um algoritmo é uma sequência de passos que devem ser executados para atingir determinado objetivo, como por exemplo uma receita de bolo. Um algoritmo não necessariamente é um programa de computador, ele pode ser executado por uma pessoa. Uma tarefa pode ser realizada por diferentes algoritmos, não existe somente uma forma de correta de realizar determinada ação.

2.2. VisuaAlg

O VisuAlg é um programa que permite a criação e execução de algoritmos escritos em Portugol, também conhecido como Português Estruturado. Essa linguagem permite a escrita de algoritmos utilizando comandos em português, sendo assim muito indicada para iniciar os estudos de lógica de programação.

Figura 5 - VisuAlg

Fonte: https://visualg3.com.br/

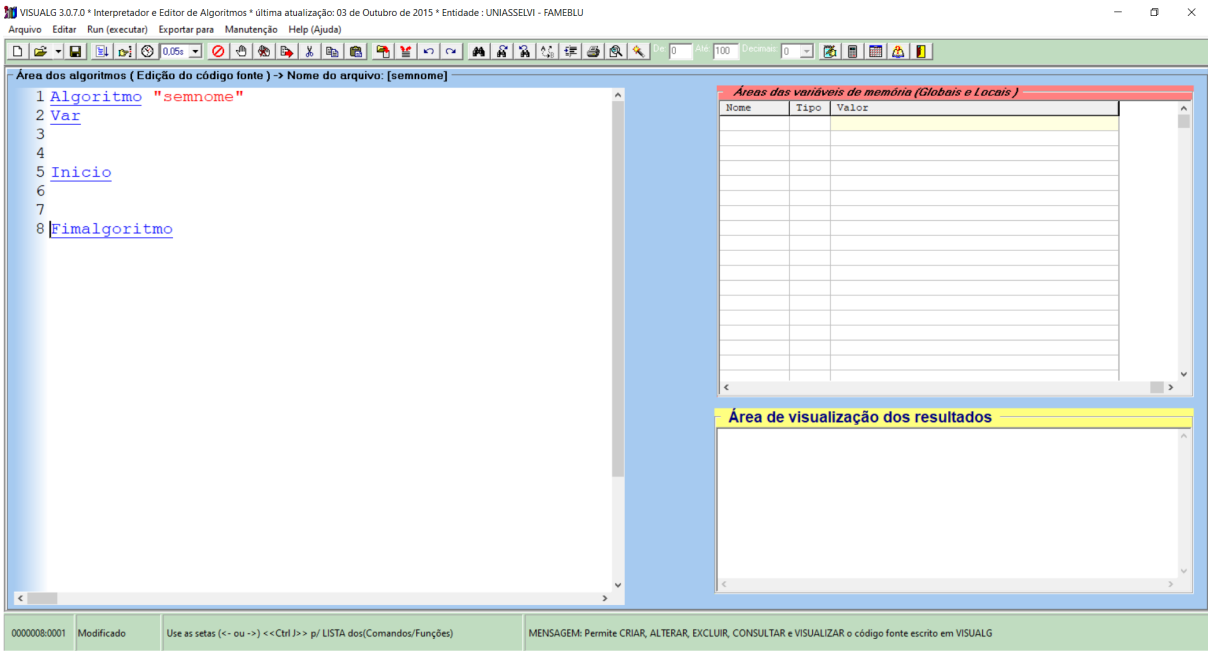
O VisuAlg possui uma interface intuitiva, que facilita o aprendizado. No lado esquerdo da tela, fica o código; no lado direito superior, ficam todas as variáveis que estão sendo criadas pelo programa e seus valores. No canto inferior direito, fica uma área para visualização dos resultados do programa. Esses resultados também podem ser vistos numa janela que é aberta durante a execução.

Para executar um código desenvolvido, basta acessar o menu “Run (executar)” e depois “Rodar o algoritmo”, ou senão utilizar o atalho pelo teclado apertando a tecla F9. Caso queira acompanhar a execução passo a passo, basta

apertar a tecla F8 ou ir pelo menu “Run (executar)” e, depois, “Rodar Passo a passo”. Esse processo é chamado de debug ou depuração e permite que você veja exatamente o que o algoritmo está fazendo em cada etapa, facilitando a correção de eventuais problemas.

O VisuAlg 3.0 foi criado pelo professor Antônio Carlos Nicolodi, criador também do Portugol. A imagem abaixo ilustra a interface do VisuAlg.

Figura 6 - Interface do VisuAlg



Fonte: Elaborada pelo próprio professor

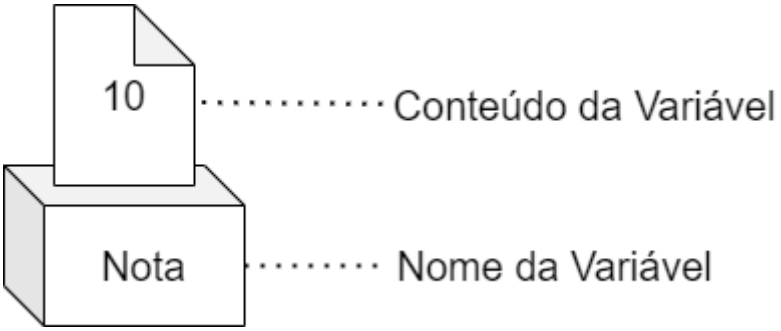
2.3. Variáveis

Uma variável é um local que podemos armazenar um valor. Essa variável fica armazenada na memória e pode ser utilizada pelo algoritmo para realizar suas ações. Um algoritmo pode ter diversas variáveis, e elas podem ter seus valores alterados durante a execução do algoritmo. Com as variáveis, é possível realizar

operações, tais como adição e subtração. Os valores das variáveis podem ser alterados de acordo com a lógica implementada.

As variáveis podem ter tipos diferentes, como texto e número, por exemplo. Cada variável tem um identificador, que é o seu nome, que será utilizado pelo programador durante o código para utilizá-la. Cada variável também tem o seu conteúdo. A imagem abaixo ilustra uma variável chamada nota, que possui o valor 10.

Figura 7 - Variáveis



Fonte: Elaborada pelo próprio professor

Podemos pensar nas variáveis como se fossem uma caixa, que tem um nome e podem armazenar coisas, neste caso, os valores, como um texto ou um número, por exemplo. Você pode, em determinado momento, trocar o conteúdo dessa caixa, tirando o item anterior e colocando outro. Nas variáveis, você também pode alterar o valor desta variável. As variáveis ficam armazenadas na memória RAM, e o processador acessa e altera esses valores à medida que o algoritmo vai sendo executado.

Por exemplo, vamos pensar no funcionamento de uma calculadora. O usuário precisa informar à calculadora a operação que ele deseja fazer, como por exemplo uma soma, e os dois números envolvidos na conta. O algoritmo pode, então, armazenar esses valores em duas variáveis e realizar a operação armazenando o seu resultado em uma outra variável. No fim da execução, essa variável com o resultado poderia ser impressa na tela. A imagem abaixo ilustra a criação de algumas variáveis.

Figura 8 - Criando variáveis

```
1  Algoritmo "Variaveis"
2  Var
3     Abacate, Melancia, TotalFrutasSacola: Inteiro
4  Inicio
5     //Declarando variaveis
6     Abacate <- 2
7     Melancia <- 3
8
9     //Declaracao variavel da soma
10    TotalFrutasSacola <- Abacate + Melancia
11
12    //Trocando o valor da variavel abacate
13    Abacate <- 5
14
15    //Somando novamente
16    TotalFrutasSacola <- Abacate + Melancia
17
18    //Imprimindo resultado
19    Escreva(TotalFrutasSacola)
20 Fimalgoritmo
```

Fonte: Elaborada pelo próprio professor

Na figura acima, podemos ver a estrutura de um algoritmo no VisuAlg. O nome do algoritmo fica na primeira linha, em frente à palavra “Algoritmo”. Por convenção, utilizaremos a escrita das palavras sempre iniciando com letra maiúscula e as demais minúsculas. Caso o nome seja composto, ou seja, mais de uma palavra, o nome do comando fica todo junto, sem espaços, porém com a letra inicial de cada palavra maiúscula. Por exemplo, para criar uma variável para armazenar o total de frutas de uma sacola, chamaríamos de “TotalFrutasSacola”. O programa irá funcionar se você colocar as letras todas minúsculas, no entanto aqui utilizaremos o formato conforme descrito para facilitar a leitura e o entendimento.

Após a definição do nome do algoritmo, na linha 2, temos a palavra “Var”, que indica que ali se inicia a parte da criação das variáveis do programa. Na próxima seção, veremos os tipos de variáveis existentes. Neste exemplo, as variáveis “Abacate”, “Melancia” e “TotalFrutasSacola” irão receber números inteiros, por isso temos um “: Inteiro” na frente das variáveis, indicando que elas poderão receber somente números inteiros, e não poderemos atribuir a elas textos ou números com casas decimais, por exemplo.

Na linha 4, temos a palavra “Inicio” marcando que, a partir dali, inicia-se o algoritmo. O corpo do algoritmo fica entre a palavra “Inicio” e “Fimalgoritmo”, que, neste exemplo, está na linha 20. Dentro desse algoritmo, escrevemos o código colocando um espaçamento de 3 espaços à esquerda, para indicar que o código está dentro do bloco do algoritmo. Isso é uma convenção e poderia funcionar de outra forma, mas adotaremos dessa maneira com o objetivo de facilitar a leitura e o entendimento.

Nas linhas 6 e 7, são atribuídos valores às variáveis. No Portugol, isso é feito com o operador “<-“, indicando que a variável está recebendo aqueles valores. Na linha 10, estamos fazendo uma soma das variáveis “Abacate” e “Melancia” e atribuindo o resultado à variável “TotalFrutasSacola”. Na linha 13, é feita uma mudança de valor da variável, e, na 16, é somado novamente, atualizando o valor

antigo. Na linha 19, é utilizado o comando “Escreva”, que imprime na tela o resultado.

2.4. Tipos de Variáveis

As variáveis podem ter tipos diferentes. Por exemplo, uma variável pode receber um texto, enquanto outra recebe um número. No Portugol, ao definir que uma variável será de um determinado tipo, ela não poderá receber dados de outro tipo posteriormente. A seguir, há os quatro tipos de variáveis que utilizaremos no VisuAlg:

- **Caractere:** também chamado de string, é representado por aspas duplas. Por exemplo: “Teste de Caractere”
- **Inteiro:** representa um número inteiro, que não aceita casas decimais. Por exemplo: 100.
- **Real:** número que aceita casas decimais. Por exemplo: 100.95.
- **Lógico:** aceita valores booleanos, VERDADEIRO ou FALSO.

A imagem abaixo ilustra a criação de variáveis dos tipos citados acima, com atribuições de valores possíveis e a impressão na tela.

Figura 9 - Tipos de variáveis

```
1  Algoritmo "TiposVariaveis"
2  Var
3     Modelo: Caractere
4     Ano: Inteiro
5     Valor: Real
6     IpvaPago: Logico
7     SeguroPago: Logico
8  Inicio
9     Modelo <- "Gol"
10    Ano <- 2010
11    Valor <- 20500.14
12    IpvaPago <- FALSO
13    SeguroPago <- VERDADEIRO
14
15    Escreval(Modelo)
16    Escreval(Ano)
17    Escreval(Valor)
18    Escreval(IpvaPago)
19    Escreval(SeguroPago)
20 Fimalgoritmo
```

Fonte: Elaborada pelo próprio professor

2.5. Operadores

Os operadores são utilizados para realizar operações entre números e variáveis. Abaixo, listamos os principais operadores e as operações realizadas por eles:

- ❑ Soma: +
- ❑ Subtração: -
- ❑ Multiplicação: *
- ❑ Divisão: /
- ❑ Resto: %

- Precedência: ( )
- Incremento: + 1
- Decremento: - 1

O operador de precedência é utilizado para modificar a precedência natural dos operadores, assim como acontece na matemática. Na matemática, os operadores de multiplicação e divisão têm maior precedência quanto aos de soma e subtração. Por exemplo, nesta operação: 10 + 80 / 8 + 2, o resultado é 22, pois, primeiro, é realizada a divisão e, depois, as somas. Caso queira realizar as somas antes da divisão, é preciso colocar os parênteses da seguinte forma (10 + 80) / (8 + 2). Nessa expressão, o resultado seria 9.

Os operadores de incremento e decremento são muito utilizados para servirem de contadores para outra ação. Ao longo do código, uma variável pode ir sendo incrementada ou decrementada, como uma espécie de auxílio ao que está sendo realizado. Isso será muito utilizado quando estudarmos as estruturas de repetição. A imagem abaixo ilustra a utilização dos operadores citados anteriormente.

Figura 10 - Operadores

```
1  Algoritmo "Operadores"
2  Var
3     Soma, Subtracao, Multiplicacao, Resto: Inteiro
4     Divisao, Potenciacao, Resultado: Real
5     A, B, Incremento, Decremento: Inteiro
6  Inicio
7     Soma <- 4 + 2
8     Subtracao <- 5 - 3
9     Multiplicacao <- 5 * 4
10    Divisao <- 20 / 4
11    Resto <- 20 % 3
12    //3 ^ 4 = 3 * 3 * 3 * 3
13    Potenciacao <- 3 ^ 4
14
15    //precedencia
16    A <- 10
17    B <- 80
18    Resultado <- (A + B) / (8 + 2)
19
20    //incremento
21    Incremento <- 11
22    Incremento <- Incremento + 1
23
24    //decremento
25    Decremento <- 10
26    Decremento <- Decremento - 1
27 Fimalgoritmo
```

Fonte: Elaborada pelo próprio professor

2.6. Entrada de Dados

Um programa pode solicitar que o usuário digite informações. Isso fará com que o programa fique dinâmico, variando o resultado final de acordo com os dados informados. Isso também prova uma interação do usuário com o programa.

No VisuAlg, essa interação é realizada através da linha de comando. Podemos utilizar o comando “Escreva” para imprimir valores na tela, e o comando “Leia” para solicitar ao usuário que digite uma informação. Para o comando “Leia”, uma variável é passada entre parênteses e é nessa variável que o dado digitado pelo usuário será armazenado.

Por exemplo, na imagem abaixo é mostrado um algoritmo para calcular a idade de um usuário calculando a diferença de anos entre o ano atual e o ano de seu nascimento. O ano atual e o ano de nascimento são armazenados nas variáveis “AnoAtual” e “AnoNascimento”, pois elas são passadas para o comando “Leia”, que irá aguardar o usuário digitar as informações na linha de comando. Antes de solicitar ao usuário que digite alguma coisa, é bom informá-lo sobre o que ele precisa fazer, utilizando para isso o comando “Escreva”.

Figura 11 - Entrada de dados

```
1  Algoritmo "CalculoIdade"
2  Var
3     AnoAtual, AnoNascimento, Resultado: Inteiro
4  Inicio
5     Escreva("Informe o ano atual: ")
6     Leia(AnoAtual)
7
8     Escreva("Informe o ano do seu nascimento: ")
9     Leia(AnoNascimento)
10
11    Resultado <- AnoAtual - AnoNascimento
12    Escreva("Sua idade é: ", Resultado)
13 Fimalgoritmo
```

Fonte: Elaborada pelo próprio professor

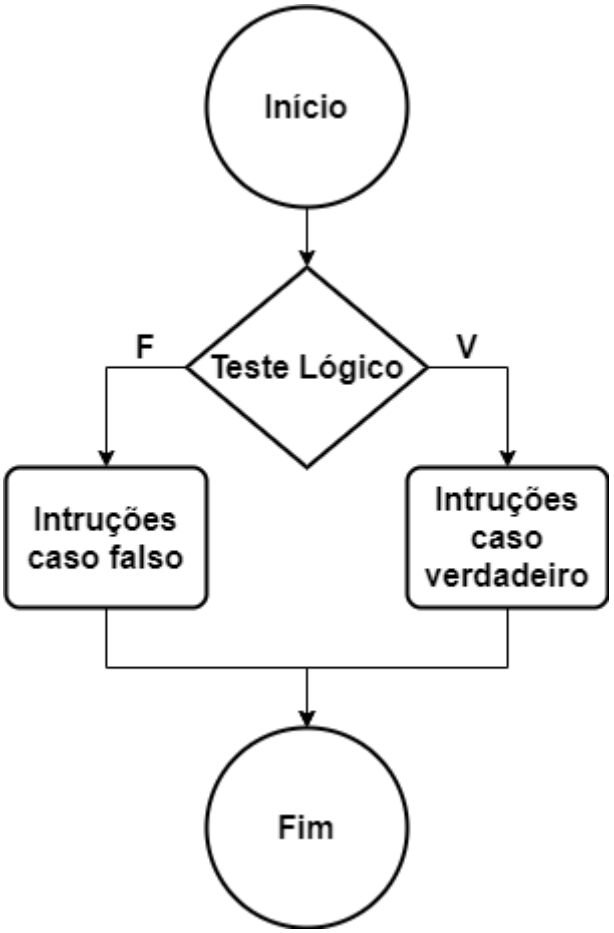
2.7. Estruturas de Decisão

Estruturas de decisão são utilizadas para que o código consiga tomar decisões de acordo com as condições fornecidas. Elas servem para desviar o fluxo de execução de acordo com os critérios definidos.

Também são chamadas de estruturas condicionais, servindo para desviar o fluxo de execução do algoritmo de acordo com determinadas condições. O programador deve especificar uma ou mais condições a serem testadas pelo algoritmo. É possível encadear estruturas de decisão.

A imagem abaixo ilustra o fluxo de execução de uma estrutura de decisão do tipo Se Senão. Inicialmente, é feito um teste lógico, como por exemplo a verificação se determinada condição é verdadeira. Se for verdadeira, um bloco de instruções será executado. Se for falsa, um outro bloco de condições é executado.

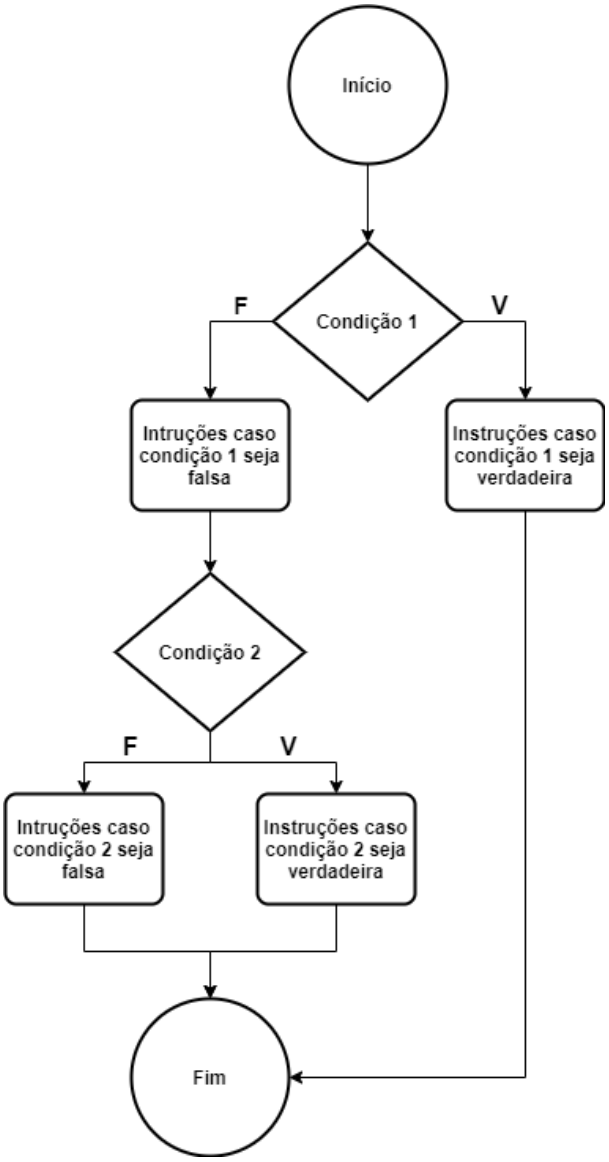
Figura 12 – Fluxo Se / Senão



Fonte: Elaborada pelo próprio professor

É possível encadear novas verificações caso o primeiro teste lógico seja falso. A imagem a seguir ilustra este fluxo. Nela, caso a “Condição 1” seja falsa, o código pode executar algumas instruções caso queira e, depois, fazer outro teste, chamado de “Condição 2”, podendo variar o restante do código, caso ela seja verdadeira ou falsa.

Figura 13 - Fluxo Se / Senão Se / Senão



Fonte: Elaborada pelo próprio professor

A imagem abaixo ilustra a utilização da estrutura “Se Senão” no Portugol. No algoritmo da imagem, é verificado se um aluno passou de ano analisando se a nota informada pelo usuário é maior ou igual a 60. Se for, é impresso na tela que o aluno foi aprovado, senão é verificado se a nota é maior ou igual a 40, neste caso, o aluno iria para a recuperação e, em caso contrário, estaria reprovado. Os comandos utilizados no Portugol são “Se”, “Entao”, e deve-se sempre fechar o bloco com um

"FimSe". O lado falso da condição é feito utilizando a palavra "Senao". Dentro do "Senao", é possível encadear vários outros comandos do tipo "Se", bastando apenas se lembrar de fechá-los com um "FimSe", para que o código não gere erro.

Figura 14 - Estrutura de decisão

```
1  Algoritmo "AprovacaoAluno"
2  Var
3     Nota: Inteiro
4  Inicio
5     Escreva("Digite a nota do aluno: ")
6     Leia(Nota)
7
8     Se Nota >= 60 Entao
9        Escreva("Aluno aprovado")
10    Senao
11       Se Nota >= 40 Entao
12          Escreva("Aluno vai para a recuperacao")
13       Senao
14          Escreva("Aluno reprovado")
15       FimSe
16    FimSe
17 Fimalgoritmo
```

Fonte: Elaborada pelo próprio professor

Os testes lógicos que são realizados podem ser feitos com vários operadores de comparação. Os testes podem ser feitos com valores ou variáveis. Abaixo, listamos os operadores possíveis:

- **Igual a:** valida se duas variáveis ou valores são iguais
    - Exemplo: 3 = 3

- **Menor que:** valida se o primeiro valor é menor que o segundo
    - Exemplo: 5 < 7
- **Maior que:** valida se o primeiro valor é maior que o segundo
    - Exemplo: 7 > 2
- **Menor ou igual a:** valida se o primeiro valor é menor ou igual ao segundo
    - Exemplo: 4 <= 5
- **Maior ou igual a:** valida se o primeiro valor é maior ou igual ao segundo
    - Exemplo: 6 >= 6
- **Diferente:** valida se o primeiro valor é diferente ao segundo
    - Exemplo: 7 <> 4

Uma outra estrutura de condição é a chamada “Escolha”. A imagem abaixo ilustra a sua utilização. Em frente à palavra “Escolha”, é colocada uma variável, que será utilizada para comparar com os valores passados em frente às palavras “Caso”. O algoritmo compara a variável escolhida com os valores colocados em “Caso”, até que se encontre algum valor que seja igual. Ao encontrar, o código abaixo dele é executado. Caso nenhum dos valores em frente aos comandos “Caso” seja igual ao da variável colocado em frente ao “Escolha”, o código pode executar o código abaixo do “OutroCaso”. Essa estrutura deve ser encerrada com um “FimEscolha”. A imagem a seguir ilustra um trecho de um algoritmo para uma calculadora, no qual a variável operação seria informada pelo usuário e, de acordo com a operação solicitada, seriam feitos os cálculos com os números Num1 e Num2, também informados pelo usuário. Caso nenhuma das alternativas fosse verdadeira, o código imprimiria na tela “Operacao invalida”, pois o usuário teria digitado uma operação que a calculadora não suporta.

Figura 14 - Estrutura escolha

```
Escolha Operacao
   Caso "SOMA"
      Resultado <- Num1 + Num2
   Caso "SUBTRACAO"
      Resultado <- Num1 - Num2
   Caso "MULTIPLICACAO"
      Resultado <- Num1 * Num2
   Caso "DIVISAO"
      Resultado <- Num1 / Num2
   Caso "RESTO"
      Resultado <- Num1 % Num2
   Caso "PORCENTAGEM"
      Resultado <- Num1 * Num2 / 100
   OutroCaso
      Escreva("Operacao invalida")
FimEscolha
```

Fonte: Elaborada pelo próprio professor

2.8. Lógica Booleana

A lógica booleana, também chamada de álgebra booleana, serve para representarmos a lógica em expressões. Quando falamos valores lógicos, estamos nos referindo a VERDADEIRO ou FALSO, também chamados de valores booleanos ou lógicos.

Assim como fazemos contas com operadores como soma e subtração, por exemplo, podemos montar expressões lógicas com operadores específicos. Estudaremos três operadores muito utilizados, o “E”, “OU” e “NAO”, também conhecidos por “AND”, “OR” e “NOT”. Cada operador possui o que chamamos de tabela verdade, mostrando a combinação dos valores possíveis e o resultado esperado.

O operador “E”, retorna verdadeiro somente se os dois lados da expressão forem verdadeiros. Por exemplo: (A > B) E (C > D). Nessa situação, o resultado

seria verdadeiro somente se A fosse maior do que B e C fosse maior do que D. Se somente A fosse maior do que B, e C fosse igual a D, por exemplo, o resultado da expressão seria falso. A seguir, temos a tabela verdade do operador “E”:

Valor 1	Valor 2	Operação E
Verdadeiro	Verdadeiro	Verdadeiro
Verdadeiro	Falso	Falso
Falso	Verdadeiro	Falso
Falso	Falso	Falso

O operador “OU”, por sua vez, retorna verdadeiro como resultado da expressão se pelo menos um dos lados da expressão retornar verdadeiro. Ele só retorna falso caso os dois lados da expressão sejam falsos. Por exemplo, na expressão (A > B) OU (C > D), se A for maior do que B a expressão será verdadeira, independentemente do resultado da segunda expressão. Da mesma forma, se C for maior do que D, o resultado total também será verdadeiro. Se os dois lados forem verdadeiros, o resultado total também será verdadeiro. A tabela verdade do operador “OU” é a seguinte:

Valor 1	Valor 2	Operação OU
Verdadeiro	Verdadeiro	Verdadeiro
Verdadeiro	Falso	Verdadeiro
Falso	Verdadeiro	Verdadeiro

Falso	Falso	Falso

O operador “NÃO” inverte o resultado da expressão. Por exemplo, se utilizado em um SE da seguinte forma: “SE NAO A ENTAO”, a expressão total retornaria verdadeiro somente se “A” for falso, pois o “NÃO” inverteria o falso para verdadeiro. A tabela verdade do operador “NÃO” é a seguinte:

Valor	**Operação NÃO**
Verdadeiro	Falso
Falso	Verdadeiro

A imagem abaixo ilustra um exemplo de código utilizando esses três operadores. Utilizando o operador “E”, caso as duas variáveis sejam verdadeiras, o texto é impresso na linha de comando. Utilizando o operador “OU”, caso pelo menos uma das duas sejam verdadeiras, o texto é impresso. Por fim, utilizando o operador “NÃO”, o texto é impresso somente se a variável for falsa.

Figura 15 - Operadores E, OU e NÃO

```
1  Algoritmo "semnome"
2  Var
3     HojeFezCalor, OntemFezCalor: Logico
4  Inicio
5     HojeFezCalor <- VERDADEIRO
6     OntemFezCalor <- FALSO
7
8     Se HojeFezCalor E OntemFezCalor Entao
9        Escreval("Fez calor ontem e hoje")
10    FimSe
11
12    Se HojeFezCalor OU OntemFezCalor Entao
13       Escreval("Fez calor ontem ou hoje")
14    FimSe
15
16    Se Nao HojeFezCalor Entao
17       Escreval("Hoje nao fez calor")
18    FimSe
19 Fimalgoritmo
```

Fonte: Elaborada pelo próprio professor

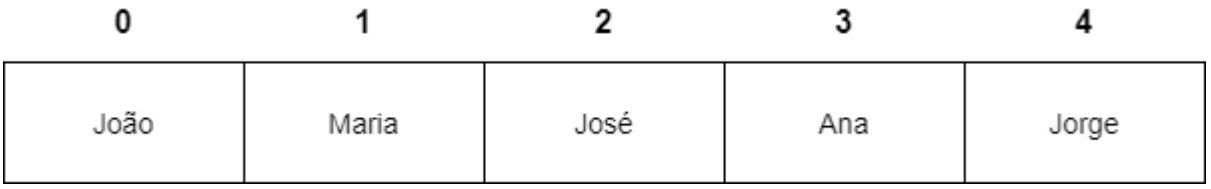
2.9. Listas

As listas são utilizadas para armazenar diversas informações de um mesmo tipo. Também são chamadas de arrays ou vetores. Por exemplo, para armazenar os

nomes dos alunos de uma escola, por exemplo, seria inviável criar uma variável para o nome de cada aluno. Nesse caso, são criadas listas que armazenam muitas informações do mesmo tipo. As listas permitem um fácil acesso a essas informações. Uma lista é criada como se fosse uma variável comum.

Os elementos são acessados na lista a partir de seu índice. Nos vetores, geralmente, o primeiro índice é o índice 0. Dessa forma, o primeiro elemento da lista possui índice 0, o segundo índice 1, o terceiro índice 2 e continua dessa maneira. A imagem abaixo ilustra um vetor de 5 posições do tipo caractere armazenado na memória, com seus índices de 0 a 4.

Figura 16 - Vetor



Fonte: Elaborada pelo próprio professor

A imagem a seguir ilustra como é feita a criação de um vetor em Portugol. A variável que irá receber o vetor é criada com o tipo “Vetor” e, entre colchetes, é colocado o intervalo de valores que esse vetor armazenará. Nesse exemplo, ele terá registros do índice 0 até 4, totalizando 5. Se tentarmos acessar um índice 5, irá gerar um erro, pois ele não foi criado. Nas linhas 5 a 9, são preenchidos valores nessas posições do vetor, e, nas linhas 11 a 15, esses registros são acessados e impressos na tela.

Figura 15 - Código utilizando Vetor

```
1  Algoritmo "Vetor"
2  Var
3     Nomes: Vetor[0..4] de Caractere
4  Inicio
5     Nomes[0] <- "João"
6     Nomes[1] <- "Maria"
7     Nomes[2] <- "José"
8     Nomes[3] <- "Ana"
9     Nomes[4] <- "Jorge"
10
11    Escreval(Nomes[0])
12    Escreval(Nomes[1])
13    Escreval(Nomes[2])
14    Escreval(Nomes[3])
15    Escreval(Nomes[4])
16 Fimalgoritmo
```

Fonte: Elaborada pelo próprio professor

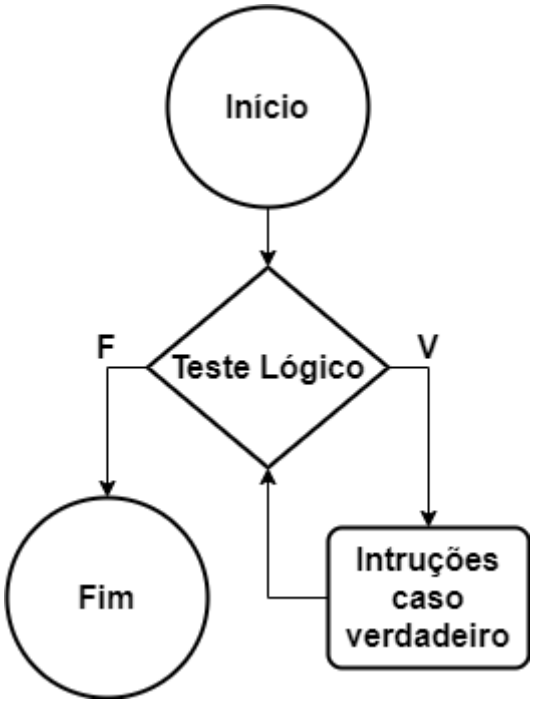
2.10. Estrutura de Repetição

As estruturas de repetição são utilizadas para executar uma ação várias vezes. Cabe à estrutura de repetição definir quando aquela repetição deve encerrar a execução. Geralmente, essas estruturas são muito utilizadas em conjunto com os vetores, para que seja possível realizar uma ação em todos os elementos.

A título de exemplo, elas são úteis para executar um trecho de código que é semelhante para todos os elementos de um vetor. Outro cenário de uso é executar um mesmo de código um certo número de vezes, ou até que determinada condição seja atingida.

Estudaremos duas estruturas de repetição, o “Enquanto” e o “Para”. O “Enquanto” executa determinado bloco de instruções enquanto determinada condição for verdadeira. O código se inicia com a verificação da condição, se for verdadeira, o bloco é executado, e depois volta para ser feita a verificação. Caso, ao verificar a condição, ela seja falsa, o bloco de instruções dentro do “Enquanto” não é mais executado, e o código segue a execução do restante. Um detalhe importante dessa estrutura é que precisamos fazer com que a condição se transforme em falsa em algum momento, pois, caso contrário, o código ficará executando até o programa travar por falta de memória, pois em nenhum momento ele chega ao fim. Por exemplo, se quisermos executar aquele trecho 10 vezes, uma abordagem comum é criar uma variável inteira que irá contar quantas vezes já foi executado, e, ao fim de cada execução, ela pode ser incrementada em uma unidade. Antes de cada execução, pode-se validar se a condição é menor do que 10, por exemplo, de forma que, quando o bloco tiver sido executado 10 vezes, a condição se tornará falsa, saindo, então, da execução do bloco. A imagem abaixo ilustra o fluxo do “Enquanto”.

Figura 15 - Fluxo do Enquanto



Fonte: Elaborada pelo próprio professor

A imagem abaixo ilustra um exemplo de um algoritmo utilizando a estrutura “Enquanto” no Portugol. Em frente à palavra “Enquanto”, é colocada a condição que vai ser verificada toda vez antes de entrar dentro do bloco que se inicia a partir do comando “Faca”, e se encerra em “FimEnquanto”. Neste exemplo, o bloco é executado 5 vezes. A variável “I” é utilizada como contadora para auxiliar na contagem de quantas vezes a execução foi feita. Na primeira execução, “I” é igual a 0, sendo, então, menor que 5. Ao fim da primeira execução, é utilizado o incremento para transformar “I” em 1. A validação é feita novamente, e “I” continua menor que 5. Isso é feito para “I” igual a 0, 1, 2, 3, 4. Na última execução, quando “I” ainda é 4, ele é incrementado para 5 e, então, a validação do enquanto é feita novamente, só

que, agora, “I” não mais menor que 5, pois ele é igual a 5. Dessa forma, o bloco dentro do “Enquanto” não é mais executado.

Figura 16 - Código do Enquanto

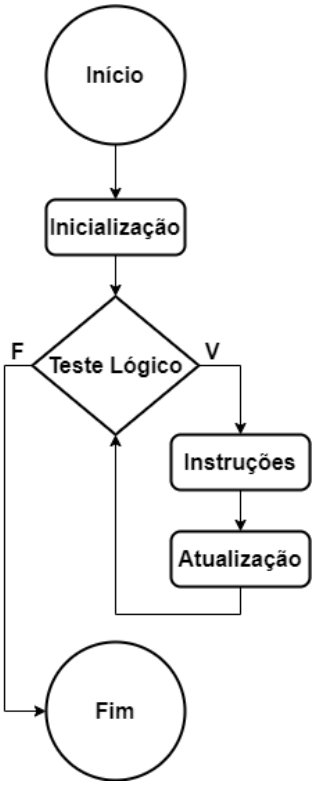
```
1  Algoritmo "Enquanto"
2  Var
3     Nomes: Vetor[0..4] de Caractere
4     I: Inteiro
5  Inicio
6     Nomes[0] <- "João"
7     Nomes[1] <- "Maria"
8     Nomes[2] <- "José"
9     Nomes[3] <- "Ana"
10    Nomes[4] <- "Jorge"
11
12    I <- 0
13    Enquanto I < 5 Faca
14       Escreval(Nomes[I])
15       I <- I + 1
16    FimEnquanto
17 Fimalgoritmo
```

Fonte: Elaborada pelo próprio professor

Assim como o “Enquanto”, o “Para” é uma estrutura de repetição muito utilizada e funciona de forma parecida. A diferença é que a variável utilizada para contar quantas vezes ele será executado é iniciada em frente ao comando “Para” e logo a frente já é colocado o intervalo em que ele será executado, tal como de 1 até 4. Nessa estrutura, não é preciso incrementar o contador manualmente, porque a própria estrutura se encarrega de incrementar ao fim de cada execução.

A imagem a seguir ilustra como é o fluxo de execução da estrutura “Para”. Inicialmente, a estrutura inicia a variável contadora, e depois é feito o teste lógico da condição definida. Caso seja verdadeiro, ele irá executar o bloco de instruções definidas e, em seguida, irá atualizar a variável contadora e fará o teste lógico novamente. Quando a condição não for mais verdadeira, ele irá parar de executar o bloco de código da estrutura e irá seguir executando o restante do código.

Figura 16 - Fluxo do Para



Fonte: Elaborada pelo próprio professor

A seguir, a imagem ilustra um exemplo de utilização da estrutura “Para”. O comando “Para” é escrito e, em frente a ele, o nome da variável que será utilizada como contadora, seu valor inicial em frente ao comando “De” e seu valor final em

frente ao comando “Ate”. O conteúdo que será executado fica no bloco delimitado pelas palavras “Faca” e “FimPara”.

Figura 17 - Código do Para

```
1  Algoritmo "Para"
2  Var
3     Nomes: Vetor[0..4] de Caractere
4     I: Inteiro
5  Inicio
6     Nomes[0] <- "João"
7     Nomes[1] <- "Maria"
8     Nomes[2] <- "José"
9     Nomes[3] <- "Ana"
10    Nomes[4] <- "Jorge"
11
12    Para I De 0 Ate 4 Faca
13       Escreval(Nomes[I])
14    FimPara
15 Fimalgoritmo
```

Fonte: Elaborada pelo próprio professor

2.11. Funções e procedimentos

Até aqui já utilizamos alguns procedimentos nativos do Portugol, como por exemplo “Escreva” e “Leia”. Veremos, agora, mais a fundo o que são funções e procedimentos, qual a diferença entre eles e como podemos criar os nossos próprios.

Podemos ver as funções e procedimentos como uma espécie de subprograma dentro do nosso programa principal. Elas têm o intuito de auxiliar na realização de tarefas. Dentro dele, podemos colocar código normal. Se não chamarmos este subprograma em nenhum momento a partir do nosso programa

principal, ele nunca terá seu código executado. Quando chamamos um subprograma, é realizado um desvio provisório no fluxo de execução do programa. O programa para de executar o programa principal, executa todo o código dentro do subprograma e, depois, volta a executar a partir do ponto que o subprograma foi chamado. Isso é muito útil, pois podemos chamar o mesmo subprograma várias vezes em locais diferentes no nosso programa principal, evitando, assim, que a gente tenha que escrever o mesmo código todas as vezes que formos utilizá-lo.

As funções e os procedimentos podem receber parâmetros, que são formas de customizar a execução do código. Quando chamamos a função “Escreva” e enviamos para ela o texto que queremos que seja impresso na tela, estamos enviando para ela um parâmetro, que será utilizado pelo procedimento. Eles podem receber vários parâmetros.

A diferença entre procedimento e função é que o procedimento não retorna nenhum valor, enquanto a função retorna. No caso do “Escreva”, por exemplo, ele é um procedimento, pois enviamos um parâmetro para ele, que realiza as devidas ações, e não retorna nenhum valor. Uma função recebe parâmetros da mesma forma e executa seu código normalmente, porém, no final, ele retorna algum valor, que pode ser armazenado pelo programa principal em uma variável. Um exemplo de uma função poderia ser uma função que calculasse a média dos valores de um vetor, ele poderia receber o vetor como parâmetro, realizar os cálculos e, depois, retornar o valor encontrado. Dessa forma, o programa principal poderia chamar essa função em vários pontos do seu código, evitando que o código de calcular a média seja feito todas as vezes que quisesse executar. Essa forma de programar facilita a reutilização do código feito, além de deixar o código mais legível e fácil de entender.

As funções e procedimentos conseguem enxergar as variáveis que foram criadas no programa principal, podendo utilizá-las normalmente. Chamamos as variáveis criadas no programa principal de variáveis globais, pois elas podem ser acessadas por todo o código. As funções e procedimentos também podem criar

variáveis internamente, dessa forma elas são chamadas variáveis locais, não sendo enxergadas fora da função ou procedimento que foram criadas.

A imagem abaixo ilustra um exemplo de utilização de uma função e de um procedimento. A função utilizada faz o cálculo da média de notas passada como parâmetro e retorna a média calculada. O procedimento recebe o nome do aluno e a média, e verifica se sua média foi maior que 60, para falar se ele foi aprovado ou reprovado. O procedimento não retorna nenhum valor, pois é a sua característica, enquanto a função retorna.

Figura 18 - Exemplo de Função e Procedimento

```
1  Algoritmo "FuncaoProcedimento"
2  Var
3     Aluno: Caractere
4     NotaA, NotaB, NotaC, NotaD, Media: Real
5  Inicio
6     Funcao CalculoMediaNotas(NotaA, NotaB, NotaC, NotaD: Real): Real
7     Var
8        Media: Real
9     Inicio
10       Media <- (NotaA + NotaB + NotaC + NotaD) / 4
11       Retorne Media
12    FimFuncao
13
14    Procedimento VerificarAprovacao(Aluno: Caractere; Media: Real)
15    Inicio
16       Se Media >= 60 Entao
17          Escreva(Aluno, " teve media de ", Media, " e foi aprovado")
18       Senao
19          Escreva(Aluno, " teve media de ", Media, " e foi reprovado")
20       FimSe
21    FimProcedimento
22
23    Escreva("Informe o nome do aluno: ")
24    Leia(Aluno)
25    Escreva("Informe a primeira nota: ")
26    Leia(NotaA)
27    Escreva("Informe a segunda nota: ")
28    Leia(NotaB)
29    Escreva("Informe a terceira nota: ")
30    Leia(NotaC)
31    Escreva("Informe a quarta nota: ")
32    Leia(NotaD)
33
34    Media <- CalculoMediaNotas(NotaA, NotaB, NotaC, NotaD)
35    VerificarAprovacao(Aluno, Media)
36 Fimalgoritmo
```

Fonte: Elaborada pelo próprio professor

Referências

VISUAL G3. **VisuAlg 3.0.** *[S.l.]*, 2017. Disponível em: https://visualg3.com.br/sobre-o-visualg/. Acesso em: 22 set. 2021.