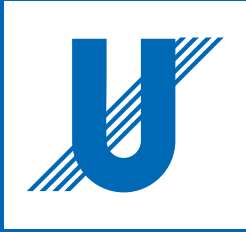




ALGORITMOS E LÓGICA DE PROGRAMAÇÃO

Prof. ESP. RONIE CESAR TOKUMOTO



Conteúdo **PROGRAMÁTICO**

- 005 ■ Aula 01: Conceitos Gerais de Programação
- 016 ■ Aula 02: Classificação de Algoritmos
- 024 ■ Aula 03: Análise de problemas e elaboração de soluções Computacionais usando Algoritmos
- 031 ■ Aula 04: Operadores e Expressões
- 041 ■ Aula 05: Tipos de Dados
- 048 ■ Aula 06: Entrada e Saída de dados
- 058 ■ Aula 07: Estruturas de Decisão
- 068 ■ Aula 08: Estruturas de Repetição
- 077 ■ Aula 09: Aninhamento de Estruturas
- 086 ■ Aula 10: Estruturas Homogêneas
- 094 ■ Aula 11: Estruturas Homogêneas Multidimensionais
- 105 ■ Aula 12: Estruturas de Dados Heterogêneas
- 109 ■ Aula 13: Sub-Rotinas
- 117 ■ Aula 14: Parâmetros
- 124 ■ Aula 15: Recursividade
- 131 ■ Aula 16: Manipulação de Arquivos

Introdução

O desenvolvimento de *softwares* é de grande importância para o mundo em constante evolução tecnológica, pois é aplicado, cada vez mais, em todas as áreas do conhecimento humano.

No entanto, o desenvolvimento de *software* em si não é todo o processo, pois para iniciar seu desenvolvimento é preciso antes realizar um trabalho de definição de requisitos e a modelagem pelo menos essencial de uma solução computacional.

As linguagens de programação servem como ferramentas para desenvolver produtos funcionais que possam atender às demandas solicitadas por clientes compondo assim um *software* propriamente dito.

Os algoritmos representam uma forma de se elaborar possíveis soluções computacionais para problemas reais servindo como ferramenta genérica que independe da linguagem de programação, e que cria uma possível solução simulada para servir de base para a elaboração de *software* em quaisquer linguagens de programação compatíveis com o tipo de *software* proposto.

Os algoritmos não possuem as limitações das linguagens e podem propor soluções de uma complexidade maior que a capacidade da tecnologia da informação de implementar uma solução funcional para o que é proposto em forma de algoritmo.

Muitas das tradicionais soluções para problemas clássicos são implementadas primeiramente como algoritmos, pois assim, pode-se imaginar quaisquer processos de forma a ignorar particularidades, sintaxe (forma como são escritos os comandos em cada linguagem de programação) e limitações que cada linguagem de programação possa ter.

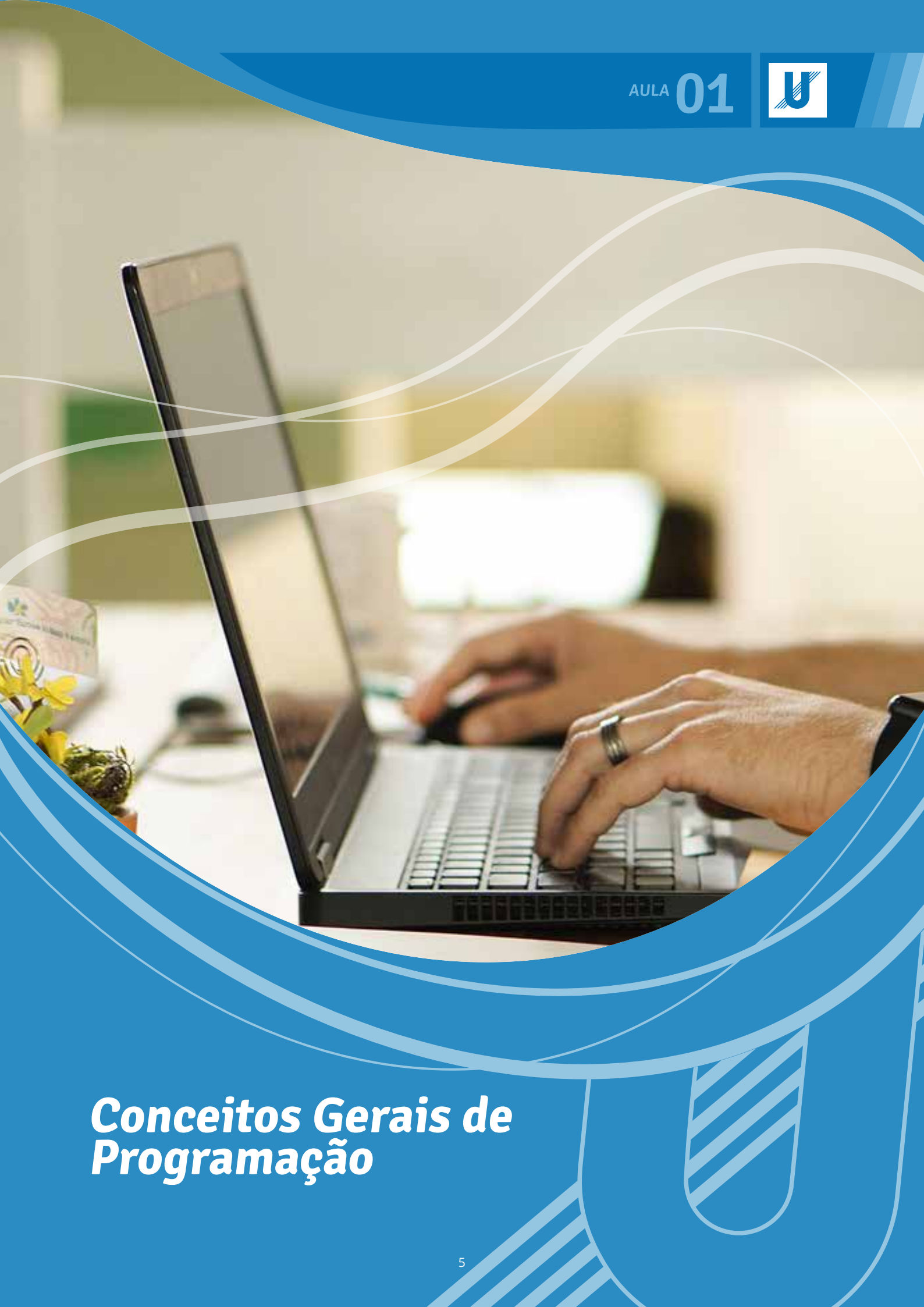
É possível escrever algoritmos sem o uso de sintaxe alguma, mas o uso de padrões na elaboração das instruções compostas por frases curtas que são formadas por palavras reservadas da língua para se referir a certos processos específicos e complementos para determinar especificidades do processamento a ser realizado, faz-se necessário. Para este material, as aulas são organizadas em função de conteúdos que sejam relevantes e complementares, pois o aprendizado todo do desenvolvimento de algoritmos não é composto por conteúdos isolados, mas por um conteúdo que se complementa a cada aula e que ao final se torna acumulativo.



Assim, a compreensão das primeiras aulas é fundamental para a sequência dos estudos, e por isso, é interessante o completo aprendizado de cada aula antes de seguir para a aula seguinte, pois conteúdos de aulas anteriores podem ser considerados pré-requisitos para aulas posteriores na compreensão de alguns conceitos, principalmente, de exemplos.

Espero que este material possa agregar alguns conceitos importantes para o desenvolvimento de *software*, mas o principal objetivo é a ajuda com a formação de uma boa capacidade de formulação de soluções em forma de algoritmo para problemas a serem resolvidos, assim como a capacidade de interpretação de algoritmos criados por outros e uma boa capacidade de raciocínio lógico essencial para trabalho na área de tecnologia da informação.

Bons estudos!



Conceitos Gerais de Programação

Programar é um ato que depende não só das habilidades de um profissional de desenvolvimento de *software* ou uma equipe de desenvolvedores, pois a complexidade dos problemas a serem resolvidos varia de pequenos componentes de um *software* a problemas computacionais complexos ou o desenvolvimento de grandes sistemas inteiros.

Isso faz com que sejam necessários outros profissionais, além de programadores, para que se possa compreender o problema a ser resolvido e a estruturar uma solução a ser desenvolvida a partir de requisitos identificados no problema, com prazo e orçamento disponíveis, pois estes dois últimos aspectos também afetam muito as decisões de como pode ser produzido um *software* para o que é pedido pelo cliente e se é possível produzi-lo de forma aceitável.

Uma das etapas do desenvolvimento de uma solução computacional pode talvez ser a mais importante, pois é nela que a solução propriamente dita é elaborada já com um olhar computacional, e esta etapa pode ser uma etapa real do processo de desenvolvimento ou algo já tão desenvolvido que pode ser praticamente pensado automático e instantaneamente paralelo ao desenvolvimento de um código em determinada linguagem de programação.

Este desenvolvimento de uma solução intermediária entre a modelagem inicial de uma solução computacional e seu desenvolvimento de código propriamente dito, chamado de algoritmo, simboliza representações em idioma local de uma possível solução computacional, mais visando a compreensão lógica de uma solução proposta e dos mecanismos de programação que podem ser utilizados na solução.

Independentemente da forma como será elaborada uma solução, o mais importante nesta etapa do aprendizado do desenvolvimento de *software* é a lógica utilizada para solucionar problemas e compreender a estrutura de um código, lembrando que existem diferentes formas de se estruturar códigos devido a diferenças que existem nos chamados paradigmas que as linguagens de programação podem utilizar em suas implementações, além das diferenças de sintaxe na construção de instruções.

Os paradigmas se referem a diferentes formas de se programar como na programação estruturada ou orientada a objetos, por exemplo, e a chamada sintaxe se refere a como cada instrução é implementada em cada linguagem, pois em algumas há muitas semelhanças, mas em outras, a sintaxe varia muito em função da diferença de paradigma e da forma como foi implementada a linguagem.

Formas de representação de soluções de problemas

Além do uso de algoritmos, é possível representar possíveis soluções computacionais de problemas utilizando narrativas em idioma local em forma de frases curtas contendo todos os principais processos que ocorrem na execução da solução. Observe um exemplo de descrição narrativa na imagem 1.

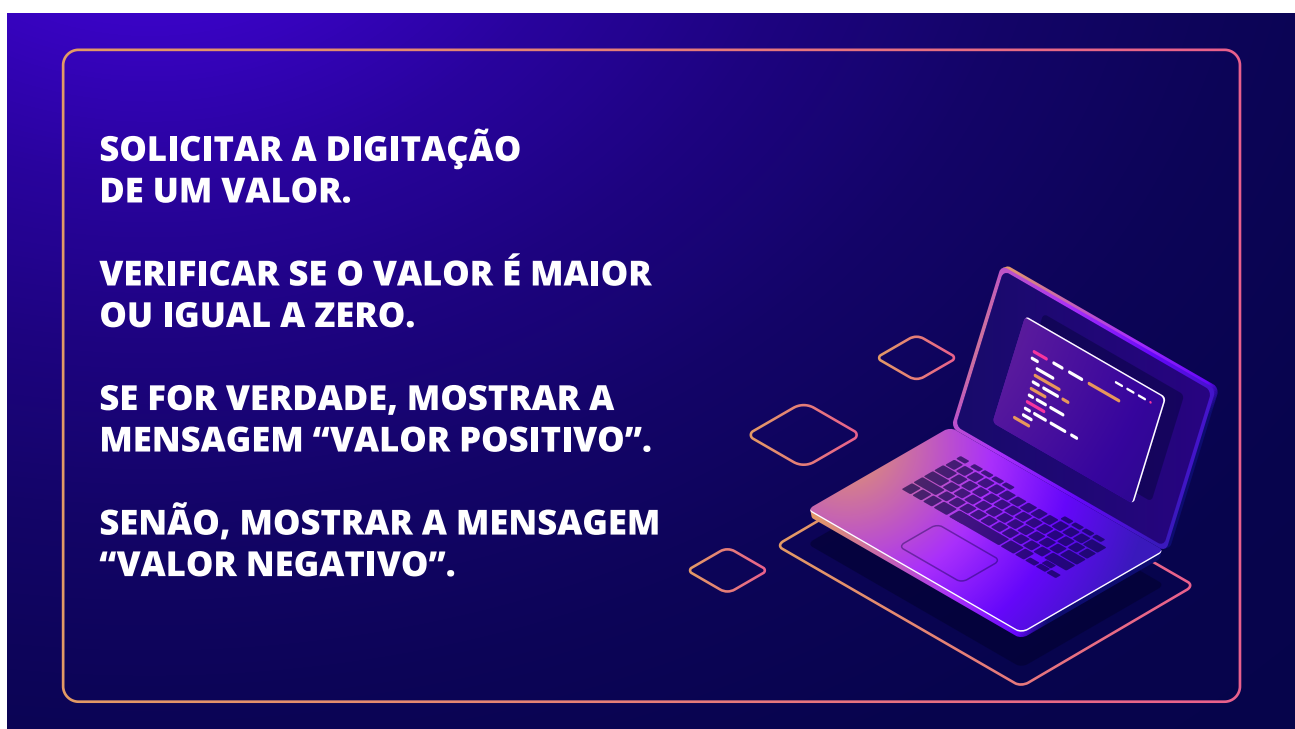


Imagem 1: Exemplo de narração descritiva | Fonte: O autor.

Neste exemplo da imagem 1, a descrição narrativa traz uma alternativa de solução para a identificação de valores positivos ou negativos que podem ser inseridos como entradas de dados em uma solução computacional. Nesta alternativa é possível perceber o uso de frases curtas em idioma local que são facilmente compreendidas por pessoas, mas não representa exemplo de código em linguagem de programação *software* propriamente dito.

Outra forma de solução utiliza um diagrama bastante específico chamado de fluxograma que tem por função essencial, demonstrar a sequência de ações que ocorre durante uma execução, seguindo um fluxo lógico que pode ser alterado, se

necessário.

Figuras como retângulos podem representar ações e losangos decisões a serem tomadas pela aplicação a partir de dados disponíveis, assim como linhas podem indicar o fluxo de execução entre os demais símbolos, e círculos podem unir linhas que se encontram para que seus fluxos continuem por um mesmo caminho. Observe o exemplo da imagem 2

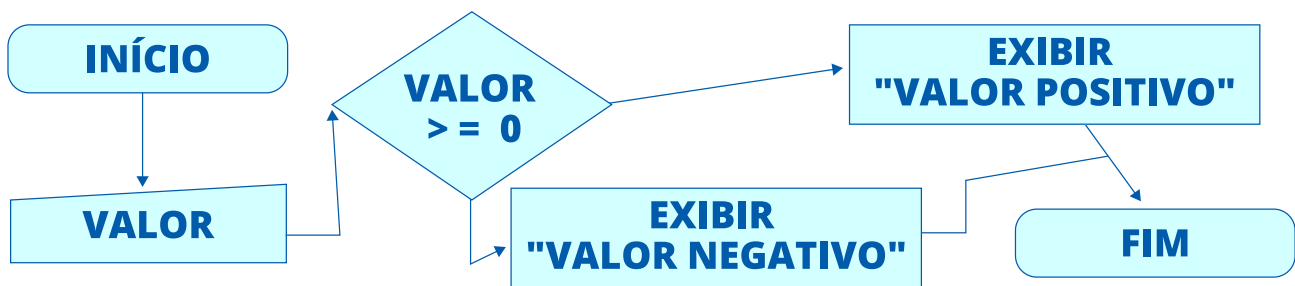


Imagem 2: Exemplo de fluxograma para analisar valor informado por usuário | Fonte: O autor.

Em relação ao que é mostrado acima, a solução lógica apresentada se refere ao mesmo problema desenvolvido na imagem 1, mas de uma forma diferente e mais intuitiva, baseada em formas que são padronizadas em significado, mas variando em sua aplicação e texto inserido nas mesmas para complementação do sentido de cada forma no diagrama.

Uma terceira forma de indicar uma solução computacional ainda não executável a princípio é por meio de algoritmos escritos usando pseudocódigos em uma linguagem conhecida como portugol em nosso país, e que podem ser escritos à mão de forma simplificada, usando pouca quantidade de texto em cada figura, mas possuindo alto poder de elaboração de soluções lógicas para problemas. Observe o exemplo na imagem 3.


```
INÍCIO  
DECLARE  
  INTEIRO: VALOR;  
  LEIA (VALOR);  
  SE (VALOR > = 0)  
    ENTÃO  
      ESCREVA ("VALOR POSITIVO");  
    SENÃO  
      ESCREVA ("VALOR NEGATIVO");  
  FIMSE;  
FIM.
```

Imagem 3: Exemplo de algoritmo | Fonte: O autor.

Neste algoritmo de exemplo na imagem 3, também se mostra uma alternativa para o mesmo problema proposto de identificação de um valor inserido por um usuário como positivo ou negativo, mas nesta solução existe um maior nível de detalhamento em função de uma maior proximidade com uma solução computacional.

Vários dos elementos utilizados no exemplo representam palavras que não podem ser utilizadas senão para o propósito ao qual se prestam, chamadas de palavras reservadas e que serão detalhadas ao longo dos estudos, e outros elementos que junto com estas palavras reservadas, constroem comandos em uma linguagem mais próxima das linguagens de programação, mas ainda mantendo uma leitura de fácil compreensão.

Um detalhe importante que deve ser colocado logo de início, é que qualquer uma das formas citadas de elaboração de soluções computacionais para problemas pode variar de uma fonte de pesquisa para outra. No exemplo da imagem 3, foi seguida a padronização de sintaxe do autor Forbellone, mas há outras formas de se escrever algoritmos variando, principalmente o uso de símbolos e algumas palavras reservadas, por exemplo.

CONECTE-SE

Machine Learning (Aprendizado de Máquina) representa um dos diversos assuntos da área de TI e necessita de algoritmos muito bem elaborados para tratar de dados e como utilizá-los para que um *software* possa evoluir por meio de novas construções que possam ampliar uma base de informações existente.

Acesse o link: [Disponível aqui](#)

Estrutura de um algoritmo

Os algoritmos representam conjuntos de instruções organizadas de forma a oferecer soluções lógicas para problemas computacionalmente possíveis. Um algoritmo deve ser claro e o mais otimizado possível, pois é apenas uma representação de uma solução, mas não a própria ainda, pois algoritmos precisam ser convertidos em alguma linguagem de programação desejada para se tornar um *software* e poder ser executado em um *hardware*.

Em geral, um algoritmo possui uma parte inicial em que são definidas estruturas de dados para armazenamento dos dados a serem processados na parte principal do algoritmo, e em geral, as estruturas de dados possuem tipos de dados aceitos definidos nessa parte inicial.

Após a definição das estruturas de dados, geralmente inicia-se a parte principal do algoritmo contendo todas as instruções necessárias para se solucionar problemas com base em entradas e saídas de dados e estruturas de controle, por exemplo. Observe o exemplo da imagem 4.

```
INÍCIO  
DECLARE  
    INTEIRO : VALOR;  
    LEIA (VALOR);  
    SE (VALOR >= 0)  
        ENTÃO  
            ESCREVA ("VALOR POSITIVO");  
        SENÃO  
            ESCREVA ("VALOR NEGATIVO");  
    FIMSE;  
FIM.
```

Imagem 4: Exemplo de algoritmo | Fonte: O autor.

No exemplo da imagem 4, temos o mesmo exemplo da imagem 3, mas com as duas seções básicas de um algoritmo separadas. Nas duas primeiras linhas é iniciado o algoritmo e definida a estrutura de dados a ser utilizada, e na segunda metade, o restante do algoritmo com a parte que desenvolve a solução em si, lembrando que cada algoritmo proposto para diferentes problemas geralmente possui diferentes instruções e estruturas de dados.

Pode haver outras partes em um algoritmo e estas serão aos poucos explicadas e exemplificadas para que se conheça mais a fundo a elaboração de algoritmos e a construção lógica de instruções, estruturas de controle, etc.

ABORDAGEM PRÁTICA



Uma prova da importância dos algoritmos está se concretizando em robôs autônomos e cada vez mais “inteligentes”, sendo capazes de tomar decisões mais complexas e de evoluir cada vez mais dentro de suas especificidades.

Existem *softwares* de auxílio funcionamento como atendentes virtuais em empresas para atendimentos mais simples e padronizados, assim como existem robôs que conseguem simular certos comportamentos humanos ou animais por meio de algoritmos capazes de realizar certo nível de “interpretação”.

A ideia é sempre observar as evoluções proporcionadas pela TI para si mesma ou para as demais áreas do conhecimento humano, e lembrar que para este desenvolvimento ocorrer, geralmente existem algoritmos utilizados para aplicar o que é proposto para gerar evoluções.

Palavras reservadas

As palavras reservadas servem como comandos para que cada ação a ser definida em um algoritmo seja padronizada através do uso de palavras definidas como padrão e cada uma delas tenha uma chamada sintaxe que define os elementos necessários e opcionais para a composição de instruções utilizando estas palavras reservadas como elemento principal de cada instrução.

Estas palavras são utilizadas também em todas as linguagens de programação, pois esta padronização, além de facilitar o aprendizado e uso na elaboração de algoritmos, é necessária para que compiladores e interpretadores possam compreender as ações e a lógica que foi definida pelas instruções em uma linguagem de programação qualquer.

Observe a tabela 1 que traz uma lista de palavras reservadas a serem utilizadas neste material e uma breve descrição de suas funcionalidades.

PALAVRA RESERVADA	DESCRIÇÃO
ABRA	AÇÃO PARA ARQUIVO
ARQUIVO COMPOSTO DE	DECLARAÇÃO DE ARQUIVO
ATE	FINAL DE COMANDO DE REPETIÇÃO
ATÉ	COMPLEMENTO DE COMANDO DE REPETIÇÃO
ATRIBUTO	CARACTERÍSTICA DE UM OBJETO
CARACTERE	TIPO DE DADO
CASO	COMANDO CONDICIONAL
CLASSE	DESCRIÇÃO PARA GERAR ABSTRAÇÃO DE OBJETO
CONJUNTO	TIPO DE DADO
COPIE	AÇÃO PARA ARQUIVO
DE	COMPLEMENTO DE COMANDO DE REPETIÇÃO
DECLARE	SEÇÃO DE DECLARAÇÃO DE VARIÁVEIS
ELIMINE	AÇÃO PARA ARQUIVO
ENQUANTO	COMANDO DE REPETIÇÃO
ENTÃO	COMPLEMENTO DE COMANDO CONDICIONAL
ESCREVA	SAÍDA DE DADOS
FAÇA	COMPLEMENTO DE COMANDO DE REPETIÇÃO

FECHE	AÇÃO PARA ARQUIVO
FIM	FINAL DE BLOCO DE INSTRUÇÕES
FIM_REGISTRO	FINAL DE DECLARAÇÃO DE REGISTRO
FIMCASO	FINAL DE COMANDO CONDICIONAL
FIMENQUANTO	FINAL DE COMANDO DE REPETIÇÃO
FIMPARA	FINAL DE COMANDO DE REPETIÇÃO
FIMSE	FINAL DE COMANDO CONDICIONAL
FUNÇÃO	INÍCIO DE SUB-ROTINA
GUARDE	AÇÃO PARA ARQUIVO
INÍCIO	INÍCIO DE BLOCO DE INSTRUÇÕES
INTEIRO	TIPO DE DADO
LEIA	ENTRADA DE DADOS
MÉTODO	AÇÃO QUE PODE SER EFETUADA POR UM OBJETO
PARA	COMANDO DE REPETIÇÃO
PASSO	COMPLEMENTO DE COMANDO DE REPETIÇÃO
PROCEDIMENTO	INÍCIO DE SUB-ROTINA
REAL	TIPO DE DADO
REGISTRO	INÍCIO DE DECLARAÇÃO DE REGISTRO

REPITA	COMANDO DE REPETIÇÃO
SE	COMANDO CONDICIONAL
SEJA	COMPLEMENTO DE COMANDO CONDICIONAL
SENÃO	COMANDO CONDICIONAL
TIPO	COMPLEMENTO DE COMANDO DE REGISTRO

Tabela 1: Exemplo de algoritmo | Fonte: O autor.

Nesta tabela 1, foram incluídas as palavras reservadas para os estudos neste material, e as palavras utilizadas seguem como base os autores Manzano e Forbellone que em seus livros, utilizaram conjuntos semelhantes de palavras reservadas, mas parte delas é encontrada apenas no material de Forbellone relativo à parte sobre arquivos, mas o restante das palavras foi baseado no que foi descrito por Manzano.

PARA GABARITAR



Conhecer as palavras reservadas de uma linguagem não é uma obrigação, mas se torna cada vez mais automática a escolha de cada palavra a utilizar em cada situação, e quais os elementos adicionais utilizáveis com cada palavra à medida que se desenvolvem os estudos na área e se realiza a prática de desenvolvimento de algoritmos para resolver diferentes tipos e complexidades de problemas.

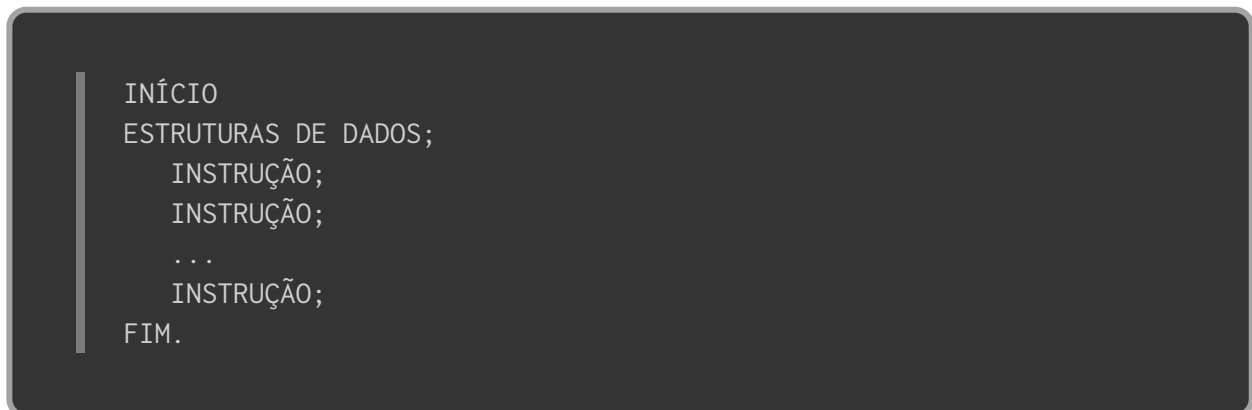


Classificação de Algoritmos

Algoritmos são aparentemente simples por utilizarem idioma comum e não necessitar de ferramenta alguma além de um editor de textos simples para ser escrito, além do que pode também ser escrito à mão, pois seu intuito é de propor ideias e soluções, mas não de programá-las.

Por esta característica, o nível de detalhamento dos processos pode ser baixo e apenas as principais ideias serem colocadas na elaboração, pois a codificação em alguma linguagem de programação precisa ser detalhada o suficiente para implementar todos os requisitos necessários e demais elementos de forma a compor um *software* que seja funcional e atenda às expectativas do cliente.

Geralmente, a ideia na concepção de um algoritmo é que seu pseudocódigo seja estruturado de forma que a execução das instruções seja realizada da mesma forma que a leitura de uma página de texto, de forma sequencial, desde o início até o fim da página. Observe a imagem 5 para que se tenha uma ideia da estrutura de um algoritmo com instruções sequenciais.



```
INÍCIO
ESTRUTURAS DE DADOS;
  INSTRUÇÃO;
  INSTRUÇÃO;
  ...
  INSTRUÇÃO;
FIM.
```

Imagem 5: Exemplo de estrutura de algoritmo imperativo | Fonte: O autor.

Neste exemplo da figura 5, a estrutura do algoritmo inicia por uma palavra reservada **INÍCIO**, tendo em seguida a declaração das estruturas de dados a serem utilizadas no mesmo.

Em seguida, inicia-se uma sequência variável de instruções a serem executadas sequencialmente no algoritmo, podendo estas ser responsáveis por ações de entrada ou saída de dados, e pelo processamento destes dados dentro da proposta de funcionalidade do algoritmo.

Esta forma sequencial e seriada de execução de algoritmos, por ser a mais comum, também acaba sendo a mais indicada para iniciantes, pois além de uma lógica mais simples de compreensão, também contém muito mais conteúdo em meios de pesquisa como livros e a *web*.

Os chamados paradigmas de linguagens de programação que representam formas distintas de programação possuem também maneiras de serem representados em forma de algoritmo, pois esta representação usando pseudocódigo é limitada apenas pela capacidade criativa de quem propõe soluções utilizando este recurso.

Esta forma sequencial de escrita de pseudocódigos está relacionada com um paradigma chamado imperativo que predomina a construção de código de forma sequencial do início ao final de um código, utilizando apenas as estruturas básicas da linguagem ou neste caso, do pseudocódigo para a implementação.

Outro tipo de algoritmo é constituído por partes parcialmente independentes de pseudocódigos estruturados em forma de sub-rotinas que servem para que um algoritmo seja composto por partes independentes entre si de preferência, e que possam se comunicar de forma integrada, oferecer uma solução computacional completa que aparentemente é executada como um algoritmo sequencial.

Os algoritmos podem ser inseridos em uma segunda categoria de paradigmas de programação chamados de estruturados por permitirem a subdivisão de um pseudocódigo em partes que possam se comunicar por meio da troca de dados entre estas sub-rotinas. Observe a imagem 6 para um exemplo de estrutura de algoritmo estruturado.

```
PROCEDIMENTO TESTE()  
INÍCIO  
    INSTRUÇÃO;  
    ...  
    INSTRUÇÃO;  
FIM.  
INÍCIO  
    INSTRUÇÃO;  
    ...  
    TESTE();  
    ...  
    INSTRUÇÃO;  
FIM.
```

Imagem 6: Exemplo de estrutura de algoritmo estruturado | Fonte: O autor.

No exemplo da imagem 8, ao invés de iniciar o algoritmo pela palavra reservada **INÍCIO** como de costume, utiliza-se a palavra reservada **PROCEDIMENTO** para indicar o início de uma sub-rotina chamada **TESTE()** que contém instruções para alguma finalidade específica e que pode ser chamada a qualquer momento pelo trecho principal de pseudocódigo do algoritmo.

A parte principal do algoritmo que é por onde inicia sua execução possui instruções normais após a palavra reservada **INÍCIO**, mas em determinado momento, chama o procedimento **TESTE()** para que este realize um procedimento necessário para a sequência da execução do código.

Esta divisão do algoritmo em parte principal e sub-rotina contendo diferentes trechos de pseudocódigo são a base da programação estruturada, e esta é uma das formas mais utilizadas de programação no mercado.

PARA GABARITAR



Compreender a estrutura de um algoritmo ou código de uma linguagem de programação é o primeiro passo essencial para se aprender a programar, mas antes ainda, é preciso aprender a interpretar problemas e deles obter informações capazes de auxiliar no início do esboço da lógica de uma provável solução computacional.

Existem algoritmos que tratam de uma lógica mais complexa e evoluída do uso de sub-rotinas capazes de chamar a si mesmas de forma iterativa ou repetitiva, também chamada de recursiva, em que existe um controle da quantidade de iterações baseado em uma condição que muitas vezes envolve valores numéricos que ao serem atingidos, encerram as sucessivas chamadas à sub-rotina e às seguidas repetições.

O paradigma estruturado citado anteriormente permite uso desta técnica, mas há outros paradigmas que permitem uso deste recurso no desenvolvimento de *software* como o paradigma funcional e o paradigma orientado a objetos.

Esses paradigmas representam maneiras mais específicas e de lógica mais complexa para elaboração de algoritmos e não são as formas mais populares de uso. Este tipo de recurso para lidar com sub-rotinas será melhor detalhado mais adiante em uma aula posterior.

ABORDAGEM

PRÁTICA



Praticar programação não representa apenas digitar códigos prontos e executá-los. Realizar ajustes quando erros ocorrem de forma a demonstrar que a lógica da solução é compreendida e a sintaxe do pseudocódigo ou linguagem de programação é compreendida, representa que se possui preparo e raciocínio lógico necessários para o correto desenvolvimento de uma solução para um problema.

Se esta solução é escrita da maneira mais correta ou eficiente, isto depende muito da complexidade da solução e experiência em programação do desenvolvedor, mas isto se alcança com a experiência de desenvolver *softwares* variados.

Um outro paradigma chamado de funcional se baseia em sub-rotinas que executam processos como se fossem funções matemáticas em que se evita, ao máximo, mudanças de estado ou dados dinâmicos que são características comuns nos paradigmas imperativo e estruturado que utilizam muitas estruturas de dados variáveis e estados que mudam o tempo todo na execução.

Por ser uma forma de programação mais específica e não tradicional será deixada de lado neste material, sabendo que sua relevância está aumentando em diversas linguagens de programação, devido a algumas características deste paradigma, como

facilidade na estruturação da semântica de seus códigos, e por serem linguagens com uma escrita bem específica e compacta, facilita a busca por erros quando necessário.

Outro paradigma bastante popular é chamado de orientação a objetos e possui pontos em comum com a programação estruturada, mas com diferenciais em sua metodologia que diferenciam os paradigmas como a chamada abstração que converte problemas reais em classes que possuem as principais características do problema e as soluções computacionais são estruturas semelhantes a sub-rotinas com uma maior independência que no paradigma estruturado.

Esta forma de programação cria códigos mais independentes entre si e que podem ser integrados com certa facilidade, devido a uma característica do paradigma que busca tornar seus componentes independentes ao máximo para que seja mais bem realizado o chamado reuso de código. Observe o exemplo da imagem 7 para um auxílio nos estudos deste paradigma.

```
CLASSE EXEMPLO
  ATRIBUTO DADO;
  ...
  MÉTODO PROCESSA();
  ...
EXEMPLO OBJ;
```

Imagem 7: Exemplo de estrutura de algoritmo orientado a objetos | Fonte: O autor.

Neste exemplo contendo um trecho de pseudocódigo baseado na ideia de programação orientada a objetos, uma classe chamada EXEMPLO é declarada contendo um atributo DADO de exemplo, e um método PROCESSA() funcionando ambos de forma semelhante a variáveis e funções, mas de forma diferente, seguindo os princípios desse paradigma.

Neste pseudocódigo, exemplo da imagem 9, algumas palavras reservadas foram utilizadas, e foram definidas para este estudo como sem a utilização de algum padrão conhecido, e assim, as palavras CLASSE, ATRIBUTO e MÉTODO foram definidas para este material como representativas de elementos do paradigma de orientação a objetos.



Outra forma de se estruturar algoritmos se baseia em regras da lógica matemática para avaliar expressões usando dados imutáveis para obtenção de resultados verdadeiros ou falsos, ou para construção de novas regras a partir das existentes.

Este método de desenvolvimento de código é bastante específico e utilizado em poucas linguagens de programação para geração de *software* voltado para o desenvolvimento de sistemas especialistas que trabalham com relações entre dados para obter novas informações tendo influências da área de inteligência artificial no desenvolvimento deste paradigma.

Esses chamados paradigmas de programação são a base de todas as linguagens de programação, e há mais paradigmas diferentes existentes. Com o avanço da tecnologia da informação e o desenvolvimento de novos tipos de *software* para estas tecnologias, novos paradigmas também surgem para adequar a programação às novas necessidades.

Além dos paradigmas que representam diferentes formas de se elaborar *software* para resolução de problemas computacionais, existem as diferentes técnicas para se elaborar algoritmos que possuem suas características particulares e se adequam a tipos diferentes de problemas ou trazem formas diferentes para solucionar os mesmos problemas.

Uma forma de se solucionar problemas é fracioná-los em problemas menores, e esse fracionamento pode se repetir com as partes já reduzidas até que cada parte do problema seja simples o suficiente para ser facilmente resolvido e que uma solução possa ser desenvolvida, mantendo relação com as demais partes para que possam ser integradas posteriormente.

Um exemplo desta técnica seria, por exemplo, a separação de cada parte relativa a cada cálculo de um algoritmo de implementação de uma calculadora para que todas estas partes organizadas em estruturas chamadas de sub-rotinas, por exemplo, possam ser acessadas sempre que necessário pelo restante do pseudocódigo, mas mantendo-as como estruturas mais independentes dentro do algoritmo como um todo.

Algoritmos podem ser utilizados para resolver outros problemas, ou terem processamento realizado por um algoritmo reaproveitado em outro algoritmo para reduzir o uso de recursos e ganho de tempo e desempenho geral.



Existem algoritmos bastante específicos e até muito conhecidos para resolver problemas e subproblemas bastante comuns a problemas maiores como a ordenação de dados, a maximização ou minimização de resultados obtidos, processamento matemático ou físico, etc.

Algoritmos conhecidos como algoritmos de ordenação (*bubble sort*, *merge sort*, etc.), algoritmos de estruturas chamadas de grafos que se baseiam na ligação de pontos por arestas que podem ser aplicados a muitos problemas reais, etc.

Alguns algoritmos mais complexos envolvem heurística que representam soluções cognitivas e não racionais para processamento de escolhas mais rapidamente e ignorando detalhes que em outros tipos de algoritmos seriam relevantes.

Assim, os algoritmos que se baseiam nos tipos de funcionalidades devem possuir: problemas a resolver e técnicas a serem aplicadas em alguma linguagem baseada em um ou mais paradigmas de programação para gerar soluções computacionais para problemas reais.

Algoritmos podem ser simples e não necessitar de recursos muito específicos para resolver problemas lógicos complexos, ou precisar de recursos muito completos para atender demandas muito complexas e necessitar de muito mais tempo para serem implementados por completo.



Análise de problemas e elaboração de soluções Computacionais usando Algoritmos

Problemas são a razão de serem desenvolvidos os algoritmos. Desde problemas menos complexos como um cálculo pequeno ou uma simples entrada e saída de dados com o mínimo de processamento, até sistemas complexos compostos de vários módulos que realizam processamento de grandes volumes de dados ou atividades complexas como jogos, cálculos avançados, etc.

Algoritmos representam meios de se programar dispositivos diversos que podem ser simples como autômatos mecânicos programáveis e circuitos elétricos simples ou dispositivos mais complexos que possuam sistemas operacionais sendo executados e que sobre eles sejam executados os demais *softwares* desenvolvidos a partir de algoritmos.

CONECTE-SE



A programação de autômatos foi uma das formas de se materializar a teoria das linguagens formais desenvolvida na década de 1950, e que permitiu o desenvolvimento de aplicações baseadas em análise léxica e sintática para o desenvolvimento de linguagens artificiais precursoras das linguagens de programação utilizadas até os dias de hoje.

Acesse o link: [Disponível aqui](#)

A ideia de todo *software* desenvolvido a partir de um algoritmo é a de realizar uma série de passos para solucionar problemas, atendendo a requisitos definidos como necessários para que o *software* seja considerado adequado e realmente funcional, pois não basta apenas um *software* ser funcional sem realizar o que dele se espera.

Há *software* que é desenvolvido especificamente para determinado *hardware* e com finalidade bastante específica desde o princípio quando Ada Lovelace (Augusta Ada King, Condessa de Lovelace), matemática e escritora inglesa, que na década de 1840 traduziu um artigo de um engenheiro militar italiano. Também escreveu anotações adicionais que foram consideradas o primeiro programa de computador da história.

Com a evolução do *hardware*, e a expansão do que era possível realizar com as diversas variações do mesmo, o desenvolvimento de *software* para estes também foi sendo modificado e toda uma nova indústria surgiu para atender à demanda crescente de desenvolvimento de *software*. Desde a grande expansão tecnológica ocorrida em função da maior popularização do *hardware* proporcionada por empresas como a IBM e Apple, que desenvolveram equipamentos com bom desempenho, com tamanho e preço acessíveis a partir da década de 1980 quando surgiu o computador pessoal principalmente.

Assim, com equipamentos cada vez mais potentes e menos específicos, o *software* passou por grandes mudanças e com o surgimento dos sistemas operacionais capazes de serem executados nesses computadores de propósito geral, o desenvolvimento de *software* passou a um nível mais alto em que não era mais primordial que o *software* conversasse com o *hardware* diretamente, e sim, com o sistema operacional encarregado da intermediação desta interação.

Da mesma forma que o *hardware* foi sendo inserido em todas as atividades humanas, desde a indústria até o lazer, foi sendo então desenvolvido *software* para atender a toda esta nova demanda de produtos que permitissem que computadores com sistemas operacionais instalados pudessem ser utilizados para facilitar e automatizar tarefas antes realizadas apenas de forma manual ou mecânica.





Uma mudança importante que ocorreu foi uma adaptação dos programadores e linguagens de programação aos sistemas operacionais, pois estes passaram de uma interface totalmente textual e monocromática ao uso de cores num primeiro momento e unidades de discos magnéticos muito mais eficientes e rápidos em relação ao uso de rolos de fita ou cartões perfurados, além de quantidades maiores de memória temporária para permitir a execução de *softwares* mais complexos.

Isto já gerou muitas mudanças na forma como se desenvolvia *software* e permitiu aplicações com maior complexidade computacional, pacotes de funcionalidades como o editor de textos Microsoft Word, desenvolvido na década de 1980 ou jogos que iniciaram uma nova era com evoluções muito rápidas em sua complexidade, em comparação aos existentes antes dos computadores pessoais.

Outra mudança impactante no desenvolvimento de *software* ocorreu com o desenvolvimento de interfaces gráficas para os principais sistemas operacionais existentes como os desenvolvidos por Apple, primeiramente, e Microsoft logo em seguida, por exemplo.

Já se pensava em interfaces gráficas para usuário (GUI) desde décadas anteriores como a desenvolvida pela XEROX, na década de 1970, mas com a Apple, na década de 1980, houve uma grande evolução do projeto original da XEROX usando um dispositivo apontador que foi implementado para uso nos computadores Lisa e Macintosh.

A GUI para o sistema MS-DOS surgiu em 1985 e se popularizou, na década de 1990, com as versões 3.0 e 3.11 que revolucionaram o mercado de computadores pessoais, antes dominado pela Apple, mas que passou a ter os computadores da IBM com sistema operacional MS-DOS e GUI Windows como um forte concorrente.

O desenvolvimento de *software* passou a ser voltado em grande parte, a produtos que se baseassem nestas interfaces gráficas, abrindo novos nichos de mercado e permitindo uma explosão de empresas de desenvolvimento e elevando a inserção da informática em todas as áreas de mercado e nos lares por todo o mundo em uma velocidade maior.

Softwares para jogos e aplicações comerciais começaram a ser produzidos em uma escala maior, permitindo que muitas empresas, novas no mercado, surgissem e outras, já mais estabilizadas, se tornassem verdadeiros gigantes na tecnologia.



Os computadores e sua infinidade de *softwares* desenvolvidos para aplicações diversas se tornaram importantes no processo de melhoria dos processos.

Codificação e Execução de Programas

O processo de produção de *software* envolve uma série de etapas, desde a chamada modelagem de *software*, que trata dos contatos iniciais da elaboração de uma solução computacional para problemas, até a documentação inicial a que a solução se propõe a ter de funcionalidades para atender às demandas solicitadas.

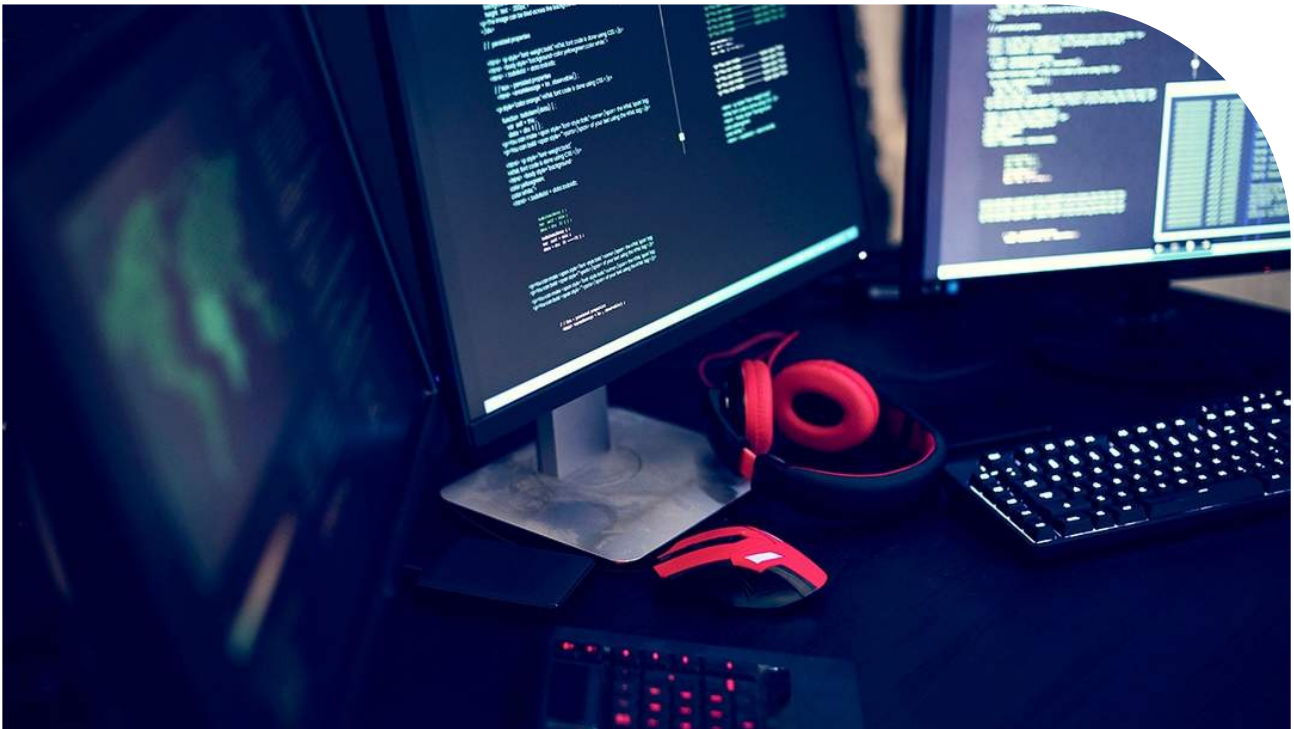
Uma das formas de se desenvolver código em linguagens de programação se baseia na ideia de que o código deve ser pensado de forma que sua execução seja baseada num seguimento de instruções que vão sendo sequencialmente executadas de forma direta por um *software* encarregado de ler cada instrução e executar as instruções de forma direta sem que nenhuma conversão intermediária seja necessária no processo de execução do código fonte digitado.

Assim, para que seja possível a execução de programas de forma interpretada, é necessário que este programa seja digitado em forma de texto simples e gravado da forma adequada.

Depois, é necessário que a plataforma em que o mesmo será executado contenha um *software* para interpretação que pode ser próprio para tal função, como ocorre em navegadores para *Internet* que são criados para interpretar páginas *web* escritas em linguagem de marcação HTML.

Sem este *software* intermediário, o sistema operacional em si não contém recursos próprios para interpretação de programas diretamente, pois há muitas linguagens de programação variadas e a intenção do sistema operacional, quando instalado em um *hardware*, é de conter apenas recursos necessários para disponibilização das funcionalidades dos dispositivos de *hardware* aos usuários e algumas ferramentas básicas adicionais.

Outro tipo de linguagem de programação possui uma etapa intermediária de tradução de código em linguagem de programação para algo mais próximo da compreensão da máquina e excluindo elementos desnecessários para gerar um arquivo temporário necessário para uma segunda conversão.



Esta segunda conversão gera um *software* executável a partir do arquivo temporário gerado chamado de arquivo objeto e não é utilizado posteriormente na execução. Apenas o *software* gerado pela segunda conversão é considerado *software* mesmo e pode então ser executado sem dependência com os dois arquivos da etapa anterior de código fonte e objeto.

Este método é empregado por todas as linguagens que utilizam o chamado compilador para a realização deste processo de tradução, e muitas linguagens se baseiam neste método para gerarem *software* com bom desempenho, que possam ser executados independentemente de outros *softwares* como em linguagens interpretadas.

Um ponto importante é que existem diferentes compiladores para as diferentes plataformas de *software* compatíveis com a linguagem de programação, e assim, para se executar um mesmo programa em várias plataformas é preciso que exista um compilador adequado a cada plataforma em que se deseja executar o *software*.

Caso um sistema operacional não possua um compilador para determinada linguagem de programação, *softwares* desta linguagem não seriam compatíveis com o sistema operacional por não haver meios para se gerar *softwares* executáveis para este sistema. Linguagens como C funcionam desta forma.



Um terceiro meio de se desenvolver programas para algumas linguagens de programação é utilizar um método híbrido dos dois anteriores em que um código é escrito normalmente em um editor e um arquivo contendo o código digitado é gerado.

Depois, um *software* compilador gera um arquivo intermediário objeto que não pode ser executado diretamente, mas sim, interpretado em um outro *software* interpretador que se baseia então no arquivo intermediário e não no código digitado diretamente.

Este *software* interpretador não gera *software* executável, mas apenas executa as instruções do programa, e por este motivo, necessita estar instalado e ser compatível com o sistema operacional desejado.

Assim, neste terceiro tipo de processo de execução de código, é preciso ter um arquivo texto com o código, utilizar um compilador para gerar um arquivo intermediário que pode então servir de base para um interpretador que então realiza a execução do *software*.

Neste caso são necessários dois *softwares* para a execução de outro, e por isto, uma tendência é um tempo maior para a execução de um *software* criado como acontece em linguagens como Java que possui boas qualidades e recursos, mas que possui este ponto negativo em relação ao seu desempenho geral em relação a algumas outras linguagens.

Avaliando as diferentes formas de se executar um *software*, é possível compreender que há fatores que tornam uma linguagem de programação mais atrativa ou indicada de acordo com os tipos de problemas a serem resolvidos, mas existe também a influência de como deve ser executado o *software*, seja de forma interpretada, compilada ou híbrida.

$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

$$x + y = 0$$

$$x = 6 - 2y$$

$$x + a = b$$

$$f(x) = \tan x$$

 $f(x)$

$$x_{1/2}$$

Operadores e Expressões

Fundamentos de Lógica para Programação

Para se desenvolver algoritmos que representam formas de se resolver problemas reais de modo computacional, é preciso que se compreenda que além de imaginar uma sequência de instruções para realizar tarefas, é preciso uma compreensão do problema a ser resolvido e se é ou não possível que se elabore uma solução computacional para o problema em questão de forma completa ou parcial ao menos.

Interpretar problemas e avaliar situações é muito importante, assim como prever situações e elaborar meios para que um algoritmo possa conter as funcionalidades desejadas.

Muitas vezes é preciso realizar cálculos, avaliar situações e escolher diferentes fluxos na execução de algoritmos de forma que este seja capaz de realizar atividades variadas dentro das limitações que possam existir nos algoritmos.

Para a elaboração de algoritmos, existem diversas regras e definições que precisam ser seguidas para que os algoritmos criados sejam todos padronizados e existem muitos símbolos com finalidades específicas como cálculos, por exemplo, assim como palavras que são utilizadas na elaboração e devem ser “reservadas” para uso apenas como comandos, sendo que demais palavras utilizadas representam outros elementos que serão aos poucos estudados.

Operadores Matemáticos

A utilização de operadores matemáticos é semelhante ao seu uso na matemática e representam diferentes cálculos, mas alguns símbolos são diferentes em função da limitação de símbolos existentes no teclado, visando facilitar sua identificação em meio a um algoritmo que pode conter muitos elementos distintos.

Os operadores matemáticos utilizados na elaboração de algoritmos são os mais comuns, pois em geral são utilizados na grande maioria dos algoritmos, e por isto, são aqueles implementados geralmente nas linguagens de programação. Observe a seguir os operadores mais utilizados e sua finalidade.

OPERADOR	DESCRIÇÃO
+	Operador soma.
-	Operador subtração.
*	Operador multiplicação.
/	Operador divisão.
%	Operador resto da divisão.

Tabela 2: Operadores matemáticos | Fonte: O autor.

A partir da observação da simbologia utilizada na tabela 1, é possível perceber que os símbolos matemáticos para soma e subtração permanecem os mesmos, mas o símbolos utilizados para multiplicação e divisão são alterados para os símbolos “*” e “/” em função do uso constante da letra X e do símbolo “:” e da ausência do símbolo “÷”.

O símbolo “%” que é utilizado como operador resto da divisão entre dois valores serve para devolver como resposta o valor restante de uma divisão com resultado inteiro apenas. O uso para estruturação de cálculos em geral segue a mesma e será demonstrado posteriormente no estudo de expressões.

Operadores Relacionais

Outro tipo de operador se chama relacional, pois estabelece relação entre dados que resultam em valores verdadeiro ou falso, sendo muito utilizado como meio para a tomada de decisão para desvios na execução ou repetições de processamento, por exemplo.

Estes operadores envolvem o uso de raciocínio lógico para seu emprego, diferente dos operadores matemáticos que necessitam apenas que se saiba qual o tipo de cálculo realizado por cada um. A tabela 3 traz os operadores relacionais padrão para algoritmos.

OPERADOR	DESCRIÇÃO
>	Maior.
>=	Maior ou igual.
<	Menor.
<=	Menor ou igual.
=	Igual.
<>	Diferente.

Tabela 3: Operadores relacionais | Fonte: o autor.

Os símbolos contidos na tabela 3 são praticamente os mesmos da matemática, com exceção do símbolo "<>" que é utilizado ao invés do símbolo "≠" em função da ausência do mesmo no teclado. Existem linguagens de programação que utilizam simbologia diferente para este operador.

Operadores Lógicos

Existem várias operações lógicas utilizando operadores distintos para comparações que resultam sempre em valores verdadeiro ou falso, assim como os operadores relacionais, mas seu uso é diferente dos operadores relacionais.

Enquanto os relacionais comparam dois dados, os operadores lógicos são utilizados para comparações compostas, em que são definidas regras para o uso destes operadores e a forma como podem ser utilizados em linguagens de programação.

Os operadores básicos lógicos utilizados na elaboração de algoritmos são E, OU e NÃO, mas existem outros que podem ser estudados como NOR, NAND e XOR, por exemplo. Observe a imagem 8 que traz um meio de se compreender o uso destes operadores.

Expressão A	Expressão B	OPERADOR E
V	V	V
V	F	F
F	V	F
F	F	F
Expressão A	Expressão B	OPERADOR OU
V	V	V
V	F	V
F	V	V
F	F	F
Expressão A		OPERADOR E
V		F
F		V

Imagem 8: Tabelas-verdade | Fonte: o autor.

Por meio do conhecimento das tabelas-verdade de cada operador lógico, é possível compreender sua aplicabilidade e interpretar o que ocorre na aplicação destes operadores em algoritmos.

O operador E age como se, para que seu resultado seja verdadeiro, as expressões analisadas devem ser todas verdadeiras, sem nenhum valor falso que invalide este operador.

O operador OU possui maior facilidade em retornar verdadeiro, pois se pelo menos um dos dados ou expressões avaliados sejam verdadeiros, já é suficiente para que um retorno verdadeiro seja obtido.

Por fim, o operador NÃO apenas inverte o valor verdadeiro ou falso contidos em uma estrutura de dados e pode ser útil em diversas situações relativas à inversão de estados em um algoritmo. Mais adiante serão criados alguns exemplos utilizando estes tipos de operadores.

CONECTE-SE



A compreensão dos operadores lógicos e sua variedade é algo importante para o desenvolvimento de algoritmos, pois podem ser utilizados em diversas situações em que são realizadas tomadas de decisão baseadas em expressões lógicas.

Acesse o link: [Disponível aqui](#)

Expressões Matemáticas e Lógicas

O uso de operadores é muito comum em algoritmos. E expressões de tipos variados podem compor uma solução que não precisa ser necessariamente voltada à realização de cálculos, pois além dos operadores matemáticos, existem os operadores relacionais e lógicos, voltados à elaboração de comparações.

Expressões matemáticas são geralmente elaboradas com valores e operadores matemáticos, mas para obedecerem à forma como uma instrução que use expressões matemáticas, a chamada sintaxe deve ser obedecida, mantendo uma padronização na estrutura de algoritmos.

Já os operadores relacionais e lógicos costumam ser utilizados em condições de estruturas de controle do fluxo de execução do algoritmo e complementam comandos intermediados por condições a serem avaliadas durante a execução. Observe os exemplos da imagem 9.

```
SOMA <- 3 + 5;  
MEDIA <- (VALOR1 + VALOR2) / 2;  
RESULTADO <- 5 + (3 * 2 - (1 + 4));  
SE (IDADE >= 18) ENTÃO  
SE (VALOR > 0 E VALOR < 10) ENTÃO
```

Imagem 9: Exemplo de expressão matemática | Fonte: O autor.

Nestes exemplos da imagem 9, existem alguns pontos de atenção, como, por exemplo, a inclusão do símbolo "<-" que representa uma atribuição de dados a uma estrutura de dados para que possa este valor ficar temporariamente armazenado para uso posterior do algoritmo enquanto está sendo executado.

Em relação às instruções em si, a primeira traz uma soma simples entre dois valores numéricos inteiros, em que o resultado será inserido na estrutura chamada de "SOMA". Na sequência, a estrutura "MEDIA" receberá o resultado do cálculo da soma dos dois dados contidos nas estruturas valor1 e valor2, que após a soma, devem ser divididos por 2, pois a operação realizada primeiro é a contida nos parênteses.

Depois, uma expressão mais complexa é posta como exemplo, em que os cálculos entre parênteses são realizados antes dos demais, mas como há dois níveis de parênteses, o mais interno é calculado antes, para depois os parênteses mais externos e, por fim, os demais cálculos restantes.

Nos dois últimos exemplos, são utilizadas instruções de controle para exemplificar a inclusão de condições utilizando operadores relacionais e lógicos, em que em um, a condição retorna verdadeira apenas se o dado contido na variável “IDADE” for maior ou igual a 18. E no outro exemplo, são usadas duas condições relacionais, mas que pelo uso do operador “e”, devem ser consideradas juntas, pois apenas as duas sendo verdadeiras é possível a obtenção de um resultado verdadeiro.

ABORDAGEM PRÁTICA



Desde os estudos da matemática até a elaboração de instruções de controle de fluxo da execução de algoritmos, expressões matemáticas, relacionais e lógicas são utilizadas frequentemente na implementação de algoritmos, pois permitem que os algoritmos tenham a capacidade de tomada de decisões a partir de resultados obtidos nestas expressões.

A automatização proporcionada por algoritmos em equipamentos e máquinas de todos os tipos avança há décadas, e muito dessa possibilidade de evolução tecnológica se baseia na programação da máquina a partir do *software* que possui em si vários mecanismos capazes de avaliar dados e realizar ações a partir de resultados obtidos em expressões.

Por isso, é essencial ao desenvolvedor ser capaz de elaborar expressões diversas utilizando operadores e sendo capaz de avaliar as possibilidades resultantes destas expressões.

Precedência de Operadores

Com tantos operadores disponíveis é comum que vários sejam utilizados em uma mesma expressão, e assim, é preciso que seja determinada uma ordem para que estes operadores sejam executados de forma padronizada, seguindo os princípios da matemática principalmente.

Pela regra geral, a multiplicação, divisão e resto são prioritárias sobre soma e subtração, mas o uso de parênteses pode mudar avaliação natural e operações mais fracas podem ser realizadas antes de operações de maior prioridade.

Os operadores relacionais possuem todos os mesmos níveis de prioridade e só são executados após os operadores matemáticos, pois como trabalham com análises de veracidade de expressões, é importante terminar os cálculos e após, verificar se o resultado é verdadeiro ou falso.

No exemplo da imagem 13, por exemplo, a expressão que serve de base para uma estrutura `"RESULTADO <- 5 + (3 * 2 - (1 + 4));"` deve ser calculada primeiramente $(1+4)$ por serem os parênteses mais internos, para depois serem realizadas as operações dos parênteses mais externos, e neste, há uma multiplicação e uma subtração. Neste caso, primeiro multiplica-se $3 * 2$, para depois ocorrer a subtração do valor obtido na multiplicação pelo valor resultante dos parênteses internos. Com isto, o valor gerado para alocação em memória seria 6, nesta expressão exemplo.



**PARA
GABARITAR**

A precedência de operadores é fundamental para a lógica de algoritmos ser correta, mas para isto é importante conhecer a finalidade de cada tipo e operador específico e a partir da construção de expressões diversas mudando operadores e adicionando ou removendo parênteses, testar resultados a partir do ajuste da precedência.

Tipos de Dados

Noções de Dados e Tipos Básicos

A base da tecnologia da informação são os dados. Sejam eles dos tipos mais simples como números ou texto, ou mais complexos como estruturas heterogêneas de dados, imagens, sons, etc., todos são formados por bits e seu processo de armazenamento é semelhante em qualquer dos tipos.

Em algoritmos os dados devem ser definidos logo no momento da declaração de uma estrutura de dados, e geralmente, logo no início do algoritmo, antes do início da escrita das instruções que servirão para manipular estes dados.

Os tipos básicos de dados para manipulação são os utilizados para armazenamento temporário e manipulação de valores numéricos inteiros ou decimais, caracteres simples que englobam símbolos gerais do teclado como letras, números, e demais símbolos, e por último, um quarto tipo se refere a valores verdadeiro e falso apenas chamado lógico.

Estes tipos podem ser usados livremente na implementação e algoritmos, mas precisam estar associados a estruturas de dados de forma a definirem quais tipos de dados são aceitos por cada estrutura de dado declarada.

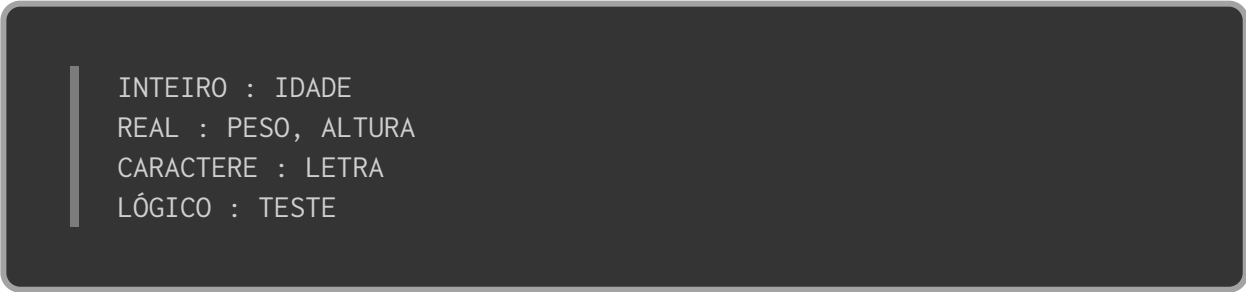
Estruturas Básicas de Dados (variáveis)

As estruturas de dados podem ser simples espaços em memória para armazenamento de um único dado declarado com algum dos tipos básicos citados e servem como armazenamento temporário em memória para uso durante a execução de um algoritmo.

Esse tipo de estrutura de dado é chamado de variável em função da volatilidade alta dos dados inseridos nestas, em função da possibilidade de troca dos dados inseridos e da perda dos mesmos ao final da execução do algoritmo.

As variáveis devem ter seus tipos definidos em suas declarações por padrão, mas não há impedimentos de alguém optar por não definir tipos e apenas no momento em que uma variável declarada sem tipo definido receber um dado, este definirá qual o tipo associado à variável, mas isso reduz a previsibilidade do algoritmo e aumenta as chances da ocorrência de erros.

Em geral, a sintaxe para declaração de variáveis, segundo Forbellone (2005), define que a declaração se inicia pelo tipo de dado a ser associado à variável, seguido do símbolo ":" que age como separador, e em seguida, o nome (identificador) que será atribuído à variável, finalizando com um símbolo ";" a instrução. Observe os exemplos da imagem 10.



```
INTEIRO : IDADE  
REAL : PESO, ALTURA  
CARACTERE : LETRA  
LÓGICO : TESTE
```

Imagem 10: Exemplos de declaração de variáveis. | Fonte: O autor.

Nos exemplos da imagem 15, são declaradas variáveis de tipos distintos para que sejam conhecidos todos os tipos básicos, sendo "INTEIRO" para números sem casas decimais, e "REAL" para números que aceitam casas decimais. O tipo "CARACTERE" é utilizado para caracteres diversos, mas também pode ser utilizado para texto em geral contendo uma série de caracteres de qualquer quantidade. Por fim, o tipo "LÓGICO" é usado para variáveis que possam aceitar apenas os estados lógicos verdadeiro ou falso.

Alguns detalhes importantes para que se compreenda melhor a sintaxe de algoritmos baseados no padrão adotado por Forbellone (2005), ao final da maior parte das instruções, é utilizado o símbolo ";" para indicar que a instrução encerra no ponto em que houver este símbolo, e assim, mesmo que uma instrução seja dividida em mais de uma linha, as quebras de linhas não simbolizam final de instrução, mas não será utilizado este padrão neste material para maior facilidade de compreensão.

Outro símbolo importante utilizado no exemplo da imagem 15 é "," que serve como separador para o caso de duas ou mais variáveis serem declaradas com um mesmo tipo, economizando assim, digitação e linhas de código.

CONECTE-SE

As variáveis funcionam como depósitos de dados que podem ser manipulados durante a execução e algoritmos e devem ser bem compreendidas para que seu uso constante não represente um problema. As constantes complementam as variáveis como recurso para armazenamento de dados não variáveis durante a execução de algoritmos.

Acesse o link: [Disponível aqui](#)

Atribuição de Dados

Após a declaração de variáveis ou outras estruturas de dados, estas ficam disponíveis para o algoritmo que pode inserir e alterar dados nestas livremente, lembrando que estas são estruturas temporárias, e ao final da execução do algoritmo, os dados são perdidos com a liberação da memória utilizada pelo algoritmo.

Para que dados sejam adicionados a variáveis e outras estruturas, um dos meios mais utilizados é a chamada atribuição de dados que utiliza o símbolo "<-" para que o dado ou resultado de uma expressão que esteja à direita do símbolo seja adicionado à estrutura de dado. Observe os exemplos da imagem 11.

```
IDADE <- 20;  
PESO <- 50.5;  
NOME <- "RONIE";  
MEDIA <- (VALOR1 + VALOR2) / 2;
```

Imagem 11: Exemplos de atribuição de dados a variáveis. | Fonte: O autor.

Pelos exemplos da imagem 11, é possível compreender a forma como se realizam atribuições simples de valores a variáveis como nos três primeiros exemplos que atribuem um valor inteiro, um decimal e uma *string* (como também pode ser chamada uma sequência de caracteres quaisquer), respectivamente.

O último exemplo da imagem 16 traz uma expressão que tem seu resultado atribuído à variável “MEDIA”, lembrando que o cálculo de expressões é realizado antes de a atribuição ser realizada.

PARA GABARITAR



O uso de estruturas de dados como variáveis é fundamental para o desenvolvimento de algoritmos e saber manipular seus dados de forma adequada é importante. Uma prática que auxilia a desenvolver algoritmos seguros é a do uso da atribuição para realizar inicializações de variáveis antes de seu primeiro uso, pois assim, valores indesejados contidos na área de memória alocada para variáveis não correm o risco de serem utilizados.

Exemplos de Algoritmos com Variáveis

Tendo então conhecimento dos elementos essenciais de um algoritmo em relação à aquisição e tratamento de dados, é interessante conhecer a estrutura básica de um algoritmo e assim, já estar habilitado a formular pequenos algoritmos, mesmo que com funcionalidades mínimas ainda.

Os autores costumam divergir sobre a forma como constroem algoritmos e em função disto, é mais complexa a citação de um ou outro autor especificamente, mas para este material, foram unidos elementos utilizados por três autores distintos

sendo estes Forbellone (2005), Manzano (1997) e Ascencio (2012) para que se possam aproveitar aspectos positivos de cada um deles e compor um estudo bem completo da elaboração de algoritmos.

Para este material, a estrutura básica para algoritmos seguirá alguns padrões como, por exemplo, o uso de letras maiúsculas, apenas para maior destaque visual no texto, sintaxe seguindo o padrão utilizado por Ascencio (2012) que utiliza pequena quantidade de símbolos na construção das instruções que possuem o mesmo significado das sintaxes de outros autores que escrevem seus algoritmos de forma um pouco diferente.

Observe o exemplo da imagem 12 para conhecer a estrutura básica de elaboração de algoritmos neste material, observando as palavras reservadas utilizadas em instruções e os elementos que complementam estas palavras reservadas em muitos casos, chamados de parâmetros para as palavras reservadas.

```
INÍCIO  
DECLARE  
    INTEIRO A, B, C;  
    A <- 10;  
    B <- A * 2;  
    C <- A * B;  
FIM.
```

Imagem 12: Exemplos de atribuição de dados a variáveis. | Fonte: O autor.

Neste exemplo de algoritmo completo da imagem 12, observa-se a estrutura básica de escrita de algoritmos usada nesse material onde a palavra reservada INÍCIO, como foi utilizado em exemplos anteriores, é a primeira instrução.

Logo em seguida, a palavra reservada DECLARE indica a declaração de variáveis e outras estruturas de dados a partir deste ponto, e neste exemplo, são declaradas três variáveis A, B e C do tipo inteiro para uso no processamento do algoritmo.

Logo em seguida da declaração, ocorre uma atribuição do valor inteiro 10 para a variável A, onde se utiliza o símbolo padrão "<-" para indicar a atribuição, e o símbolo ";" para encerrar a instrução como já comentado anteriormente.



Em seguida, em uma segunda instrução de atribuição, o valor de A, 10, é multiplicado por 2 resultando em um valor 20 a ser atribuído à variável B, e por fim, em uma última instrução de atribuição, os valores 10 da variável A e 20 da variável B são multiplicados para que o valor resultante 200 seja armazenado em C.

A elaboração de algoritmos parece simples inicialmente, mas tende a tornar-se mais complexa à medida que são incluídas novas estruturas e palavras reservadas com funcionalidades que necessitam de maior esforço lógico para a elaboração de soluções complexas.



Entrada e Saída de dados

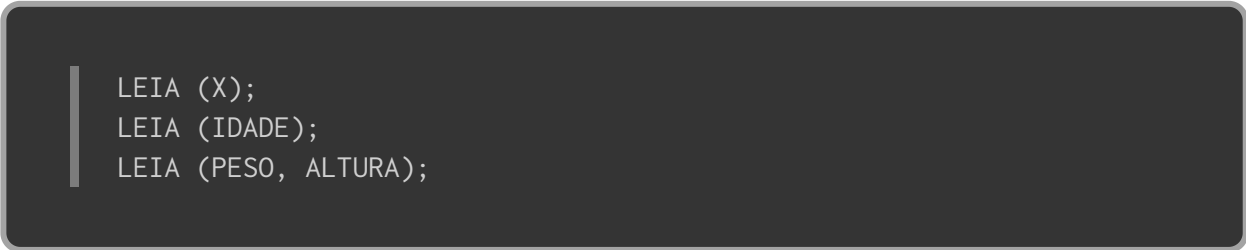
Entrada de Dados

Após os estudos sobre dados simples, é preciso compreender como é possível obter dados para manipulação em um algoritmo. Este é um dos fundamentos da elaboração de algoritmos, pois é o recurso que permite a interatividade fundamental entre usuário e algoritmo.

A princípio, o processo de entrada de dados mais comum é a digitação de dados pelo usuário que serve como periférico padrão para interação com algoritmos, mas há outras formas como leitura de dados gravados em arquivos, dados obtidos em redes de computadores, ou até na rede mundial Internet.

Para os estudos deste material, será inicialmente tratada apenas a entrada de dados via teclado, pois é uma forma comum de realização deste processo e a mais simples de entendimento em uma etapa de iniciação nos estudos em relação ao desenvolvimento de *software*.

A palavra reservada “LEIA” é utilizada como comando para a elaboração de instruções de entrada de dados e sua sintaxe para construção de instruções é bastante simples. Observe o exemplo da imagem 13.



```
LEIA (X);  
LEIA (IDADE);  
LEIA (PESO, ALTURA);
```

Imagem 13: Exemplos de entrada de dados. | Fonte: O autor.

Nestes exemplos da imagem 13, a mesma palavra reservada padrão “LEIA” é utilizada e isto mostra que é simples o seu uso, por ser um comando único para a função deste estudo.

No primeiro exemplo, o comando solicita a digitação de um dado para uma variável “X” que espera a inserção de um dado quando executado o algoritmo, da mesma forma que o segundo exemplo, mas este, utiliza como estrutura de dados para

armazenamento temporário, uma variável “IDADE” que pelo nome, possui um significado mais claro de sua finalidade em relação ao nome dado à variável do primeiro exemplo.

Este detalhe de escolha de nomes é importante, pois à medida que se desenvolvem códigos cada vez maiores, uma maior quantidade de variáveis e outras estruturas de dados podem ser declarados, e nomes muito genéricos utilizados como identificadores para estas estruturas de dados prejudica a interpretação de algoritmos.

Como dica é bom sempre pensar em nomes intuitivos e bem claros em seu significado, mesmo que formados por palavras compostas, lembrando que não é permitido o uso de espaços em nomes e por isto, pode-se criar nomes com as partes compostas todas unidas ou usando o símbolo “_” permitido na elaboração de nomes em algoritmos e em linguagens de programação normalmente.



Nomes como “PESOMAXIMO” ou “PESO_MINIMO” são aceitos sem problemas e possuem um significado bastante intuitivo para interpretação do tipo de dado que poderá estar contido nestas estruturas de dados.

No terceiro exemplo da imagem 18, é importante observar que é permitido que uma instrução de entrada de dados possa incluir mais de uma variável por vez, separando seus nomes com o símbolo “,” quando colocadas dentro dos parênteses do comando

“LEIA”.

Assim, neste terceiro exemplo, foram solicitadas as digitações de dados para as variáveis “PESO” e “ALTURA” em uma mesma instrução, e o usuário poderá digitar os dois dados em sequência, sendo necessário que a cada dado inserido, seja pressionada a tecla “ENTER” para finalizar a inserção e poder então passar à digitação de outro dado para outra variável solicitada numa instrução, ou para que a execução normal do algoritmo prossiga.

PARA GABARITAR



A entrada de dados é fundamental para um algoritmo, pois é a interação com usuário que ocorre por este mecanismo. Um recurso bastante útil e que deve se tornar uma prática comum é o uso de validações para redução de entradas de dados incorretas.

Estruturas de dados são tipadas para que os tipos de dados adequados sejam inseridos, mas há também casos em que existem exceções que necessitam ser tratadas, como no caso da realização de uma operação de divisão, que o segundo valor a ser utilizado como divisor não pode ser zero, pois isto gera um erro em função de não ser possível esta operação.

Pode-se então, incluir um mecanismo no algoritmo para verificar um valor recebido por uma variável, e caso seja zero, um novo valor diferente deve ser solicitado ou atribuído a uma variável.

Saída de Dados

O processo que complementa a entrada de dados e o processamento destes é a saída de dados que também pode ser realizada de diferentes maneiras, como a exibição de dados em tela que é a opção padrão para esta atividade, a gravação em disco que também é bastante comum, o envio por meio de redes de computadores, ou como na entrada de dados, até como envio de dados pela *web*.

Para esta aula a saída de dados padrão em tela será a única explorada, pois também sendo a opção padrão, é a mais adequada para este ponto dos estudos, mas a implementação de instruções com comandos de saída é um pouco mais complexa que na entrada de dados.

Neste tipo de instrução de saída, utiliza-se a palavra reservada “ESCREVA” que também necessita de parâmetros inseridos entre parênteses para a estruturação de instruções de saída, mas é possível mesclar diferentes tipos de parâmetros neste tipo de comando. Observe os exemplos da imagem 14.

```
ESCREVA (“Texto Exemplo”);  
ESCREVA (IDADE);  
ESCREVA (PESO, “ - “, ALTURA);  
ESCREVA (“Nome: “, NOME);  
ESCREVA (“O valor da média entre “, VALOR1, “ e “, VALOR2, “ é: “,  
MEDIA);
```

Imagem 14: Exemplos de saída de dados. | Fonte: O autor.

Nos exemplos trazidos na figura 14, o primeiro exemplo traz apenas uma mensagem a ser exibida ao usuário, indicada entre aspas que fazem parte da sintaxe do comando “ESCREVA” e devem sempre ser utilizadas em caso de exibição de mensagens.

Um detalhe importante é que as mensagens entre aspas não representam palavras reservadas da linguagem ou nomes de estruturas de dados, e por isso, o texto a ser inserido pode conter qualquer conjunto de caracteres formando ou não palavras, em letras maiúsculas ou minúsculas, símbolos diversos do teclado, e números, por exemplo.

No segundo exemplo, apenas o nome de uma variável “IDADE” é utilizado, e esta instrução apenas exibirá o dado contido na variável, que pelo seu nome, imagina-se que seja um número inteiro, pelas características deste tipo de informação.

Assim, percebe-se que não é obrigatório o uso de aspas em todas as instruções de saída de dados, sendo então livre a composição de instruções de saída utilizando texto ou variáveis em quantidades variadas.

O terceiro exemplo já mescla estes elementos como parâmetros do comando de saída, e é importante observar que cada elemento a ser exibido, por serem distintos, devem ser separados pelo símbolo “,” que no momento da exibição concatena todos os parâmetros contidos no comando em sequência para serem exibidos ao usuário, e por isto que neste exemplo o texto utilizado “-” contém espaço em branco antes e depois do traço, para evitar que os dados das variáveis apareçam emendados ao símbolo de traço.

O quarto exemplo é bem típico em algoritmos e mostra ao usuário o dado contido na variável nome, mas antes informa por meio do texto entre aspas que tipo de informação está sendo passada, deixando assim, a aparência da mensagem semelhante à de um formulário de dados comum.

O quinto exemplo é o mais complexo e contém vários parâmetros que unidos, exibem uma informação completa ao usuário. O conteúdo “(“O valor da média entre “, VALOR1, “ e “, VALOR2, “ é: “, MEDIA)” parece bastante complexo, mas na verdade facilita a compreensão se forem separados os elementos a partir das vírgulas, e assim, temos os parâmetros indicados na imagem 15.

“O valor da média entre“	- Mensagem simples a ser exibida
VALOR1	- Valor da variável VALOR1 a ser exibido
“ e “	- Mensagem simples a ser exibida
VALOR2	- Valor da variável VALOR2 a ser exibido
“ é: “	- Mensagem simples a ser exibida
MEDIA	- Valor da variável MEDIA a ser exibido

Imagem 15: Separação dos parâmetros do exemplo da imagem 19. | Fonte: O autor.

A concatenação ou união entre estes elementos exibe na tela algo como “O valor da média entre 10 e 2 é: 5”, por exemplo, imaginando que foram inseridos os valores 10 e 2 para as variáveis “VALOR1” e “VALOR2”, respectivamente, e o valor para a variável “MEDIA” foi obtido a partir do cálculo da média entre os dois valores.

CONECTE-SE



Após conhecer a ideia de entrada e saída de dados, é interessante complementar os estudos conhecendo a parte de entrada e saída de dados de *hardware*, pois os algoritmos podem utilizar estes recursos para realizar estes dois processos.

Acesse o link: [Disponível aqui](#)

Analizando um Primeiro Algoritmo Completo

Após conhecer parte dos conceitos fundamentais do desenvolvimento de algoritmos, é possível avaliar um exemplo completo de algoritmo para compreender a aplicação do que foi estudado até então e compreender melhor o uso de cada conceito visto até então.

Partindo de um problema tema para o algoritmo, pode-se simular o desenvolvimento de uma possível solução computacional lembrando que não existe apenas uma solução possível para cada problema, mas sim, variações possíveis que contemplam funcionalidades semelhantes ou iguais, podendo ser implementadas de diferentes formas e com variações na lógica inclusive.

Observe a imagem 16 para um exemplo completo de algoritmo que se propõe a receber três valores, e retornar a média entre estes valores, lembrando que para este cálculo é preciso somar os três valores e dividir esta soma por três para obter o resultado esperado.

```
INÍCIO
DECLARE
    REAL VALOR1, VALOR2, VALOR3, MEDIA;
    ESCREVA ("DIGITE UM PRIMEIRO VALOR: ");
    LEIA (VALOR1);
    ESCREVA ("DIGITE UM SEGUNDO VALOR: ");
    LEIA (VALOR2);
    ESCREVA ("DIGITE UM TERCEIRO VALOR: ");
    LEIA (VALOR3);
    MEDIA <- (VALOR1 + VALOR2 + VALOR3) / 3;
    ESCREVA ("A MÉDIA ENTRE OS TRÊS VALORES É ", MEDIA);
FIM.
```

Imagem 16: Separação dos parâmetros do exemplo da imagem 20. | Fonte: O autor.

Neste algoritmo da imagem 16, é possível observar todos os detalhes de estrutura de um algoritmo comum e os principais conceitos estudados até então, desde a primeira aula.

Como de costume na elaboração de algoritmos, inicia-se um algoritmo com a palavra reservada "INÍCIO" e na sequência, a palavra reservada "DECLARE" inicia a etapa de declaração de variáveis para uso no algoritmo.

São declaradas quatro variáveis "VALOR1", "VALOR2" e "VALOR3" para receber os dados a serem informados pelos usuários para o cálculo como solicitado como requisito para o algoritmo. Por fim, é declarada uma quarta variável "MEDIA" para receber, posteriormente, o resultado do cálculo a ser realizado com os dados contidos nas três outras variáveis declaradas no algoritmo.

Em seguida, uma sequência de comandos "ESCREVA" e "LEIA" realizam a função de informar ao usuário que ele deve digitar um primeiro valor, para em seguida solicitar a digitação do valor em si através de um "prompt" para inserção de dados pelo teclado.

Este processo é repetido por três vezes a fim de que os dados para as três variáveis bases para o cálculo da média sejam preenchidos com valores que podem ser inteiros ou decimais em função do tipo “REAL” definido na declaração das variáveis.



Depois da obtenção dos dados, uma atribuição é realizada para a variável “MEDIA” que recebe o resultado da soma e posterior divisão dos valores contidos nas três variáveis para obtenção da média solicitada.

Por fim, após inserção do resultado do cálculo na variável “MEDIA”, o valor da mesma é utilizado no comando de saída de dados “ESCREVA” para exibir uma mensagem informando o resultado do cálculo da média com o valor obtido.

ABORDAGEM PRÁTICA



Aplicações em geral possuem entradas e saídas de dados para interação com usuários. Algumas aplicações podem ser totalmente automatizadas e não dependerem de nenhum tipo de interação humana, mas em geral, a possibilidade de interação existe e é importante em uma grande quantidade de algoritmos.

Existem sistemas complexos que necessitam de entradas de dados como os chamados ERPs (*Enterprise Resource Planning* ou Sistema de Gestão Empresarial) que recebem e manipulam grandes quantidades de dados em negócios diversos, jogos, *softwares* aplicativos em geral, etc.

Implementar algoritmos com mensagens de auxílio ao correto preenchimento de entradas de dados, por exemplo, é um bom exemplo de uso de saídas e entradas de dados em conjunto, muito comuns em aplicações de todo tipo.



Estruturas de Decisão

Estruturas de Controle

Após os estudos de tipos, variáveis e os recursos de interatividade para entrada e saída de dados, já é possível avançar nos estudos e conhecer novos recursos de programação capazes de influenciar o chamado fluxo de execução da aplicação.

Esse fluxo representa a ordem de execução das instruções e por meio dos recursos estudados até então, os algoritmos poderiam ser executados apenas de forma sequencial, sem opções de escolhas na forma como as instruções são executadas, ou até se devem ser executadas em determinadas situações controladas pelo próprio algoritmo.

As chamadas estruturas de controle representam meios de definir critérios para que determinadas instruções ou blocos de instruções sejam executadas ou não dependendo de condições pré-determinadas no código, e que afetam a execução dos algoritmos.

Estrutura de Decisão Condicional Simples

Uma estrutura de decisão condicional simples representa um recurso de controle de fluxo de execução que se baseia em uma condição controlada pelo algoritmo em tempo de execução que define se uma ou mais instruções devem ser executadas ou não.

A ideia do uso de uma condição para a execução de instruções remete ao conteúdo visto na aula 4 em que alguns tipos de operadores não realizavam cálculos, mas serviam para avaliar expressões com o objetivo de retornar valores verdadeiros ou falsos, perfeitos para o tipo de condição utilizável neste tipo de estrutura de controle.

Estas expressões condicionais devem ser muito bem elaboradas, pois desvios incorretos causados por erros lógicos associados a condições equivocadas em sua elaboração comprometem funcionalidades de algoritmos e podem gerar problemas como instruções que nunca são executadas ou executadas em momentos inoportunos.

A palavra reservada mais conhecida para este tipo de instrução é “SE” e possui uma sintaxe bem simples para sua estruturação, sendo que os maiores problemas em seu uso são ocasionados pelas condições elaboradas para as instruções. Observe o exemplo da imagem 17 para compreender como se pode estruturar um comando deste tipo.

```
SE (VALOR > = 0) ENTÃO  
    ESCREVA(“NÚMERO POSITIVO”);  
FIMSE;
```

Imagem 17: Exemplo de estrutura condicional simples. | Fonte: O autor.

Neste exemplo da imagem 17, um comando condicional simples é implementado com o objetivo de avaliar o valor contido na variável “VALOR”, e caso contenha um valor numérico maior ou igual a zero, é exibida uma mensagem informando que o número digitado é positivo.

É importante neste exemplo observar a sintaxe do comando, em que após o uso da palavra reservada “SE”, a condição é inserida na instrução, que neste caso, verifica se o valor contido na variável “VALOR” é maior ou igual a zero para que a mensagem adequada seja exibida.

A sintaxe básica de um comando utilizando a palavra reservada “SE” inicia pela palavra, seguida por uma expressão condicional normalmente entre parênteses, e por fim, outra palavra reservada “ENTÃO” para, na sequência, ser colocada uma instrução ou bloco de instruções a serem executadas apenas quando a expressão condicional resultar verdadeiro.

Assim, no exemplo específico da imagem 17, a mensagem “NÚMERO POSITIVO” seria exibida quando o dado contido na variável “VALOR” desse exemplo for qualquer número maior ou igual a zero.

Na linha seguinte, é inserida uma instrução para exibir ao usuário uma mensagem, lembrando que esta será exibida apenas em caso de a expressão resultar verdadeiro, e o dado contido na variável “VALOR” ser realmente positivo. Para encerrar a instrução de controle de decisão simples, utiliza-se a palavra reservada “FIMSE”.

Estrutura de Decisão Composta

Partindo do que foi estudado na estrutura de decisão simples, foi visto a importância de se poderem criar opções na execução de algoritmos além da sequencial execução de instruções sem desvios na ordem em que são executadas.

A estrutura de decisão composta é um tipo mais completo de estrutura de decisão, pois permite que se tenha duas opções de instruções a serem executadas a partir do resultado de expressões condicionais, e para isto uma nova palavra reservada “SENÃO” é adicionada para implementação deste tipo de instrução.

Observe o exemplo da imagem 18 para conhecer a sintaxe deste tipo de instrução e compreender o que pode ser agregado a uma estrutura de controle para decisão a partir desta variação da construção deste tipo de instrução.

```
SE (VALOR >= 0) ENTÃO  
    ESCREVA (“NÚMERO POSITIVO”);  
SENÃO  
    ESCREVA (“NÚMERO NEGATIVO”);  
FIMSE;
```

Imagem 18: Exemplo de estrutura condicional simples. | O autor.

Observando o exemplo da imagem 18, é possível perceber que a mesma instrução de controle de fluxo possui agora duas instruções de exibição de mensagem ao usuário.

A primeira será exibida apenas se o dado contido na variável “VALOR” for positivo, ao passo que em caso de a expressão condicional resultar falso e o dado contido for negativo, a outra mensagem é automaticamente executada.

Com isso é possível que um algoritmo possa controlar a execução de instruções de forma adequada para dados que vão sendo avaliados em tempo de execução, por exemplo, permitindo que sejam criados algoritmos mais dinâmicos.

**PARA
GABARITAR**

Estruturas de decisão trazem um adicional importante para o desenvolvimento de algoritmos e devem ser cuidadosamente utilizados, pois envolvem desvios na execução de algoritmos e equívocos na estruturação destas instruções pode acarretar processamento incorreto de dados ou até de perda de funcionalidade.

Sempre é importante avaliar as expressões condicionais utilizadas, testar seus possíveis resultados e validar se a condição funcionará como desejado, reduzindo as chances de ocorrências de erros em tempo de execução.

Estrutura de Decisão Aninhada

O uso de estruturas de decisão permite que se possa controlar o fluxo de execução de um algoritmo, mas para todas as situações ter duas alternativas é suficiente?

Há casos em que pode ser necessário que se tenha mais de duas alternativas de escolha de desvio de fluxo, como, por exemplo, na escolha entre decisões a serem executadas a partir da escolha entre cores.

Uma das situações mais comuns que podem gerar a necessidade de maior possibilidade de escolhas é na implementação de menus de opções em que cada escolha pode desviar o fluxo de execução de um algoritmo para diferentes conjuntos de instruções. Observe o exemplo da imagem 19 para melhor compreensão deste tipo de situação.

```
SE (OPCAO = 1) ENTÃO
    ESCRIVA ("A OPÇÃO ESCOLHIDA FOI A PRIMEIRA");
SENÃO SE (OPCAO = 2) ENTÃO
    ESCRIVA ("A OPÇÃO ESCOLHIDA FOI A SEGUNDA");
    SENÃO SE (OPCAO = 3) ENTÃO
        ESCRIVA ("A OPÇÃO ESCOLHIDA FOI A TERCEIRA");
        SENÃO SE (OPCAO = 4) ENTÃO
            ESCRIVA ("A OPÇÃO ESCOLHIDA FOI A QUARTA");
            SENÃO
                ESCRIVA ("OPÇÃO DESCONHECIDA");
        FIMSE;
    FIMSE;
FIMSE;
```

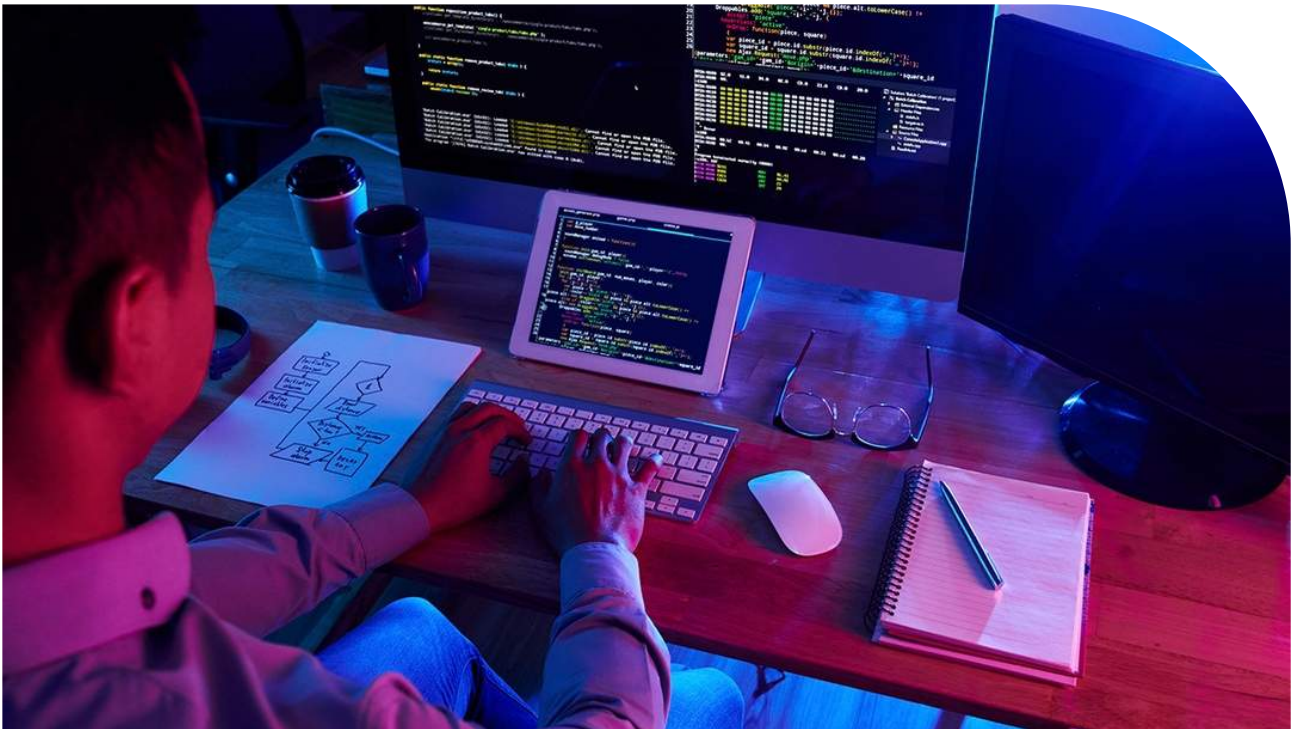
Imagem 19: Exemplo de estrutura condicional composta. | Fonte: O autor.

Tendo como base o exemplo da imagem 19, observa-se que existem múltiplas opções sendo tratadas em uma composição de instruções baseadas no comando “SE” que são organizadas de forma a fazerem parte de uma estrutura única em que no caso de uma das condições ser verdadeira, não há o risco de as instruções contidas nas demais serem executadas.

A estrutura deve ser organizada de forma que cada nova estrutura de decisão seja implementada como a alternativa falsa da estrutura anterior, fazendo assim com que cada nova estrutura esteja agregada à estrutura anterior, e assim, para melhor compreensão do algoritmo, aconselha-se a endentação (espaçamento) mais à direita para cada nova estrutura que seja aninhada.

No caso específico desse exemplo da imagem 19, a ordem de execução das estruturas aninhadas se baseia na ideia de que primeiro é avaliado se o valor contido na variável “OPCAO” é igual a 1. Caso seja, executará a instrução referente e não executará o conteúdo de “SENÃO” que contém todo o restante das estruturas aninhadas, e assim, encerra-se toda esta estrutura aninhada.

Caso a primeira condição resulte falsa, a estrutura parte para a avaliação do “SENÃO” que traz toda a estrutura aninhada que realiza nova avaliação do valor contido na variável “OPCAO”, e caso seu valor seja 2, resulta verdadeiro e a instrução contida na estrutura é executada.



Pode ocorrer desta segunda condição resultar falsa também, e assim, o algoritmo passa a avaliar a próxima estrutura contida no “SENÃO” desta segunda estrutura que contém outro aninhamento de estruturas condicionais.

Nesta terceira avaliação, caso a condição resulte verdadeira pela variável conter o valor 3, a instrução que pertence a esta estrutura é executada, mas caso resulte falsa, novamente segue-se para o conteúdo contido agora em “SENÃO” desta estrutura que agora avalia se o valor contido na variável “OPÇÃO” é igual a 4.

Caso seja verdadeira a expressão, executa a instrução contida, mas caso seja falsa, executa agora o comando “SENÃO” sem mais nenhuma estrutura aninhada e nem condição a ser avaliada, e assim, qualquer valor diferente dos demais avaliados anteriormente entre 1 a 4 resultará na execução da instrução contida nesta última opção.

Outra forma de se aninhar estruturas de decisão é por meio da complementação entre condições, em que uma deve ser avaliada apenas se uma outra anterior tiver um resultado esperado para esta segunda estrutura. Observe o exemplo da imagem 20.

```
SE (DOCUMENTO = VERDADEIRO) ENTÃO  
    SE (IDADE >= 18) ENTÃO  
        ESCREVA ("ENTRADA PERMITIDA")  
    SENÃO  
        ESCREVA ("MENOR DE IDADE");  
SENÃO  
    ESCREVA ("SEM DOCUMENTO");  
FIMSE;
```

Imagem 20: Exemplo de estrutura condicional aninhada. | Fonte: O autor.

Neste exemplo da imagem 20 ocorre uma situação também bastante comum no desenvolvimento de algoritmos em que a avaliação de uma pessoa estar ou não devidamente documentada para que possa então ser verificado o documento em relação à idade da pessoa.

Para isto, são avaliados os valores contidos primeiramente na variável "DOCUMENTO", e caso seja "VERDADEIRO", a segunda estrutura condicional é iniciada e é então verificado se o valor contido na variável "IDADE" é maior ou igual a 18. Se tanto a primeira, quanto a segunda condição forem verdadeiras, é exibida a mensagem de 'entrada permitida', mas se a segunda condição resultar falsa, é exibida a mensagem de a pessoa ser menor de idade. Por fim, se a primeira condição resultar falsa, a segunda nem é avaliada e já é exibida uma mensagem informando a falta de documentação.

ABORDAGEM PRÁTICA



Complementando o que já foi citado sobre operadores e expressões, as estruturas de controle de fluxo baseados em estruturas de decisão simples, compostas ou de seleção múltipla, todas estas estruturas necessitam de expressões para que se possam definir quando e se trechos de algoritmos devem ser executados.

As estruturas de decisão são essenciais na elaboração de algoritmos que realizam verificações relacionadas a dados como datas, idades, verificações de palavras, validações de dados, etc.

É importante compreender como se elaboram instruções condicionais para que se possam automatizar cada vez mais os processos, e assim, criar algoritmos cada vez mais complexos, completos e eficientes.

Estrutura de Seleção Múltipla

Por último, existe um tipo mais específico de estrutura de decisão para quando uma variável pode definir ao menos dois ou mais valores distintos de uma variável que podem ser utilizados como critérios para a escolha entre instruções a serem ou não executadas. Um exemplo de uso deste tipo de estrutura é trazido na imagem 21:


```
ESCOLHA OPCAO
```

```
CASO 1: ESCRIVA ("A OPÇÃO ESCOLHIDA FOI A PRIMEIRA");
```

```
CASO 2: ESCRIVA ("A OPÇÃO ESCOLHIDA FOI A SEGUNDA");
```

```
CASO 3: ESCRIVA ("A OPÇÃO ESCOLHIDA FOI A TERCEIRA");
```

```
CASO 4: ESCRIVA ("A OPÇÃO ESCOLHIDA FOI A QUARTA");
```

```
CASO CONTRÁRIO: ESCRIVA ("OPÇÃO DESCONHECIDA");
```

```
FIMESCOLHA;
```

Imagem 21: Exemplo de estrutura de seleção múltipla. | Fonte: O autor.

Por meio do exemplo trazido na imagem 21, a alternativa de implementação do trecho de algoritmo mostrado na imagem 19 parece muito mais simples em sua lógica, interpretação e escrita, mas é preciso observar que este tipo de estrutura serve apenas para casos em que um único valor deve ser avaliado em relação a uma variável, e não é possível a composição de expressões lógicas ou relacionais como utilizadas no comando "SE".

CONECTE-SE



A escolha entre utilizar um tipo de estrutura de decisão SE...SENÃO ou ESCOLHA...CASO é simples, pois a segunda opção é bastante restrita a algumas situações, mas para um estudo complementar, segue link para material de apoio.

Acesse o link: [Disponível aqui](#)



Estruturas de Repetição

Estruturas de Controle

Também existem estruturas que permitem que uma instrução ou conjunto de instruções sejam executados repetidas vezes de uma forma controlada para que determinadas situações encerrem as repetições, se desejado.

Este tipo de estrutura é bastante útil e permitiu que uma significativa redução na quantidade de linhas de algoritmos fosse obtida em função de ações repetitivas não mais necessitarem ser escritas diversas vezes em sequência.

A ideia de um laço de repetição é a execução de um trecho de código repetidas vezes, mas é importante deixar claro que a repetição não precisa ser necessária e exatamente das mesmas instruções.

É possível inserir, por exemplo, estruturas de decisão usando os comandos “SE” e “ESCOLHA” sem problemas em estruturas de repetição, obtendo assim, algoritmos de uma maior complexidade e capazes de realizar processamento automatizado, repetitivo e com tomada de decisão automática.

O princípio básico do uso de estruturas de repetição é a repetição de trechos de algoritmos para que certo conjunto de instruções seja repetido certo número de vezes, mas é comum que certos algoritmos sejam praticamente inteiros inseridos em uma estrutura de repetição, pois toda a sua funcionalidade é repetitiva e a própria estrutura de repetição mantém o algoritmo funcionando até que se decida encerrá-lo pela condição estabelecida para a continuidade da repetição.

Quando uma estrutura de repetição entra em funcionamento, seguidas iterações das instruções contidas na estrutura ocorrem até que determinada condição falhe e o laço seja encerrado, e este pode ser pré-determinado ou ser controlado pelo andamento da execução do algoritmo.

Estruturas de Repetição Condicional

Um tipo de estrutura de repetição bastante usado inicia com a verificação de uma condição para avaliar se as instruções contidas neste devem ser executadas ou não, e para isto, uma condição deve ser elaborada logo na primeira linha de comando da instrução.

Esta condição deve ser responsável por verificar a cada iteração, se o laço de instruções deve continuar sendo executado, ou se as mudanças ocorridas no decorrer da iteração alteraram o estado dos dados que são verificados na condição da estrutura e tornam o resultado da expressão condicional falso, encerrando-o.

Um exemplo de estrutura de repetição condicional em que a verificação do laço ocorre logo no início da instrução de controle é mostrado na imagem 22. Observe a forma como a lógica para a estrutura é elaborada de forma que após certa quantidade de vezes, o laço é encerrado automaticamente.

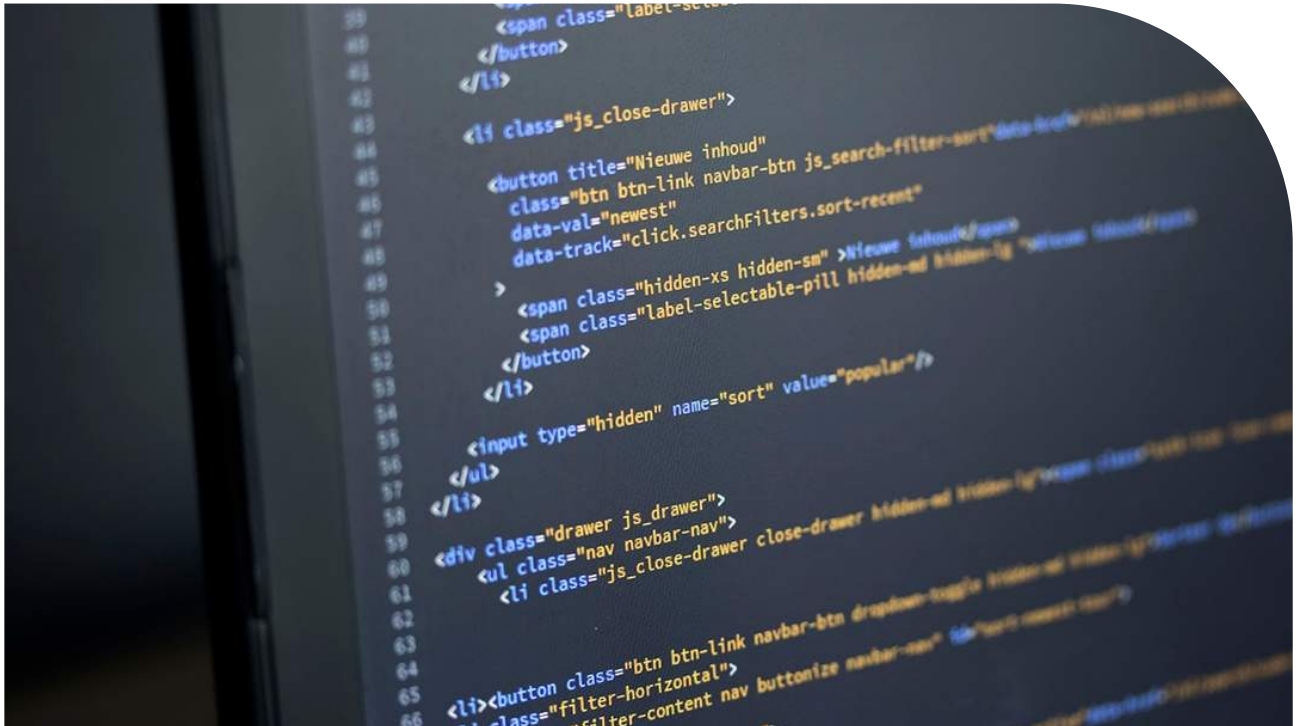
```
CONTADOR <- 1;  
ENQUANTO (CONTADOR <= 100) FAÇA  
    ESCREVA ("ITERAÇÃO ", CONTADOR);  
    CONTADOR <- CONTADOR +1;  
FIMENQUANTO;
```

Imagem 22: Exemplo de estrutura de repetição condicional. | Fonte: O autor.

Neste exemplo da imagem 22, é definida uma estrutura de repetição que utiliza uma variável "CONTADOR" para servir de base para a expressão condicional para repetição de iterações no laço definido com duas instruções a serem repetidas.

A condição verifica antes mesmo de a primeira iteração ocorrer, se o valor contido na variável "CONTADOR" é maior ou igual a 100, condição esta que define que enquanto o valor da variável for menor ou igual ao valor definido na condição, as iterações devem continuar seguidamente ocorrendo.

Alguns detalhes muito importantes neste exemplo da imagem 22 são inicialmente para reduzir riscos na execução adequada da estrutura, a variável “CONTADOR” recebe um valor zero para sua inicialização, pois pode ocorrer de haver algum valor inesperado na variável antes da verificação da condição que poderia prejudicar a execução adequada do laço de repetição ou até impedir seu início em caso de valor maior que 100.



Outro detalhe importante é que dentro do próprio laço de repetição é preciso criar condições para que a situação que torna falsa a expressão de validação de continuidade do laço de repetição aconteça se for desejado, pois caso contrário, o laço de repetição se repete infinitas vezes e isso pode não ser uma premissa do algoritmo.

Por último, é importante observar que a variável “CONTADOR” foi inicializada com o valor 1, mas este valor varia de acordo com a lógica do algoritmo e pode ser qualquer outro valor, da mesma forma que o valor da condição que define quando o laço deve ser interrompido e a forma como o valor da variável de controle é ajustada a cada iteração.

Em um segundo exemplo postado na imagem 23, outra forma de uso de laços de repetição é trazida para agregar novas ideias de como elaborar uma estrutura de repetição.

```
CONTADOR <- 1;  
REPITA  
  ESCREVA ("ITERAÇÃO ", CONTADOR);  
  CONTADOR <- CONTADOR + 1;  
ATÉ (CONTADOR > 100);
```

Imagem 23: Exemplo de estrutura de repetição condicional. | Fonte: O autor.

Aqui no exemplo da imagem 23, ocorre uma mudança que aparentemente afeta pouco o algoritmo em si, comparando-o com o exemplo da imagem 23, mas na verdade esta pequena mudança faz toda diferença muitas vezes.

O detalhe mais importante a ser observado para este exemplo é o que acontece com a condição que verifica se as iterações devem continuar ocorrendo ou não. Houve uma mudança de posição da expressão que avalia esta decisão e ela foi posicionada ao final da instrução que controla a estrutura de repetição.

Além disso, é muito importante observar que a lógica para este exemplo foi alterada em relação à construção da expressão de avaliação, pois foi necessária uma modificação na expressão condicional para que, ao invés de ser verificado se o valor da variável "CONTADOR" é menor ou igual a 100 para que o laço de repetição seja interrompido, agora, é verificado se o valor é maior que 100 como condição para que as iterações continuem ocorrendo.

CONECTE-SE

O uso de estruturas de repetição é bastante importante e permite que algoritmos realizem tarefas repetitivas com menos código. A escolha adequada entre as opções depende de preferências do programador, mas também é influenciada pelo tipo de processamento a ser realizado e da condição para que tal ocorra.

Acesse o link: [Disponível aqui](#)

Estruturas de Repetição Contada

Outro tipo de estrutura de repetição se baseia na possibilidade de se estabelecer previamente a quantidade de iterações que devem ocorrer logo na composição do comando de controle da estrutura de repetição.

Este controle é feito indicando uma variável para controle das iterações a partir de uma regra definida no próprio comando, indicando o valor inicial para a variável de controle, uma condição de continuidade das iterações e a forma como o valor da variável de controle vai sendo afetado a cada iteração.

A construção do comando pode ser feita de formas alternativas, mudando inclusive a proposta inicialmente citada, mas a princípio será tratada a estrutura de repetição contada como uma estrutura com a predefinição da quantidade de iterações na própria construção do comando.

Para isso, utiliza-se a palavra reservada “PARA” que na forma a ser estudada inicialmente, recebe três parâmetros que definem, respectivamente, um valor inicial para a variável de controle, depois o valor final para a variável de controle que definirá

um valor final para a quantidade de iterações, e por fim, uma regra para incremento ou decremento do valor da variável de forma a partir do valor inicial, e após certo número de iterações, o valor final ser atingido. Observe o exemplo da imagem 24.

```
PARA CONTADOR DE 1 ATÉ 100 PASSO 1 FAÇA  
  ESCREVA ("ITERAÇÃO ", CONTADOR);  
FIMPARA;
```

Imagem 24: Exemplo de estrutura de repetição contada. | Fonte: O autor.

No trecho de algoritmo da imagem 24, foi criada uma versão alternativa para a estrutura de repetição do exemplo da imagem 23 em que é possível primeiramente observar uma grande redução na quantidade de linhas de algoritmo necessárias para se obter uma mesma funcionalidade.

Um dos motivos para a redução foi a não necessidade de se inserir duas linhas de algoritmo extras para controle da variável "CONTADOR", pois todo o controle da estrutura de repetição é definido no comando principal da instrução.

O comando é construído inicialmente pela palavra reservada "PARA" e em seguida, pela variável de controle "CONTADOR". Depois a palavra reservada "DE" indica um valor inicial para a variável de controle.

Na sequência, a palavra reservada "ATÉ" contém um valor que serve como condição de parada para o laço de repetição, seguido da palavra reservada "PASSO" que define a regra de alteração do valor da variável "CONTADOR" a cada iteração. Por fim, a palavra reservada "FAÇA" inicia o conjunto de instruções que podem estar contidas no laço de repetição.

Após as instruções inseridas na estrutura de repetição, encerra-se a estrutura com a palavra reservada "FIMPARA", assim como no laço de repetição "ENQUANTO" utiliza-se a palavra reservada "FIMENQUANTO".

Em muitas linguagens de programação é possível controlar uma estrutura de repetição do tipo contada "PARA" de forma semelhante às citadas nas estruturas de repetição condicionais "ENQUANTO" e "REPITA".

Para isso, é necessário que se retire de dentro do comando de controle da estrutura de repetição alguns dos elementos que predeterminam a quantidade de iterações a serem realizadas. Em termos de algoritmos, seria algo semelhante ao que é trazido na imagem 25.

```
CONTADOR <- 1
PARA CONTADOR ATÉ 100 FAÇA
    ESCREVA ("ITERAÇÃO ", CONTADOR);
    CONTADOR <- CONTADOR + 1
FIMPARA;
```

Imagem 25: Exemplo de estrutura de repetição contada. | Fonte: O autor.

Com as alterações feitas no exemplo da imagem 24 para a obtenção do exemplo da imagem 25, houve aumento na quantidade de linhas necessárias para gerar o trecho de algoritmo com a mesma funcionalidade do anterior, mas retirando de dentro do comando da estrutura o controle total do laço de repetição.

Essa forma de se construir uma estrutura de repetição com base na palavra reservada "PARA" é muito comum no meio profissional, se aproximando do uso das demais palavras reservadas estudadas nesta aula, mas a escolha entre o uso de cada opção de comando depende do que é preciso fazer em cada problema e da forma como cada desenvolvedor aplica sua lógica na elaboração de soluções.

PARA GABARITAR



Estruturas de repetição são excelentes recursos para o desenvolvimento de algoritmos e é importante observar que o tipo de laço contado representado nesta aula pela palavra reservada “PARA” é uma opção que possui sintaxe que permite que todos os parâmetros colocados definam exatamente a quantidade de iterações a serem realizadas, mas é possível ajustar esta opção de laço omitindo partes dos parâmetros e utilizando-os fora do comando “PARA” de forma a poder controlar a quantidade de iterações a partir dos processos ocorridos durante as iterações.

O uso dos comandos “ENQUANTO...FAÇA” ou “REPITA...ATÉ” possui a particularidade de terem a diferença da condição colocada no início do laço ou no final, variando o controle realizado e tendo assim, a possibilidade de ocorrerem ou não iterações no laço.



Aninhamento de Estruturas

Aninhamento de Estruturas de Controle

Aos poucos, à medida que os estudos vão avançando na disciplina e novos conceitos vão sendo conhecidos, é normal que se possam ter pensamentos sobre como se podem interligar conceitos aprendidos até então e como integrar estes conceitos pode contribuir para um melhor aprendizado.

Nesta disciplina, os conceitos são todos complementares e a cada novo conceito, este pode ser integrado aos demais, pois é comum em algoritmos maiores que praticamente todas as estruturas e palavras reservadas estudadas até então sejam utilizadas.

Assim, é mais fácil compreender o todo em relação ao aprendizado da disciplina, e também facilita o aprendizado dos conceitos separados, pois são frequentemente reaplicados em novas aulas deste material.



É importante utilizar o espaço desta aula para uma complementação do que foi estudado até o momento de forma a mostrar de que maneira os conceitos estudados podem contribuir como desenvolvimento de algoritmos.



Por meio de alguns exemplos comentados e alguns apontamentos pertinentes, a ideia é analisar as possibilidades de criação de algoritmos a partir do que já se sabe e preparar o terreno para a adição de novos conceitos nas aulas seguintes.

Como estas estruturas possuem uma maior complexidade lógica, será usado um artifício bastante útil em teste de algoritmos chamado de teste de mesa, em que se utilizam amostras de dados escolhidas aleatoriamente ou de forma tendenciosa para simular a execução de algoritmos e buscar possíveis falhas se houver.

Para cada exemplo desta aula e em outras de aulas posteriores vamos demonstrar a lógica e funcionalidade de algoritmos que possam ser mais complexos e necessitem de esclarecimento adicional mais prático.

Os testes de mesa são utilizados por autores como Manzano (1997) como meios para exibir dados a serem utilizados em simulações de algoritmos e também para conter os dados obtidos com o processamento dos dados escolhidos para testes.

Aninhamento de Estruturas de Decisão

O uso de estruturas condicionais é bastante comum, pois, como já dito, permitem que sejam tomadas decisões durante a execução de um algoritmo de forma automatizada, pelo próprio algoritmo.

Seu uso é simples em sua estrutura, mas a elaboração de expressões pode ser complexa e necessitar de boa análise lógica para prever os possíveis resultados e erros na formulação da expressão de controle.

Também é importante ter bem clara a composição de estruturas de decisão de forma que uma estrutura posta dentro de outra estrutura apenas pode ser executada em caso de a estrutura na qual está inserida obter resultado verdadeiro em sua expressão condicional.

Observe o exemplo a seguir contido na imagem 26 para compreender melhor como se podem organizar as alternativas de execução de um algoritmo por meio de estruturas aninhadas de decisão.

```

SE (A >= B) ENTÃO
SE (A >= C) ENTÃO
    ESCRIVA ("A PODE SER O MAIOR VALOR");
SENÃO SE (C >= A) ENTÃO
    ESCRIVA ("C PODE SER O MAIOR VALOR");
SENÃO
    SE (B >= C) ENTÃO
    ESCRIVA ("B PODE SER O MAIOR VALOR");
SENÃO
    ESCRIVA ("C PODE SER O MAIOR VALOR");
    ESCRIVA ("DEVE HAVER NÚMEROS IGUAIS");
FIMSE.

```

Imagem 26: Exemplo de estrutura de repetição contada.

TESTE DE MESA

A	B	C	RESULTADO
1	2	3	C PODE SER O MAIOR VALOR
2	3	1	B PODE SER O MAIOR VALOR
3	2	1	A PODE SER O MAIOR VALOR
1	1	1	DEVE HAVER NÚMEROS IGUAIS

Fonte: O autor.

Neste exemplo da imagem 26 é exibido um trecho de algoritmo contendo um encadeamento de estruturas condicionais para verificar três valores e determinar qual o maior em caso de todos serem diferentes e uma análise profunda ser possível, ou indicar que deve haver números iguais, dificultando a avaliação do maior.



A estrutura de decisão poderia ser mais completa e avaliar mais casos, pois é importante ter a noção de que para três variáveis, é possível uma série de combinações de comparações que podem ser feitas entre estes valores, resultando em diferentes cenários.

Importante é compreender a lógica do encadeamento de estruturas de forma que algumas comparações são realizadas apenas se a anterior for verdadeira na primeira estrutura encadeada.

A segunda estrutura encadeada depende da primeira resultar falsa para ser avaliada, e assim, a forma como se estruturam encadeamentos de estruturas devem ser bem planejadas para evitar problemas em sua execução.

Aninhamento de Estruturas de Repetição

Continuando com os estudos de integração entre conceitos vistos nas aulas anteriores, é o momento de avaliar melhor o uso das estruturas de repetição com a intenção de agrupar estruturas de repetição de forma aninhada para que se possam realizar duas ou mais iterações paralelamente.

Este tipo de recurso é importante em situações como no caso de ser necessário realizar leituras ou atualizações de dados em estruturas multidimensionais que serão estudadas mais adiante, mas também em casos em que sejam necessários contadores ocorrendo de forma complementar como em ponteiros de relógios, contadores do tipo hodômetro, etc.

Este tipo de estrutura pode ser confuso inicialmente, mas depois de compreendido, é um recurso útil e muito utilizado nas estruturas multidimensionais pela facilidade de sua implementação e funcionamento preciso.

```
PARA I DE 0 ATÉ 9 PASSO 1 FAÇA  
  PARA J DE 0 ATÉ 0 PASSO 1 FAÇA  
    PARA K DE 0 ATÉ 9 PASSO 1 FAÇA  
      ESCREVA (I, J, K);  
    FIMPARA;  
  FIMPARA;  
FIMPARA;
```

Imagem 27: Exemplo de estrutura de repetição aninhada. | Fonte: autor.

Neste exemplo da imagem 27 é implementado um trecho de algoritmo para simular o funcionamento de um hodômetro com três dígitos, e para isto, são necessárias três variáveis para cada um dos três dígitos utilizados neste algoritmo.

Cada dígito do hodômetro deve ser trocado em um momento específico que é quando o dígito a sua direita completar um ciclo completo de 0 a 9, exceto pelo primeiro dígito da direita que representa a unidade a partir da qual os demais dígitos sucessivamente se baseiam para suas trocas de valores.

Com três dígitos, o resultado do comando “ESCREVA” será a exibição dos dígitos de centena, dezena e unidade de 000 a 999, sequencialmente, como ocorre em contadores manuais ou digitais.

**ABORDAGEM
PRÁTICA**

O aninhamento ou encadeamento de estruturas de controle de mesmo tipo é bastante utilizado na elaboração de algoritmos e permite que algoritmos com necessidades mais complexas possam ser elaborados.

Estruturas de dados maiores que variáveis, por exemplo, necessitam de recursos mais complexos para sua manipulação, assim como condições que são avaliadas em sequência podem ser encadeadas de forma a se complementarem, e assim, permitir situações nas quais uma avaliação é realizada apenas dependendo do resultado obtido em uma condição anterior realizada necessariamente primeiro.

Estruturas de repetição encadeadas permitem que dois contadores ou mais estejam trabalhando em conjunto como nos números de um hodômetro ou ponteiros de um relógio, e assim, cada vez mais funcionalidades são obtidas pela combinação de estruturas de controle.

Estruturas de Repetição em Estruturas de Decisão

Seguindo em frente com os estudos mais avançados no uso de estruturas de controle, é possível combinar estruturas diferentes de forma que possam ser integradas e realizem processamento mais complexo.

Estruturas de repetição podem ser inseridas dentro de estruturas de decisão de forma que determinado laço de repetição seja acionado apenas se determinada condição for satisfeita, para que a partir deste desvio de fluxo, a expressão de controle do laço seja avaliada para que sejam ou não iniciadas as iterações.

```
VALOR <-10;  
SE (VALOR > 0) ENTÃO  
    PARA I DE 1 ATÉ VALOR PASSO 1 FAÇA  
        ESCREVA (I);  
    FIMPARA;  
FIMSE;
```

Imagem 28: Exemplo de estrutura de repetição em estrutura de decisão. | Fonte: O autor.

No exemplo trazido na imagem 28, é colocado um laço de repetição que exibirá valores entre 1 e o número contido na variável “VALOR”, inicialmente atribuído com 10, mas isto só ocorre em função da verificação da condição do comando “SE” resultar verdadeiro, pois “VALOR” possui número maior que 0.

Estruturas de Decisão em Estruturas de Repetição

Por último e não menos importante, existe outra forma de se combinar estruturas de controle de fluxo em algoritmos na qual estruturas de decisão podem ser inseridas em estruturas de repetição para que possam ser avaliadas novamente a cada iteração de um laço de repetição.

Um exemplo seria no caso do preenchimento de uma estrutura mais complexa de dados que utilizaria um laço de repetição para percorrer todos os espaços para armazenamento, mas os dados destes seriam atualizados apenas em caso de uma expressão avaliada a cada iteração resultar verdadeiro, por exemplo.

```
VALOR <- 10;  
PARA I DE 1 ATÉ VALOR PASSO 1 FAÇA  
  SE (I=5) ENTÃO  
    I <- VALOR;  
  FIMSE;  
FIMPARA;
```

Imagem 29: Exemplo de estrutura de decisão em estrutura de repetição. | Fonte: autor.

Neste outro exemplo contido agora na imagem 29, é inicializada uma variável “VALOR” com 10 para que possa servir de base para um laço de repetição que inicialmente está programado para realizar 10 iterações com uma variável contadora “I” variando de 1 a 10 pela definição do comando “PARA”.

Existe internamente ao laço de repetição, uma estrutura condicional que verifica a cada iteração o conteúdo da variável “VALOR”, e caso seja igual a 5, força o conteúdo da variável “VALOR” ser ajustado para 10, encerrando as iterações na próxima verificação da condição de parada das repetições.

PARA GABARITAR



O chamado aninhamento de estruturas de controle é um recurso muito útil e acrescenta muitos recursos no desenvolvimento de algoritmos, mas também traz um aumento na complexidade de algoritmos que pode confundir iniciantes.

É importante praticar o desenvolvimento de pequenos algoritmos contendo estruturas de controle variadas para compreender a mecânica dos aninhamentos e como uma estrutura possui influência sobre a outra, além de compreender melhor os motivos necessários para aninhar estruturas e como definir quais tipos são adequados a cada situação.



Estruturas Homogêneas



Estruturas de Dados Homogêneas Unidimensionais

As estruturas de dados como vêm sendo estudadas ao longo deste material, representam um dos conceitos mais importantes no desenvolvimento de algoritmos, pois a ideia central dos algoritmos é o processamento de dados.

Até o momento foram utilizadas estruturas de dados simples para armazenamento de dados únicos em forma de variáveis que eram declaradas e tinham um tipo de dado atribuído para que estes tipos de dados declarados pudessem ser atribuídos a elas.

A partir desta aula, será ampliada a possibilidade de armazenamento temporário e manipulação de dados durante a execução de algoritmos por meio de um novo tipo de estrutura de dados.

Existe a possibilidade de se organizar uma maior quantidade de dados em forma de espécies de listas de dados de um mesmo tipo chamadas de estruturas de dados homogêneas.

Este tipo de estrutura pode ser declarado com um dos tipos de dados simples estudados, mas todos os elementos da estrutura deverão conter dados deste mesmo tipo a princípio.



Existem linguagens de programação que não necessitam que um tipo de dado seja definido durante a declaração da variável e o tipo de dado acaba sendo dinamicamente definido ao longo da execução do *software* de acordo com o primeiro tipo de dado recebido, por exemplo.

Mas para os estudos neste material sempre será considerado o tipo definido logo na declaração de uma estrutura de dados para efeito em todo o algoritmo e entradas de dados de tipos incorretos ou atribuições de um tipo de dado para uma estrutura serão considerados erros em caso da execução de um algoritmo.

Quando forem realizados testes de mesa em algoritmos, é importante observar os tipos definidos e respeitá-los para melhor desenvolvimento dos testes, a menos que se queira buscar por erros em função da ausência de uma validação em entradas de dados.

Quando se imagina uma estrutura de dados homogêneas então, é preciso pensar numa sequência de variáveis todas de um mesmo tipo e nomeadas de forma única para efeitos de elaboração de algoritmos.

Este tipo de estrutura é interessante pela possibilidade de organizar listas de dados que podem ser adequadas para casos em que um mesmo tipo de dado ocorre diversas vezes em um algoritmo como no caso de listas de nomes de pessoas,

números de senhas de espera, e qualquer tipo de lista de dados com um mesmo significado.

Este tipo de estrutura normalmente é nomeado como uma variável qualquer, mas seu identificador é acompanhado por um valor inteiro que define a quantidade máxima de dados suportados pela estrutura. Observe o exemplo a seguir.

1	2	3	4	5	6	7	8	9	10

Exemplo de estrutura de decisão em estrutura de repetição. | Fonte: O autor.

Pela representação gráfica trazida na imagem 30, percebe-se que cada elemento desse tipo de estrutura de dado homogênea reserva um espaço em memória para armazenamento de um único dado por vez, assim como ocorre com as variáveis.

Este tipo de estrutura possibilita várias funcionalidades adicionais ao que foi estudado até então, e simplifica a manipulação de uma maior quantidade de dados através de uma única estrutura de dados com um único nome.

Por meio de um índice associado a cada elemento da estrutura, esta pode ser facilmente manipulada pelas estruturas de controle para manipulação de seus dados, tornando este tipo de estrutura ideal para a organização de dados em forma de listas com quantidade limitada de elementos.

Laços de repetição podem ser utilizados para se mover pelos índices de uma estrutura de dados desse tipo, de forma sequencial para a realização de buscas, inserções de dados e edições, e estruturas condicionais podem permitir diferentes ações a serem realizadas com elementos da estrutura.

Este tipo de estrutura é chamada de vetor ou matriz unidimensional, mas para os estudos neste material, optou-se a utilização do termo vetor para diferenciação com matrizes que serão vistas em aula posterior.

Vetores

Os vetores representam, então, estruturas de dados homogêneas que contêm uma quantidade pré-definida de dados de um mesmo tipo e que possuem índices para identificação de cada posição do vetor.

Para se trabalhar com vetores em algoritmos existe uma sintaxe que será adotada neste material para declarar e posteriormente para o uso destes em conjunto com os demais conteúdos estudados até o momento. Observe o exemplo da imagem 31 para conhecer a forma como será padronizada a declaração de vetores.

```
TIPO DADOS = VETOR [1..100] DE INTEIROS;  
DADOS : LISTA;
```

Imagem 31: Exemplo de estrutura de decisão em estrutura de repetição. | Fonte: O autor.

Neste exemplo de declaração de vetor contido na imagem 40, utiliza-se como base a sintaxe definida por Forbellone (2005), em que a palavra reservada “TIPO” para iniciar a declaração do tipo vetor, de forma que o mesmo possa ser utilizado como tipo para estruturas de dados desejadas num algoritmo.

O nome “DADO” é um identificador escolhido pelo desenvolvedor, e após o sinal de igualdade, é indicado que está sendo declarado um vetor com uma determinada quantidade de elementos em colchetes “[1..100]” do tipo inteiro indicado como “DE INTEIROS” para completar a instrução de definição do tipo vetor.

Na sequência, é declarada uma estrutura de dados “LISTA” com o uso do tipo definido “DADOS” da linha anterior para que seja então declarado um vetor com até 100 elementos.

CONECTE-SE

O estudo de vetores é bastante interessante, pois permite que determinados tipos de aplicação possam ser implementados mais facilmente como listas de dados que permitam buscas e manipulações.

Acesse o link: [Disponível aqui](#)

A manipulação de dados em vetores é bastante simples, mas é preciso que se tomem os devidos cuidados com a correta utilização da estrutura, pois erros na elaboração de algoritmos que manipulem estas estruturas podem acarretar perdas de maiores volumes de dados.

```
INÍCIO
DECLARE
TIPO DADOS = VETOR [1..10] DE INTEIROS;
DADOS : LISTA;
INTEIRO : I, BUSCA;
PARA I DE 1 ATÉ 10 PASSO 1 FAÇA
    LEIA (LISTA [ I ]);
FIMPARA;
ESCREVA ("DIGITE UM VALOR PARA BUSCA NO VETOR: ");
LEIA (BUSCA);
PARA I DE 1 ATÉ 10 PASSO 1 FAÇA
    SE (LISTA [I] = BUSCA) ENTÃO
        ESCREVA ("VALOR ENCONTRADO NA POSIÇÃO ", I);
    FIMSE;
FIMPARA;
FIM.
```

Imagem 32: Exemplo de estrutura de decisão em estrutura de repetição. | Fonte: O autor.

No algoritmo exemplo da imagem 32, é declarado um tipo vetor de inteiros chamado “DADOS” com até 10 números para exemplificar a criação e uso de uma lista de números inteiros. Em seguida, uma variável “LISTA” é declarada a partir do tipo “DADOS” definido. Na sequência, um laço de repetição é utilizado para que dados sejam inseridos com o comando “LEIA” em todas as dez posições possíveis do vetor.

Depois, numa próxima etapa, é solicitado ao usuário que digite um número a ser pesquisado no vetor que será armazenado na variável “BUSCA” para dentro de outro laço de repetição que passará por todo o vetor, ter seu valor comparado com o valor armazenado em cada posição do vetor. Caso seja encontrado um valor do vetor igual ao da variável que armazena o número indicado pelo usuário, uma mensagem informando a posição onde estava o valor igual é exibida.

PARA GABARITAR



Os vetores são estruturas muito úteis em situações em que seja necessário o armazenamento de listas de dados, mas sua manipulação necessita de treino para que seja possível a inserção de estruturas deste tipo em algoritmos sem que a complexidade de uso destas estruturas se torne um problema.

A ideia é criar estruturas homogêneas com tamanhos diversos e tipos variados para se conhecer os meios de manipulação para cada tipo de dado e saber trabalhar com os índices que identificam os elementos em vetores.

Uma variação para o algoritmo trazido na imagem 33 é mostrada na imagem 41 que altera uma das estruturas de controle para que o algoritmo se torne mais eficiente em sua execução.

```
INÍCIO
DECLARE
TIPO DADOS = VETOR [1..10] DE INTEIROS;
DADOS : LISTA;
INTEIRO : I, BUSCA;
PARA I DE 1 ATÉ 10 PASSO 1 FAÇA
    LEIA (LISTA [ I ]);
FIMPARA;
ESCREVA ("DIGITE UM VALOR PARA BUSCA NO VETOR: ");
LEIA (BUSCA);
I <- 1;
ENQUANTO (LISTA [I] <> BUSCA OU I <= 10) FAÇA
    SE (LISTA [I] = BUSCA) ENTÃO
        ESCREVA ("VALOR ENCONTRADO NA POSIÇÃO ", I);
    FIMSE;
I <- I + 1;
FIMENQUANTO;
FIMPARA;
FIM.
```

Imagem 33: Exemplo de estrutura de decisão em estrutura de repetição. | Fonte: O autor.

Nesta variação de algoritmo da imagem 33, após a mesma definição de tipo vetor, declaração de variável vetor, e preenchimento dos dados no mesmo, como uma primeira etapa do algoritmo, a segunda parte sofreu alterações.

Após a solicitação do número a ser pesquisado, a variável "I" é reinicializada com valor 1 e um laço de repetição condicional substitui o contado, e com a possibilidade indicada de interrupção das iterações baseadas ou na ausência de número igual no vetor ao solicitado, ou não término do vetor, a estrutura segue por cada posição do vetor comparando os valores e controlando a quantidade de iterações realizadas com o incremento manual em uma unidade da variável contadora a cada iteração.

9 Do

10 Fr

11 Sa

12 So

Estruturas Homogêneas Multidimensionais

Matrizes

Uma estrutura multidimensional homogênea possui diversas finalidades práticas como armazenamento temporário, e em certos casos é até difícil imaginar seu uso, mas a base para muitos softwares são matrizes.

Um exemplo importante é a própria formação da uma imagem na tela de um computador. As imagens são formadas por pixels que podem ser representados por matrizes.

Assim, em uma tela com resolução 1024x768 pixels, por exemplo, é possível representar cores que poderiam ser inseridas em cada ponto da tela com um valor que pode representar a cor relativa a cada pixel responsável por compor uma imagem.

Mesmo na exibição de vídeos, a ideia da matriz se mantém, pois uma imagem estática é facilmente interpretada como uma matriz de valores, mas é preciso compreender que um vídeo também é composto por imagens estáticas que são trocadas a determinada velocidade medida em quadros (imagens) por segundo. Observe a imagem 34 para um exemplo de imagem.

XY	1	2	3	4	5	6	7	8	9	10
1	0	0	0	50	50	50	50	0	0	0
2	0	0	50	50	50	50	50	50	0	0
3	0	0	50	50	50	50	50	50	0	0
4	0	0	50	50	20	50	20	50	0	0
5	0	0	50	50	50	35	50	50	0	0
6	0	0	50	50	50	50	50	50	0	0
7	0	0	50	50	20	20	20	50	0	0
8	0	0	50	50	50	50	50	50	0	0
9	0	0	0	50	50	50	50	0	0	0
10	0	0	0	50	50	50	50	0	0	0

Imagem 34: Matriz representativa de uma imagem com 10x10 pixels. | Fonte: O autor.

Na representação da matriz na imagem 34, observa-se que temos 10 linhas e 10 colunas representando uma estrutura de dados para números com a possibilidade de armazenamento temporário de até 100 valores pela multiplicação simples entre a quantidade de linhas e colunas.

Pela disposição dos valores na matriz, é possível perceber que apenas na posição indicada pela linha 5 e coluna 6, temos o único valor 35 na matriz, servindo de exemplo para a compreensão do uso de índices de linha e coluna para indicação de dados em uma matriz. Neste caso, 5x6.

Outra aplicação comum é para a elaboração de jogos de tabuleiro em que matrizes podem conter peças indicadas por números representativos de peças a serem movimentadas. Xadrez, damas, e outros tipos como palavras cruzadas, por exemplo.

Jogos mais complexos também podem ser elaborados com base em matrizes cuja ideia de posicionar elementos pela tela se mantém e diversos tipos de informações podem ser inseridos neste tipo de estrutura como a posição de elementos espalhados pela matriz, ou seja, pela tela de jogo. Observe a imagem 35 para um exemplo de jogo que pode ter seus elementos dispostos em uma matriz de duas dimensões.



Imagem 35: Exemplo de estrutura de decisão em estrutura de repetição. | Fonte: [Disponível aqui](#)

Na imagem 35, é possível imaginar os elementos que se movem pela tela sendo controlados a partir de referências indicadas numa matriz que serve de base para que todas as posições dos elementos sejam conhecidas e possam ser modificadas durante o jogo.

Para se declarar uma estrutura que possa ser utilizada para matrizes, a sintaxe básica inicia pela palavra reservada “TIPO”, seguida pelo nome escolhido para a matriz. Em seguida utiliza-se sempre a sequência “= MATRIZ” e as dimensões e intervalo de valores aceitos na forma “[1..N][1..M]...[1..Z]”, em que cada par de colchetes indicam

uma dimensão. Por fim, inclui-se depois a palavra reservada “de” e finalmente um tipo de dado para declarar a estrutura e o ponto e vírgula padrão da sintaxe. Observe os exemplos da imagem 36.

```
TIPO TABULEIRO = MATRIZ [1..8][1..8] DE CARACTERES;  
TIPO VELHA = MATRIZ [1..3][1..3] DE CARACTERES;  
TIPO CUBOMAGICO = MATRIZ [1..3][1..3][1..3] DE INTEIRO;
```

Imagem 36: Exemplos de declaração de matrizes. | Fonte: O autor.

Nos exemplos trazidos na imagem 36 temos as declarações de matrizes para situações bastante comuns como para um tabuleiro de damas ou xadrez no primeiro caso com 8 linhas e 8 colunas. No segundo exemplo, é declarada uma matriz chamada “VELHA” que pode ser usada como base para um jogo da velha, e por fim, no terceiro exemplo, uma matriz de três dimensões que pode ser base para a implementação de um cubo mágico.

CONECTE-SE



Estruturas de dados homogêneas são importantes para o desenvolvimento de algoritmos e permitem que em uma ou mais dimensões, dados sejam armazenados e manipulados de forma organizada. Procurem sempre aprofundamentos no uso deste tipo de estrutura de dados.

Acesse o link: [Disponível aqui](#)

Aplicação Prática de Matrizes

Para um exemplo completo de uso de matriz, foi escolhido como tema o jogo da velha que é bastante simples em suas regras, compreensão e adaptação para algoritmo, sendo então um bom exemplo a ser desenvolvido e compreendido.

Foi escolhida uma sintaxe alternativa que pode ser utilizada no *software* VisualG 3.0 disponível em <https://sourceforge.net/projects/visualg30/>. Esta ferramenta permite a execução de algoritmos em português e foi desenvolvida pelo professor Antonio Carlos Nicolodi. Observe a imagem 45 para conhecer a interface da ferramenta de interpretação de algoritmos.

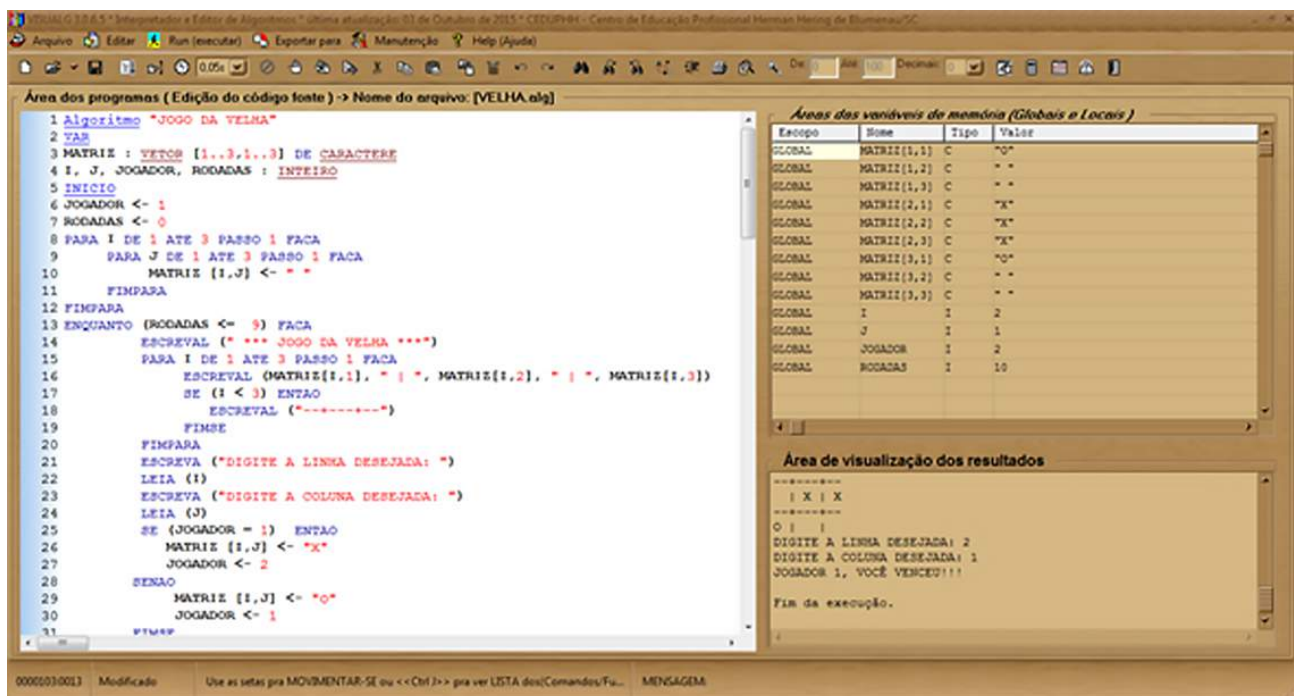


Imagem 45: Exemplo de estrutura de decisão em estrutura de repetição. | Fonte: O autor.

A sintaxe para a ferramenta é um tanto distinta em relação ao que está sendo estudado, mas a lógica em si se mantém a mesma, servindo perfeitamente como exemplo para interpretação e teste de um algoritmo completo e funcional que pode ser jogado. A seguir, na imagem 37, temos o algoritmo completo para estudos e teste na ferramenta, se desejado.

```

ALGORITMO "JOGO DA VELHA"
VAR
MATRIZ : VETOR [1..3,1..3] DE CARACTERE
I, J, JOGADOR, RODADAS : INTEIRO
INÍCIO
JOGADOR <- 1
RODADAS <- 0
// Inicialização da matriz para que todas as posições estejam em
branco.
PARA I DE 1 ATE 3 PASSO 1 FACA
    PARA J DE 1 ATE 3 PASSO 1 FACA
        MATRIZ [I,J] <- " "
    FIMPARA
FIMPARA
// Laço de repetição que para só se alguém ganha ou acabam os
espaços.
ENQUANTO (RODADAS <= 9) FACA
    // Desenha o jogo da velha atualizado.
    ESCREVAL (" *** JOGO DA VELHA ***")
    PARA I DE 1 ATE 3 PASSO 1 FACA
        ESCREVAL (MATRIZ[I,1], " | ", MATRIZ[I,2], " | ",
MATRIZ[I,3])
        SE (I < 3) ENTAO
            ESCREVAL ("---+---+---")
        FIMSE
    FIMPARA
    // Solicita as coordenadas dos jogador a cada rodada.
    ESCREVA ("DIGITE A LINHA DESEJADA: ")
    LEIA (I)
    ESCREVA ("DIGITE A COLUNA DESEJADA: ")
    LEIA (J)
    SE (JOGADOR = 1) ENTAO
        MATRIZ [I,J] <- "X"
        JOGADOR <- 2
    SENA0
        MATRIZ [I,J] <- "O"
        JOGADOR <- 1
    FIMSE
    RODADAS <- RODADAS + 1
    // Inicia a avaliação de vitória de cada jogador a cada rodada.
    SE ((MATRIZ[1,1] = "O") E (MATRIZ[1,2] = "O") E (MATRIZ[1,3]
= "O")) ENTAO
        ESCREVAL ("JOGADOR 2, VOCÊ VENCEU!!!")
        RODADAS <- 10
    FIMSE

```

```

        SE ((MATRIZ[2,1] = "O") E (MATRIZ[2,2] = "O") E (MATRIZ[2,3]
= "O")) ENTAO
            ESCREVAL ("JOGADOR 2, VOCÊ VENCEU!!!")
            RODADAS <- 10
        FIMSE
        SE ((MATRIZ[3,1] = "O") E (MATRIZ[3,2] = "O") E (MATRIZ[3,3]
= "O")) ENTAO
            ESCREVAL ("JOGADOR 2, VOCÊ VENCEU!!!")
            RODADAS <- 10
        FIMSE
        SE ((MATRIZ[1,1] = "O") E (MATRIZ[2,1] = "O") E (MATRIZ[3,1]
= "O")) ENTAO
            ESCREVAL ("JOGADOR 2, VOCÊ VENCEU!!!")
            RODADAS <- 10
        FIMSE
        SE ((MATRIZ[1,2] = "O") E (MATRIZ[2,2] = "O") E (MATRIZ[3,2]
= "O")) ENTAO
            ESCREVAL ("JOGADOR 2, VOCÊ VENCEU!!!")
            RODADAS <- 10
        FIMSE
        SE ((MATRIZ[1,3] = "O") E (MATRIZ[2,3] = "O") E (MATRIZ[3,3]
= "O")) ENTAO
            ESCREVAL ("JOGADOR 2, VOCÊ VENCEU!!!")
            RODADAS <- 10
        FIMSE
        SE ((MATRIZ[1,1] = "O") E (MATRIZ[2,2] = "O") E (MATRIZ[3,3]
= "O")) ENTAO
            ESCREVAL ("JOGADOR 2, VOCÊ VENCEU!!!")
            RODADAS <- 10
        FIMSE
        SE ((MATRIZ[1,3] = "O") E (MATRIZ[2,2] = "O") E (MATRIZ[3,1]
= "O")) ENTAO
            ESCREVAL ("JOGADOR 2, VOCÊ VENCEU!!!")
            RODADAS <- 10
        FIMSE

        SE ((MATRIZ[1,1] = "X") E (MATRIZ[1,2] = "X") E (MATRIZ[1,3]
= "X")) ENTAO
            ESCREVAL ("JOGADOR 1, VOCÊ VENCEU!!!")
            RODADAS <- 10
        FIMSE
        SE ((MATRIZ[2,1] = "X") E (MATRIZ[2,2] = "X") E (MATRIZ[2,3]
= "X")) ENTAO
            ESCREVAL ("JOGADOR 1, VOCÊ VENCEU!!!")
            RODADAS <- 10
        FIMSE

```



```

        SE ((MATRIZ[3,1] = "X") E (MATRIZ[3,2] = "X") E (MATRIZ[3,3]
= "X")) ENTAO
            ESCREVAL ("JOGADOR 1, VOCÊ VENCEU!!!")
            RODADAS <- 10
        FIMSE
        SE ((MATRIZ[1,1] = "X") E (MATRIZ[2,1] = "X") E (MATRIZ[3,1]
= "X")) ENTAO
            ESCREVAL ("JOGADOR 1, VOCÊ VENCEU!!!")
            RODADAS <- 10
        FIMSE
        SE ((MATRIZ[1,2] = "X") E (MATRIZ[2,2] = "X") E (MATRIZ[3,2]
= "X")) ENTAO
            ESCREVAL ("JOGADOR 1, VOCÊ VENCEU!!!")
            RODADAS <- 10
        FIMSE
        SE ((MATRIZ[1,3] = "X") E (MATRIZ[2,3] = "X") E (MATRIZ[3,3]
= "X")) ENTAO
            ESCREVAL ("JOGADOR 1, VOCÊ VENCEU!!!")
            RODADAS <- 10
        FIMSE
        SE ((MATRIZ[1,1] = "X") E (MATRIZ[2,2] = "X") E (MATRIZ[3,3]
= "X")) ENTAO
            ESCREVAL ("JOGADOR 1, VOCÊ VENCEU!!!")
            RODADAS <- 10
        FIMSE
        SE ((MATRIZ[1,3] = "X") E (MATRIZ[2,2] = "X") E (MATRIZ[3,1]
= "X")) ENTAO
            ESCREVAL ("JOGADOR 1, VOCÊ VENCEU!!!")
            RODADAS <- 10
        FIMSE
// Esta opção é para o caso de ser preenchida a matriz sem
vitorioso.
        SE (RODADAS = 9) ENTAO
            ESCREVAL ("PARTIDA EMPATADA!!!")
        FIMSE
    FIMENQUANTO
FINALGORITMO

```

Imagem 37: Exemplo jogo da velha para aplicação na ferramenta VisualG. | Fonte: O autor.

Neste exemplo da imagem 37, temos um algoritmo implementado para simular uma versão simples do jogo da velha no qual primeiramente é declarada uma matriz de caracteres com 3 linhas e 3 colunas chamada "MATRIZ" para conter os caracteres "X" e "O" utilizados a cada rodada no jogo.



Depois são declaradas as variáveis “I” e “J” do tipo inteiro para serem utilizadas como índices para posicionamento na matriz que representa o jogo em si para armazenamento dos símbolos que representam as coordenadas escolhidas por cada jogador a cada rodada.

Outra variável chamada “JOGADOR” é declarada para indicar a vez de quem está jogando poder informar as coordenadas de seu palpite a cada rodada. Na sequência do algoritmo, é utilizada uma instrução de atribuição para a variável “JOGADOR” para indicar quem iniciará a partida.

Por fim, a variável “RODADAS” é declarada para conter a quantidade de rodadas já jogadas e também para ser utilizada e saber se o jogo chegou ao fim sem vencedor, fato que encerra a estrutura de repetição condicional e também se chega ao final da execução do algoritmo.

Outra inicialização é feita por segurança utilizando um laço de repetição aninhado dentro de outro para percorrer toda a matriz, atribuindo um espaço em branco como dado para que não haja risco de a estrutura conter dados indevidos ou até “X” ou “O” por coincidência. Em um algoritmo não é algo essencial, mas em muitas linguagens de programação é uma prática que melhora a funcionalidade adequada de um *software*.

Na sequência, inicia-se um laço de repetição controlado pela variável “RODADAS” em que é verificado se o valor da variável “RODADAS” já atingiu o valor limite 9 que representa a matriz cheia de “X” ou “O”. Este laço de repetição é fundamental, pois ele controla o jogo em si a cada rodada até que haja um vencedor ou a matriz seja toda preenchida sem um vencedor.

Dentro do laço, inicialmente é utilizado um pequeno laço para desenhar o jogo da velha para que o usuário possa acompanhar o andamento do jogo e para melhorar a interface, mesmo que de forma simples e textual.

Depois, são pedidas as coordenadas (linha e coluna) a serem preenchidas com “X” ou “O” de acordo com a vez de cada participante, controladas pela variável “JOGADOR” que alterna de valor a cada rodada, ao mesmo tempo em que as coordenadas informadas são utilizadas para o preenchimento da posição da matriz correspondente com o símbolo correspondente de cada jogador.

Em seguida, entra uma parte grande do código onde cada possibilidade de vitória do jogador representado pelo símbolo “O” é verificada (linhas, colunas ou diagonais com o mesmo símbolo), e caso ocorra, o valor da variável “RODADAS” é atualizado para 10

para que seja encerrado o laço de repetição, ao mesmo tempo em que é exibida uma mensagem de vitória ao jogador.

O mesmo é feito para o outro jogador, com base nas mesmas regras, mudando apenas o símbolo para “X” e a mensagem sendo adequada ao outro jogador vitorioso, caso ocorra uma das possibilidades.

Por fim, é feita uma verificação se a matriz não foi preenchida sem vitorioso com a quantidade de rodadas chegando a 9 e sem nenhuma das condições de vitória tendo sido atendida, informando aos usuários que não houve vencedor.

Com esse exemplo da imagem 46, é possível perceber que os algoritmos representam importante ferramenta de aprendizado da programação e podem auxiliar em posterior estudo de qualquer linguagem de programação, pois a lógica em si se mantém, bastando adaptações de sintaxe para adequação aos padrões de cada diferente linguagem.

PARA GABARITAR



Uma boa prática para melhor aprendizado deste tipo de estrutura de dados é a implementação de mudanças em códigos testados para mudar funcionalidades, acrescentar validações de dados, alterar dimensões e limites na quantidade de dados por dimensão, etc.

Da mesma forma que foi trazido um exemplo de algoritmo para o jogo da velha utilizando matriz, pode-se, além de alterar seu algoritmo para acréscimo de melhorias e validações para o preenchimento da matriz, assim como tentar novos algoritmos usando matrizes como caça-palavras, batalha naval, etc.

Estruturas de Dados Heterogêneas

Estruturas de dados heterogêneas

As estruturas de dados vêm sendo incrementadas nas duas últimas aulas e a quantidade de dados que podem ser inseridos nestas estruturas é grande, mas com a limitação de que todos os dados sejam de um mesmo tipo.

Existe um tipo adicional de estrutura de dados a ser estudado neste material que permite que uma mesma estrutura possa conter dados de diferentes tipos sob um mesmo nome, mas é importante observar que a forma como se referenciam os dados em estruturas deste tipo é diferente das demais estudadas até então.

É comum que este tipo de estrutura seja chamado de registro ou de estrutura em algumas linguagens de programação e podem gerar alguma confusão em função de outras linguagens ou aplicações da área de tecnologia da informação utilizarem estes dois termos com finalidades distintas.

De qualquer maneira, para convencionar os estudos neste material será utilizado o termo registro em função do uso da palavra estrutura como um termo genérico para estruturas de dados.

CONECTE-SE



Em certos momentos dos estudos, os conceitos de algoritmos se misturam com conceitos de outras áreas da TI, e um destes momentos ocorre nesta aula onde a base para a teoria de bancos de dados se mistura com a de estruturas de dados homogêneas e heterogêneas.

Acesse o link: [Disponível aqui](#)

Registros

```
TIPO CADASTRO = REGISTRO
    INTEIRO : CODIGO, IDADE;
    CARACTERE : NOME, ENDEREÇO, CIDADE, ESTADO
FIMREGISTRO;
CADASTRO : DADOS;
```

Imagem 38: Exemplo de estrutura de decisão em estrutura de repetição. | Fonte: O autor.

No exemplo trazido na imagem 38 é declarada uma estrutura de dados do tipo registro com o nome de “CADASTRO” após o uso da palavra reservada “TIPO” e antes da sequência “= REGISTRO” que são elementos padrões da sintaxe e que sempre serão utilizados desta forma na escrita de algoritmos.

Em seguida, no exemplo são declarados campos para o registro que seguem os padrões de sintaxe das variáveis comuns, indicando primeiramente o tipo de dado aceito, seguido do símbolo “:”. Depois, um ou mais nomes de campos podem ser inseridos se forem todos do mesmo tipo, separados por vírgulas e finalizados pelo símbolo padrão “;” como em praticamente todas as demais instruções.

Após a declaração de todos os campos desejados com a indicação dos respectivos tipos, ao final, utiliza-se a palavra reservada “FIMREGISTRO” para encerrar a declaração da estrutura de dados do tipo registro, seguido do símbolo “;”.

Assim, com a declaração do registro, o que se tem na verdade, é um novo tipo de dados definido pelo desenvolvedor, e que pode ser utilizado na declaração de variáveis simples.

Esse recurso permite que uma variável declarada com um tipo de registro de dados possa ser então preenchida com dados e manipulada normalmente por um algoritmo no qual esteja declarada. A imagem 39 traz a forma como se pode utilizar a variável declarada no exemplo da imagem 38.

```
LEIA (DADOS.CODIGO);  
LEIA (DADOS.NOME);  
LEIA (DADOS.IDADE);  
LEIA (DADOS.ENDEREÇO);  
LEIA (DADOS.CIDADE);  
LEIA (DADOS.ESTADO);  
ESCREVA ("REGISTRO NÚMERO: ", DADOS.CODIGO);  
ESCREVA ("NOME: ", DADOS.NOME);  
ESCREVA ("IDADE: ", DADOS.IDADE);  
ESCREVA ("ENDEREÇO: ", DADOS.ENDEREÇO);  
ESCREVA ("LOCAL: ", DADOS.NOME, "- ", DADOS.ESTADO);
```

Imagem 39: Exemplo de uso de variável declarada como registro. | Fonte: O autor.

Neste exemplo da imagem 39, complementar ao da imagem 38, temos o uso da estrutura de dados do tipo registro definida e posteriormente utilizada na declaração de uma variável "DADOS".

Neste trecho de algoritmo são utilizados comandos de entrada e saída de dados para que seja possível a entrada de dados para os campos do registro com o comando LEIA, e depois sejam reexibidos de forma formatada com informações complementares ao usuário com o comando ESCREVA.

PARA GABARITAR



Compreender a forma de uso para estruturas heterogêneas é bastante importante, visto que são estruturas desiguais, e assim, aceitam dados de tipos diferentes, podendo aumentar rapidamente a complexidade da elaboração de algoritmos utilizando estes tipos de estruturas de dados, ainda mais se utilizadas em conjunto com vetores para a criação de listas de registros a serem manipulados em tempos de execução.

Fonte: [Disponível aqui](#)



Sub-Rotinas

Sub-Rotinas

Existem os chamados paradigmas de programação que definem diferentes formas de se programar, e até este momento nos estudos foi utilizado um paradigma chamado imperativo em que o algoritmo inicia em um bloco de instruções que se seguem até o final contendo todas as funcionalidades do mesmo.

Nesta aula será trabalhada uma nova forma de se construir algoritmos com base em outro paradigma. O paradigma estruturado permite que partes de um algoritmo sejam inseridos em sub-rotinas que possam ser acessadas pela parte principal deste algoritmo a qualquer momento, reduzindo a quantidade de linhas necessárias em algoritmos mais complexos e oferecendo uma maior possibilidade de reuso de código no desenvolvimento de outros algoritmos, posteriormente.

Uma sub-rotina representa um nome que se refere a todo um trecho de código com certa independência de funcionalidades em relação ao restante de um algoritmo e se torna algo como se um algoritmo fosse aninhado dentro de outro, como foi estudado com estruturas de controle em aulas anteriores.

Dentro de uma sub-rotina é possível utilizar praticamente todos os conceitos estudados até este ponto e criar algoritmos cada vez mais eficientes e mais bem organizados e fáceis de interpretar e modificar quando necessário.

Forbellone (2005) utiliza o termo módulo para todo tipo de sub-rotina, mas existem dois tipos de sub-rotinas para diversos autores e linguagens de programação, e em função desta possibilidade de separação entre tipos de sub-rotinas, será usada a definição citada por Manzano (1997) que organiza as sub-rotinas em procedimentos e funções.



Escopo de Estruturas de Dados

O chamado escopo de estruturas de dados é um importante conceito a ser trabalhado neste ponto dos estudos, pois afeta diretamente o uso e funcionalidades de sub-rotinas, podendo tornar um pouco mais complexa a compreensão das sub-rotinas e como utilizá-las corretamente.

Escopo se refere a qual ponto em um algoritmo uma estrutura de dados pode ser utilizada diretamente, e quando é necessário que referências a estruturas de dados sejam feitas para que se possa acessar seus dados e manipulá-los.

Para esta aula serão desconsiderados os meios para se transmitir dados entre sub-rotinas, pois é conteúdo para a aula seguinte, mas é importante saber que para se utilizar sub-rotinas quaisquer, não é obrigatória a declaração de estruturas de dados todas globais que possam ser acessadas de quaisquer pontos de um algoritmo para solucionar todos os problemas.

O ideal é que se complemente os conteúdos trabalhados nesta aula com os da aula seguinte para que o conjunto desses conceitos se torne uma ferramenta muito e ciente de desenvolvimento de algoritmos e software em diversas linguagens de programação.

Atuando apenas com as estruturas de dados declaradas de acordo com seus respectivos escopos apenas, é preciso um planejamento para a construção de algoritmos de forma que os dados possam ser utilizados de maneira adequada pelo algoritmo, o uso de estruturas de dados em algoritmos serão tratados agora da seguinte maneira:

- Estruturas de dados declaradas na parte principal do algoritmo são tratadas como globais, ou seja, acessíveis de qualquer ponto do algoritmo.
- Estruturas de dados locais declaradas dentro de sub-rotinas podem apenas ser acessadas por instruções contidas na sub-rotina onde foi declarada, ficando assim, inacessível em outros pontos do algoritmo.

PARA GABARITAR



O uso de variáveis e outras estruturas internamente a sub-rotinas é bastante comum, mas é preciso lembrar que qualquer estrutura de dados declarada dentro de uma sub-rotina se torna local a ela.

Assim, é preciso estar ciente de que dados gerados em sub-rotinas e armazenados em estruturas locais são todos perdidos quando se sai de uma sub-rotina e retorna-se ao trecho do algoritmo que chamou a sub-rotina.

Procedimentos

O princípio dos dois tipos de sub-rotina é o mesmo, mas os procedimentos em si são mais simples de se compreender e de utilizar, pois possuem menos detalhes a serem implementados.

Os procedimentos são sub-rotinas que executam processamento de dados com base em dados obtidos de estruturas de dados que sejam de escopo global, pois estruturas de dados que sejam de fora do procedimento e sejam considerados de escopo local não podem ser acessados pelo procedimento, a menos que sejam passados para o procedimento como será visto na aula seguinte.

Os procedimentos são sub-rotinas que executam processamento de dados de forma interna e podem atuar no restante do algoritmo fora da sub-rotina apenas se estruturas de dados globais forem manipuladas, e estruturas de dados contidas em outros procedimentos não podem ser acessados por terem escopo local.

Observe na imagem 40 um exemplo de procedimento que pode ser inserido em qualquer algoritmo, desde que suas funcionalidades e estruturas de dados necessárias para que possa ser executado sejam respeitadas.

```
PROCEDIMENTO CALCULA ()  
DECLARE  
    INTEIRO : SOMA;  
    SOMA <- V1 + V2 + V3;  
    ESCREVA ("RESULTADO = ", SOMA);  
FIM;
```

Imagem 40: Exemplo de procedimento. | Fonte: O autor.

Neste exemplo da imagem 40, é declarado um procedimento por meio da palavra reservada "PROCEDIMENTO" seguido de um nome e parênteses. Esta sintaxe para procedimentos foi padronizada com base na sintaxe de Manzano (1997), mas ajustada para se aproximar também da sintaxe trazida por Ascencio (2011).

Com esses ajustes nas sintaxes trazidas por diferentes autores, obteve-se esta sintaxe que se assemelha a do desenvolvimento para o algoritmo padrão estudado nas aulas anteriores, e assim, é mais fácil compreender a ideia de que uma sub-rotina pode ser vista como se fosse um pequeno algoritmo que pode ser chamado por outro algoritmo num mesmo arquivo.

Neste exemplo trazido na imagem 40, é declarado um procedimento chamado "CALCULA" que possui declarada, logo em seguida, uma variável "SOMA" definida como número inteiro. Depois, esta variável "SOMA" recebe os valores contidos nas variáveis "V1", "V2", e "V3" para obter um resultado atribuído à variável "SOMA".

**ABORDAGEM
PRÁTICA**

As três variáveis utilizadas no cálculo não foram declaradas no procedimento "CALCULA" como a variável para o resultado, e assim, supõe-se que, ou as variáveis foram declaradas no algoritmo principal e são de escopo global, ou foram esquecidas as declarações destas três variáveis. Em seguida, após a obtenção do resultado para o cálculo, é exibido ao usuário o valor obtido em uma mensagem, para depois, o procedimento ser encerrado, e o controle do fluxo de execução do algoritmo ser retomado pelo algoritmo principal.

Funções

As funções possuem uma característica importante que as diferem dos procedimentos, pois possuem um tipo de dado associado à própria função e que ao final da execução da mesma, deve então retornar um valor para o trecho do algoritmo que chamou a função.

Em razão desta diferença, a forma como se deve chamar uma função também ocorre de forma diferente, pois os procedimentos, por não retornarem valor nenhum, é chamado simplesmente pelo seu nome no algoritmo, ao passo que uma função, pelo retorno que deve efetuar, deve estar associada a uma atribuição de valor, a uma estrutura de dados ou como parte de uma instrução de saída de dados.

Assim, a partir de uma adaptação da sintaxe trazida por Ascencio (2011), a sintaxe para uso neste material se assemelhará ao da sintaxe dos procedimentos, tendo como diferencial, a definição do tipo e retorno de valor ao final da função. Observe o exemplo da imagem 41.

```
FUNÇÃO CALCULA () DE INTEIRO  
DECLARE  
    INTEIRO : SOMA;  
    SOMA <- V1 + V2 + V3;  
    ESCREVA ("RESULTADO = ", SOMA);  
    RETORNE SOMA;  
FIM;
```

Imagem 41: Exemplo de função. | Fonte: O autor.

Por este exemplo da imagem 41, percebe-se que é uma adaptação do procedimento da imagem 40, de forma a torná-lo uma função, e para isto, a modificação ocorre na palavra reservada “FUNÇÃO” inserida antes do nome da sub-rotina, e do acréscimo da palavra reservada “DE” seguido de um tipo válido de dados para definir o retorno na função.

O processamento interno da função se mantém igual, e ao final, antes do “FIM” da função, utiliza-se a palavra reservada “RETORNE”, seguida do valor a ser retornado ao trecho do algoritmo que chamou a função. Detalhe que o valor a ser retornado pode ser um dado em si, ou uma estrutura de dados para que o valor contido na mesma seja retornado.

Como estamos lidando com algoritmos, e não uma linguagem em si, não há muitas limitações na elaboração de algoritmos na verdade, mas para efeito de estudos, padrões são definidos para simplificar os estudos, mas é possível ampliar os conceitos para que sejam mais abrangentes, e assim, pode-se, por exemplo, utilizar vetores, matrizes ou registros como tipos válidos para funções e seus retornos, tornando a elaboração de algoritmos mais complexa e completa, mesmo que muitas linguagens não possuam essa capacidade de lidar com funções assim.

CONECTE-SE

O uso de funções em programação é muito comum, e a estruturação de código em funções é inclusive foco de uma forma de programação que se baseia na implementação de funções para todas as funcionalidades. O chamado paradigma de programação funcional trabalha com funções como base de toda a programação, assim como nos cálculos matemáticos.

Acesse o link: [Disponível aqui](#)



Parâmetros

Sub-Rotinas: Passagem de Parâmetros

Continuando os estudos iniciados na aula anterior, as sub-rotinas, sejam elas procedimentos ou funções, já se mostraram recursos extremamente úteis para tornar algoritmos mais bem estruturados e mais facilmente reutilizáveis.

O uso das sub-rotinas até então depende do escopo das estruturas de dados para que possam ser funcionais, e dependendo do escopo destas estruturas, poderia inviabilizar o desenvolvimento de parte dos algoritmos de forma estruturada em sub-rotinas.

Um exemplo de caso em que o escopo pode gerar problemas, é no caso de um algoritmo possuir duas ou mais sub-rotinas, além do algoritmo principal, e em algum momento, uma sub-rotina necessitar de dados que estavam armazenados em estruturas de dados contidos em outra sub-rotina, e assim, estas seriam de escopo local.

Além disso, existe ainda o problema de que estruturas de dados locais em uma sub-rotina deixam de ter validade e perdem seus dados quando a sub-rotina for encerrada, e a execução retorna ao ponto de onde foi chamada. Isto gera um problema, pois não é possível o uso de dados processados em estruturas de dados locais de uma sub-rotina em outra a princípio, mas existe uma forma de evitar a perda de dados de estruturas locais com o encerramento de uma sub-rotina.

É possível que dados sejam passados de uma sub-rotina a outra por meio de parâmetros que servem como ligação de dados entre sub-rotinas, ou entre sub-rotinas e o algoritmo principal.

Passagem de Parâmetros por Valor

A passagem de parâmetros pode ocorrer de duas formas, sendo a primeira delas chamada de passagem por valor em que dados são passados diretamente de um ponto do código que chama uma sub-rotina para dentro da mesma, podendo ser utilizado livremente pelos processos da sub-rotina.

A passagem de dados diretamente de uma parte de um algoritmo para dentro de uma sub-rotina representa que um valor específico ou uma cópia do conteúdo contido em uma estrutura de dados é enviado para uma sub-rotina, e isto não afeta o valor original na origem onde foi gerada a chamada da sub-rotina.

Assim, qualquer processamento realizado sobre o valor recebido como parâmetro por uma sub-rotina se mantém apenas durante a execução da sub-rotina, e estas alterações são perdidas, a menos que a sub-rotina seja uma função e o dado processado seja retornado ao ponto do algoritmo que chamou a sub-rotina.

No caso de procedimentos, como não há retorno de dados, parâmetros passados por valor servem apenas como dados simples para auxiliar nos processamentos implementados no procedimento.

No caso de funções, estas podem ser implementadas com o intuito de que os processamentos internos sejam organizados para realizar alterações necessárias em dados recebidos como parâmetros para que os retornos possam obter novos dados relevantes para a continuidade da execução de um algoritmo sem alterar dados contidos em fontes originais usadas para preenchimento dos dados utilizados como argumentos para os parâmetros permitidos pela função.

```
PROCEDIMENTO CALCULA ( INTEIRO : V1, V2, V3)
DECLARE
    INTEIRO : SOMA;
    SOMA <- V1 + V2 + V3;
    ESCREVA ("RESULTADO = ", SOMA);
FIM;
```

Imagem 42: Exemplo de passagem de parâmetro por valor. | Fonte: O autor.

No exemplo trazido na imagem 42, temos uma função para cálculo da soma de três valores. Dentro desta função, alguns detalhes devem ser observados como a declaração de três parâmetros que funcionarão como variáveis a serem utilizadas pela função “CALCULA” do tipo inteiro e chamadas de “V1”, “V2”, “V3”.

Estes parâmetros são a forma mais adequada de passagem de dados entre uma função e algum ponto do algoritmo que possa chamar a execução desta função para a realização de processos necessários.

Assim, estes parâmetros servirão como receptores de dados vindos de fora da sub-rotina, mas é importante observar que neste tipo de passagem de parâmetros por valor, apenas uma cópia dos dados é utilizada como argumento para cada parâmetro, e os dados originais permanecerão inalterados durante a execução da sub-rotina.

PARA GABARITAR



Parâmetros representam a comunicação entre partes de um algoritmo divididas em sub-rotinas. Esta comunicação é muito importante para que o fluxo de dados que transitam entre as sub-rotinas permita que algoritmos possam dividir o processamento de seus dados em partes.

Situações como cálculos a serem realizados podem estar em sub-rotinas diferentes num algoritmo em relação à obtenção dos dados, e com a passagem de parâmetros por valor, estes dados podem ser enviados a uma sub-rotina que efetue os cálculos necessários e exiba resultados, por exemplo.

Passagem de Parâmetros por Referência

Outro tipo de parâmetro que pode ser utilizado em sub-rotinas possui uma forma de utilização diferente e acaba sendo não uma alternativa à passagem de valor, mas sim, utilizado para outro tipo de finalidade. Na passagem de parâmetro por referência, os dados em si não são passados de uma parte do algoritmo para o interior de uma sub-rotina como cópia do valor original.

Nesse tipo de passagem de parâmetros, os dados em si se mantêm nas estruturas de origem no algoritmo chamador, e referências a estas estruturas são passadas para que possam ser diretamente manipuladas pela sub-rotina e todos os processamentos realizados afetam diretamente os dados contidos em estruturas de dados externas à sub-rotina como se fossem estruturas de dados locais.

O uso deste recurso é importante, pois pode auxiliar no sentido de que dados não precisem ser replicados para serem utilizados em sub-rotinas, e ainda economiza recurso de *hardware* com um menor uso de memória e processamento para seu gerenciamento, ponto relevante no desenvolvimento de *software* para dispositivos com menor poder de processamento e memória.

CONECTE-SE



Os parâmetros se referem a dados esperados por sub-rotinas para seu processamento, mas existe um termo chamado argumento que se confunde com o termo parâmetro. Na verdade, parâmetro representa o valor esperado pela sub-rotina e argumento é o dado passado como parâmetro para a sub-rotina.

Acesse o link: [Disponível aqui](#)

```
PROCEDIMENTO CALCULA (INTEIRO : V1, V2, V3, VAR INTEIRO : SOMA)  
    SOMA <- V1 + V2 + V3;  
FIM;
```

Imagem 43: Exemplo de passagem de parâmetro por referência. | Fonte: O autor.

Neste exemplo contido na imagem 43, a proposta da sub-rotina é a mesma do exemplo apresentado na imagem 42 de somar três valores, mas a construção da sub-rotina foi feita de forma diferente, utilizando o conceito da passagem de parâmetros por referência.

A distinção em termos de declaração da sub-rotina é sutil, mas faz muita diferença na prática, pois a adição da palavra reservada “VAR” no parâmetro “SOMA” faz com que este dado que será enviado como argumento pelo trecho do algoritmo que chame esta sub-rotina.

Assim, ao invés de uma cópia do dado contido na variável ser passada como cópia no argumento enviado pelo algoritmo à sub-rotina, uma referência à própria origem do dado é passada como argumento para o parâmetro, e assim, toda modificação que for realizada no valor do parâmetro, na verdade altera o dado original do algoritmo que chamou a sub-rotina.

ABORDAGEM**PRÁTICA**

O uso de sub-rotinas é bastante comum e importante no desenvolvimento de algoritmos pelas suas diversas características. De forma complementar aos estudos e prática de elaboração de algoritmos modulares utilizando sub-rotinas, é essencial que seja conhecido e praticado o uso da passagem de parâmetros.

Cada situação distinta poderia necessitar de parâmetros que fossem ou não utilizados para passagem de dados entre partes de um algoritmo, e se o melhor mecanismo para cada caso é a passagem por valor ou por referência.

Buscar sempre elaborar algoritmos de forma modular é uma excelente prática e permite além da prática dos conceitos estudados nesta unidade, o aprendizado de importante técnica de reduzir problemas maiores em problemas de menor complexidade, facilitando o desenvolvimento de soluções e a maior modularização dos algoritmos de forma natural.



Recursividad

Sub-Rotinas: Recursividade

O uso de sub-rotinas é bastante interessante, e permite que algoritmos sejam estruturados de forma distinta, permitindo melhor compreensão da lógica e maior possibilidade de reuso de código.

As sub-rotinas são utilizadas geralmente para processamentos mais específicos e que possam ser realizados com certa independência do restante do algoritmo, além de serem uma forma importante de implementação de algoritmos, pois uma sub-rotina representa um trecho de algoritmos que pode ser chamada a qualquer instante durante a execução do mesmo, sem que haja necessidade de duplicação deste trecho do algoritmo, reduzindo a quantidade de linhas e de tempo necessário para implementação.

Um recurso um tanto complexo e que será trabalhado nesta aula com alguns exemplos é a execução de sub-rotinas de forma recursiva em que uma sub-rotina em execução pode, se necessária, chamar a si mesma para que sucessivas iterações da execução da sub-rotina sejam possíveis, assim como ocorre em laços de repetição.

Um detalhe bastante pertinente é que nessa comparação feita, pode-se imaginar laços de repetição sendo implementados como sub-rotinas recursivas, em que as execuções das iterações por chamadas à sub-rotina por ela mesma podem ser programadas para ocorrerem como nas condições dos laços de repetição.

Muitas vezes, as sub-rotinas recursivas são utilizadas para a realização de cálculos matemáticos, mas podem ser também utilizadas para outras finalidades diversas sem envolver necessariamente cálculos.

O que reduz muito o uso da recursão é a forma como é elaborada a lógica e o raciocínio em soluções baseadas nesta técnica, pois a forma como se estruturam sub-rotinas recursivas é mais complexo que outras estruturas do desenvolvimento de algoritmos em geral, mas isto é algo que com a prática, se torna mais fácil e pode ser incorporada como uma prática comum no desenvolvimento de algoritmos.

CONECTE-SE

Após a apresentação do conceito de recursividade, é importante que se exercitem exemplos de código, modifique-se cada um para que se obtenham novas alternativas de funções recursivas, e assim, possa ser compreendido este conceito.

Acesse o link: [Disponível aqui](#)

A elaboração de algoritmos que utilizem o recurso é bastante comum, pois, em geral, não é todo o algoritmo que é recursivo, mas apenas uma ou mais sub-rotinas contidas no algoritmo para realização de algum processamento repetitivo que possa ser realizado utilizando recursão.

Existem alguns exemplos bastante comuns de aplicação da recursão, mas em geral, não são implementados desta forma devido ao maior grau de complexidade lógica necessário para sua elaboração, mas para efeito de estudos, podem ser exemplos bastante interessantes. Observe o exemplo de função da imagem 44.

```
FUNÇÃO POTENCIA (INTEIRO BASE, INTEIRO EXPOENTE) DE INTEIRO
  SE (EXPOENTE = 0) ENTÃO
    RETORNE 1;
  SENÃO
    RETORNE (BASE * POTENCIA (BASE, EXPOENTE-1));
  FIMSE;
FIM;
```

Imagem 44: Exemplo de função recursiva. | Fonte: O autor.

Neste exemplo da figura 44, é possível observar que a escrita de uma função recursiva em sua sintaxe é semelhante a qualquer outra função, mas existe um detalhe que indica a existência de uma possível recursão logo num primeiro olhar, quando se percebe que a função, em determinado momento, chama a si mesma.

Analisando com mais detalhes, a funcionalidade da sub-rotina se baseia em calcular, a partir de dois valores recebidos como parâmetros, a operação exponencial matemática, e o resultado seja devolvido ao ponto do algoritmo que chama esta sub-rotina.

Para que o cálculo possa ser realizado, existe uma estrutura de repetição encarregada de finalizar a função ou chamá-la novamente para que uma nova iteração ocorra desta sub-rotina, e a recursão seja realizada.

A lógica da recursão neste exemplo se baseia na forma como o cálculo é realizado para que se possa pensar em como realizar as sucessivas operações até que uma condição indique que não são mais necessárias chamadas da função, e o processo de repetição termine, mas não ocorre apenas o término das iterações, mas todos os retornos das novas chamadas da função.

Ao receber os dados por meio dos parâmetros, é realizada uma verificação em uma estrutura condicional para retornar o valor 1 como resposta no caso de o expoente ser igual a zero, operação definida como padrão pela matemática o resultado 1 para qualquer número elevado a zero.

Caso não seja 1 o valor do expoente, a função chama a si mesma novamente solicitando a multiplicação do valor a base pelo retorno da função, repetindo o mesmo valor para a base, mas reduzindo uma unidade no expoente para que a cada iteração, cada nova chamada da função, o processo se repita.

Com isto, se fossem passados os valores 5 para base, e 3 para expoente, a execução da função ocorreria de forma que 3 seria comparado a zero, e sendo diferentes, a condição alternativa para falso inicia a multiplicação do valor da base pelo resultado da chamada da própria função pela recursividade.

Numa segunda iteração da função, o expoente valeria 2, sendo diferente de zero ainda, e engatilhando nova multiplicação da base na expressão já iniciada na iteração anterior, e passando para a nova ocorrência da chamada da função com base igual, e expoente menos 1.

Na próxima iteração (terceira), tem-se agora o expoente 1, que ainda permanece diferente de zero, e por isto, nova multiplicação é organizada, tendo agora três operações “5 * ” encadeadas.

Novamente a função é chamada com a mesma base, mas agora, com mais uma redução do expoente, seu valor segue como zero para a nova iteração, e nela, como o expoente tem seu valor igual ao da comparação, realmente ocorre o retorno de um valor 1 para ser utilizado na expressão, e ao fim de uma quarta iteração, a expressão se encontra semelhante a “5 * 5 * 5 * 1”.

Realizando o cálculo normalmente, obtém-se um resultados 125 pelas três multiplicações com 5 e uma inicial pelo valor 1, e assim, com os sucessivos retornos, este resultado é obtido e a função encerra seu processamento.

Esse exemplo traz uma demonstração de como se pode aplicar recursividade em algoritmos, mas este mesmo exemplo pode ser escrito em forma de algoritmo imperativo sem a modularização em forma de função ou sem o uso da recursividade. Observe o exemplo da imagem 59.

```
FUNÇÃO POTENCIA (INTEIRO BASE, INTEIRO EXPOENTE) DE INTEIRO  
DECLARE  
  INTEIRO : I, RESULTADO;  
  RESULTADO <- 1;  
  PARA I DE 1 ATÉ EXPOENTE PASSO 1 FAÇA  
    RESULTADO <- RESULTADO * BASE;  
  FIMPARA;  
  FIM;
```

Imagem 44: Exemplo de função recursiva. | Fonte: O autor.

Nesta versão alternativa da imagem 44, a funcionalidade se mantém inalterada, mas a forma como se desenvolve o processamento é totalmente diferente, pois, ao invés de ser utilizado o recurso da função chamar a si mesma pela recursividade, apenas uma estrutura de repetição se encarrega da obtenção do mesmo resultado.

A lógica elaborada para essa versão da solução é mais simples de ser pensada e de implementar inicialmente, mas é preciso observar que alguns detalhes não podem ser ignorados como a necessidade de duas variáveis adicionais locais à função, a correta inicialização da variável “RESULTADO” e a correta elaboração do laço de repetição.

Esse processo de elaboração de estruturas de repetição necessita de prática para que se torne simples seu uso, assim como o uso da técnica de recursividade que necessita de prática para que se esteja familiarizado com a sua lógica.

PARA GABARITAR



O chamado aninhamento ou encadeamento de estruturas de controle é um recurso muito útil e acrescenta muitos recursos no desenvolvimento de algoritmos, mas também traz um aumento na complexidade de algoritmos que pode confundir iniciantes.

É importante praticar o desenvolvimento de pequenos algoritmos contendo estruturas de controle variadas para compreender a mecânica dos aninhamentos e como uma estrutura possui influência sobre a outra, além de compreender melhor os motivos de ser necessário aninhar estruturas e como definir quais tipos são adequados a cada situação.

A ideia de que funções utilizando recursividade possuem características semelhantes às de estruturas de repetição mostram que a recursividade pode ser uma ferramenta muito eficiente para a realização de ações repetitivas.

O mais importante é observar diferentes exemplos, avaliar sua lógica, e a partir do amadurecimento da compreensão do mecanismo da recursividade, construir algoritmos cada vez mais completos, eficientes e com maior complexidade para a solução de problemas também mais complexos.

```
FUNÇÃO TABUADA (INTEIRO VALOR1, INTEIRO VALOR2) DE INTEIRO
DECLARE
  SE (VALOR1 = 0) ENTÃO
    RETORNE 1;
  SENÃO
    ESCREVA (VALOR2, " X ", VALOR1, " = ", VALOR1 * VALOR2);
```



```
RETORNE (TABUADA (VALOR1-1, VALOR2));  
FIMSE;  
FIM;
```

Imagem 45: Exemplo de função recursiva. | Fonte: autor.

Neste outro exemplo trazido na imagem 45, é elaborado um algoritmo para exibir a tabuada de um determinado valor. Como artifícios para o funcionamento da tabuada são utilizados dois parâmetros para a função, mas ambos são utilizados para o mesmo valor referente a qual tabuada deve ser exibida.

O detalhe é que são utilizados esses dois parâmetros para que um possa ser utilizado nas iterações da recursão, e o outro para manter intacto o valor original da tabuada a ser exibida. Isto é feito, pois a cada nova chamada da função recursiva, um dos parâmetros é decrementado em uma unidade para controle das iterações, e o outro permanece o mesmo.

Outro ponto relevante é que a cada iteração da recursão não é apenas realizado um cálculo no momento da chamada da função, mas uma instrução para exibição de conteúdo ao usuário com texto, dados e mais o resultado do cálculo de cada iteração tem o propósito de construir a tabuada sequencialmente.

Importante realçar que a construção da tabuada é realizada sequencialmente, mas em ordem inversa à redução do valor da variável "VALOR1" a cada nova chamada da função, pois o processamento é realizado apenas após a condição de parada ser atendida e as sucessivas recursões irem retornando e realizando a instrução baseada no comando "ESCREVA".



Manipulação de Arquivos

Manipulação de Arquivos de Texto

Os dados manipulados durante a execução de algoritmos são normalmente perdidos ao final do processo, pois são todos alocados em estruturas de dados em memória temporária, a menos que sejam enviados pela *web* ou gravados em unidades físicas, por exemplo, não se podem recuperar dados processados em uma nova execução do mesmo algoritmo ou por outros algoritmos.

Para que dados processados possam estar disponíveis para uso após o encerramento da execução de um algoritmo, é preciso que sejam criados meios para que estes sejam armazenados de forma permanente e não apenas de forma temporária.

O uso de arquivos de texto ou binários é uma forma eficiente de se ter dados disponíveis sempre, permitindo a adição, consulta, edição ou exclusão dos mesmos a qualquer momento pela execução de algoritmos implementados com recursos de uso de arquivos.

Como os arquivos do tipo binário representam recursos muito mais complexos em sua manipulação, serão deixados de lado e o foco da aula será o uso de arquivos de texto para armazenamento de dados com base em caracteres textuais de forma geral.

Um arquivo é uma estrutura controlada pelo sistema operacional, e por isto, o desenvolvimento de algoritmos precisa se atentar aos detalhes dos sistemas operacionais para a implementação de certas funcionalidades, como na forma como se nomeiam arquivos nos diferentes sistemas operacionais e como são manipulados.

Arquivos de texto contêm conteúdo não necessariamente estruturado, e por isto, caso seja necessária a organização do conteúdo a ser mantido em um arquivo, é importante incluir em toda a documentação dos algoritmos, a forma como devem ser organizados os dados em arquivos, caracteres especiais que serão utilizados como separadores ou palavras-chave que serão inseridas junto ao conteúdo para melhor organização, se necessário.

Como esse tipo de arquivo possui tamanho bastante reduzido, geralmente é possível que uma grande quantidade de dados possa ser armazenada, e assim, sua utilidade e finalidades as quais possam ser aplicadas aumentam, fazendo deste recurso um

componente importante a ser considerado na elaboração de algoritmos para aplicações que necessitem de dados gravados.

Assim, é possível imaginar arquivos de texto em que cada linha do mesmo representa um conjunto de dados referentes a uma mesma estrutura definida pelo desenvolvedor, e assim, através do uso de caracteres específicos, pode indicar separações entre dados contidos em cada linha de texto.

Dessa forma, é possível elaborar um arquivo que contenha dados como nome, número de CPF e RG, data de nascimento e outros dados usando símbolos como “#” que não são comumente utilizados para separar os dados em cada linha, formando estruturas semelhantes às utilizadas em sistemas de bancos de dados simples.

Existe um trabalho padrão de manipulação de dados em estruturas de dados que, independentemente de serem ou não gravados com algum tipo de recurso de persistência de dados, servem para inserir dados e manipulá-los conhecido como CRUD (Create, Read, Update, Delete ou criação, consulta, atualização, exclusão de dados).

Estes processos básicos são essenciais para que se possam gerenciar dados de forma minimamente adequada, mas é possível adicionar funcionalidades extras como geração de relatórios, envio de dados pela *web*, conversão de dados, etc.

Para fins de estudo em algoritmos que não geram ainda produtos para mercado e se propõem apenas a introduzir os conceitos fundamentais da programação neste momento dos estudos, fica como proposta para pesquisas e estudos futuros com aplicação em linguagens de programação comerciais.

Para compreender como é estruturado um algoritmo para manipulação de dados em arquivos do tipo texto, observe o exemplo da imagem 46 para compreender o que é agregado ao que já foi estudado anteriormente.

```
TIPO CADASTRO = REGISTRO
    INTEIRO : CODIGO, IDADE;
    CARACTERE : NOME, ENDEREÇO, CIDADE, ESTADO
FIMREGISTRO;
TIPO ARQ = ARQUIVO COMPOSTO DE CADASTRO;
CADASTRO : DADOS;
ARQ : AGENDA
```

Neste exemplo da imagem 46 é possível reconhecer a declaração de uma estrutura de dados heterogênea como já estudado. O registro nomeado de “CADASTRO” possui campos que podem ser preenchidos com dados do tipo texto, podendo, se necessário, ser adicionados a um arquivo de dados.

Uma nova instrução utilizada na manipulação de arquivos inicia com a palavra reservada “TIPO”, seguida de um nome arbitrário “ARQ”, e depois, o símbolo “=” com o complemento informando que a estrutura se refere a uma arquivo de dados baseado na estrutura “CADASTRO” declarada anteriormente.

Depois, uma variável com o nome “DADOS” é declarada com o tipo “CADASTRO” para a manipulação de dados, assim como outra variável “AGENDA” é declarada com base no tipo “ARQ” declarado para manipulação de arquivos.

Importante ressaltar que não é obrigatório o uso de estruturas de dados do tipo registro para armazenamento de dados em arquivos, pois os dados podem ser estruturados em variáveis simples do tipo “CARACTERE” tendo em mente que a organização de mais de um dado por linha do arquivo teria que ser organizada de forma manual na variável.

CONECTE-SE



É importante ter em mente que os arquivos geralmente possuem nome e extensão, sendo que a extensão auxilia na identificação do *software* que utiliza cada arquivo, e assim, é interessante que se conheçam extensões mais comuns para que não sejam utilizadas na criação de arquivos de dados para algoritmos criados.

Acesse o link: [Disponível aqui](#)

Existem palavras reservadas próprias para a manipulação de arquivos de texto segundo Forbellone (2005) que necessitam de alguns parâmetros como a indicação do arquivo a ser manipulado. Observe os comandos na imagem 63 para conhecer como são estruturadas instruções com estas palavras reservadas.

```
ABRA (AGENDA);  
AVANCE (AGENDA);  
GUARDE (AGENDA, DADOS);  
COPIE (AGENDA, DADOS);  
FECHE (AGENDA);  
ELIMINE (AGENDA);
```

Imagem 46: Instruções para manipulação de arquivos. | Fonte: O autor.

Nos exemplos de instruções contidos na imagem 46 temos primeiramente uma instrução utilizada para se abrir arquivos de texto com a palavra reservada “ABRA” que necessita que o arquivo seja informado, e no caso do exemplo, a variável “AGENDA” é responsável por conter o arquivo a ser manipulado na execução do algoritmo.

A instrução composta pela palavra reservada “AVANCE” e pelo parâmetro “AGENDA” fazem com que um índice de navegação pelo arquivo avance até uma próxima linha e permita que os dados desta linha sejam manipulados.

Outra instrução apresentada no exemplo utiliza a palavra reservada “GUARDE” para realizar a ação de armazenar fisicamente os dados contidos na variável “DADOS” na próxima linha do arquivo texto indicado por “AGENDA”.

Na sequência, a instrução definida pela palavra reservada “COPIE” traz do arquivo texto, cópias dos dados contidos para que possam ser inseridos na variável “DADOS” e possam ser utilizados pelo algoritmo, e possam, se necessário, serem novamente gravados em disco.

A instrução utilizando a palavra reservada “FECHE” apenas possui a função de encerrar o uso do arquivo indicado por “AGENDA” para reduzir as chances de os dados serem corrompidos em função do arquivo de se manter aberto sem necessidade.

Por fim, a última instrução do exemplo da imagem 63 traz uma instrução que utiliza a palavra reservada “ELIMINE” para excluir todo o arquivo indicado por “AGENDA” tendo ou não dados inseridos no mesmo, e por isto, deve ser utilizado com máximo cuidado.

PARA GABARITAR



Imaginar aplicações desenvolvidas para manipular estruturas de dados heterogêneas em quantidade não pré-estabelecida é um bom tipo de utilidade para o uso de arquivos de texto.

Estes arquivos podem conter grandes quantidades de linhas, e cada linha pode conter os dados de uma estrutura heterogênea organizados de forma a um separador destacar o início e fim de cada dado na linha.

Para finalizar os estudos, um exemplo completo de algoritmo para inclusão de dados em um arquivo de dados para demonstração do uso de instruções próprias para manipulação de arquivos texto:

```
INÍCIO
DECLARE
  TIPO DADOS = REGISTRO
    CARACTERE : NOME;
    INTEIRO : TELEFONE;
  FIMREGISTRO;
  TIPO ARQUIVO = ARQUIVO COMPOSTO DE DADOS;
  DADOS : REGISTRO;
  ARQUIVO : AGENDA;
  ABRA (AGENDA);
  REPITA
    AVANCE (AGENDA);
  ATÉ FDA (AGENDA);
  LEIA (REGISTRO.NOME);
  LEIA (REGISTRO.TELEFONE);
```

```
GUARDE (AGENDA, REGISTRO);  
FECHE (AGENDA);  
FIM.
```

Imagem 47: Exemplo de algoritmo para manipulação de arquivos. | Fonte: Adaptado de Forbellone (2005).

Neste exemplo de algoritmo da imagem 47, é declarado um registro para armazenamento temporário de dados a serem incluídos posteriormente em arquivo de dados. Para isto, um laço de repetição realiza o avanço até o final do arquivo para que novos dados sejam adicionados após o último registro previamente gravado.

Depois, é realizada a leitura de dados para o registro de forma que os dois campos sejam preenchidos pelo usuário e depois gravados em disco com o comando “GUARDE” e, ao final, o arquivo seja devidamente fechado para garantir a correta gravação dos dados e integridade do arquivo.

ABORDAGEM PRÁTICA



A manipulação de arquivos de texto é bastante útil e pode agregar uma grande quantidade de outros conceitos estudados ao longo do material como estruturas de controle de fluxo de execução, e por isto, uma boa forma de se praticar é elaborar algoritmos que gerem dados de tipos variados e que sejam depois armazenados em disco.

O exemplo da imagem 65 é um exemplo de dados que está guardado em um arquivo indicado pela variável “AGENDA” e que através de uma estrutura de repetição “REPITA” busca dados em sequência do arquivo para uma estrutura de dados homogênea (vetor) declarada com um tipo de dados heterogêneos (registro).

Um detalhe a ser observado é que a estrutura de repetição é controlada pelo avanço nas linhas do arquivo com o comando “AVANCE” e encerra as iterações quando o final do arquivo é atingido com a palavra reservada “FMA” no comando “ATÉ”.

```
TIPO DADOS = VETOR [1..100] DE CADASTRO;  
DADOS : LISTA; INTEIRO : I;  
ABRA (AGENDA);  
REPITA  
  AVANCE (AGENDA);  
  COPIE (AGENDA, LISTA [I]);  
  I <- I + 1;  
ATÉ FMA (AGENDA);
```

Imagem 48: Exemplo para leitura de dados em arquivos. | Fonte: O autor.

Conclusão

Ao término deste material, existe a necessidade de avaliação do que foi estudado de maneira informal, mas buscando verificar se está ocorrendo um real aprendizado, e para se ter esta avaliação dos estudos, é interessante que se busque relembrar os principais temas estudados e sua relevância.

Durante o andamento das aulas, foram vistos, inicialmente, conceitos introdutórios sobre a arte de programar e como ela pode ser desenvolvida pelo aluno. As diferentes formas como se podem gerar soluções para problemas e os diferentes tipos de algoritmos que podem ser elaborados para conter as mesmas funcionalidades. Após algumas aulas de preparação, iniciaram-se os estudos dos tipos de dados e como se podem aplicá-los em estruturas de dados como variáveis para que seja possível armazenar dados para a realização de processamentos em algoritmos.

Na sequência, foram estudados mecanismos para interatividade entre algoritmo e usuário por meio de instruções de entrada e saída de dados para que até este ponto, os assuntos se tornem mais claros e sirvam de base para os próximos conteúdos estudados. Entram, então, as chamadas estruturas de controle de fluxo de execução como estruturas de decisão e, posteriormente, estruturas de repetição, sendo estas, fundamentais na elaboração de grande parte dos algoritmos, podendo inclusive, ser encadeadas umas com as outras de forma a permitir soluções de complexidade maior e capazes de tomadas de decisão automatizadas por parte dos próprios algoritmos.

Depois, são tratados conceitos referentes às estruturas de dados mais complexas, formadas por mais de um elemento, tendo tipos de dados definidos, mas estes podendo ser agrupados em quantidades variadas para atender a diferentes demandas. Essas chamadas estruturas de dados homogêneas servem como repositório de dados em maiores quantidades e são organizadas em forma de vetores no caso de estruturas de dados homogêneas unidimensionais ou tabelas e formas com mais dimensões para matrizes de dados homogêneas com duas ou mais dimensões.

Depois, os dados de tipos diferentes puderam ser agrupados em estruturas heterogêneas chamadas de registros, podendo então organizar dados em forma de estruturas que poderiam ser também definidas como tipos para declaração de variáveis e vetores, por exemplo. Complementado esta parte sobre estruturas de dados, o estudo do uso de arquivos de dados trouxe o acréscimo da possibilidade de gravação de dados simples ou estruturas de dados maiores como registros ou vetores, por exemplo.



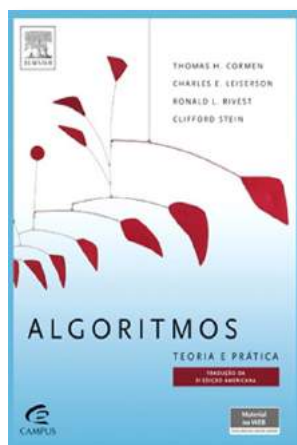
Por fim, a divisão de problemas complexos em problemas menores e mais simples permitiram o estudo das chamadas sub-rotinas que se dividem em procedimentos e funções, tendo a possibilidade de comunicação de dados por meio de parâmetros e podendo chegar ao nível de uma função poder chamar a si mesma repetidas vezes para realização de processamento de maneira recursiva.

Com tantos assuntos, é possível que o estudante não guarde todas estas informações logo a partir da primeira vez que estuda conteúdos como esses citados, mas é fundamental que o aluno pratique o que é trabalhado neste material e agregue mais conhecimentos com o auxílio de bibliografia utilizada, pesquisas em fontes diversas e a prática.

Estudar algoritmos mostra que a criatividade é importante, assim como conhecimento técnico e capacidade lógica. Estas características são importantes, mas a força de vontade é o que faz pessoas estudarem continuamente para aprender e se atualizar.

Bons estudos!

Material Complementar



Livro

Algoritmos : Teoria e Prática

Livro já bastante tradicional nos estudos contendo assuntos bastante completos sobre o desenvolvimento de algoritmos com exemplos e exercícios que pode agregar bastante aos estudos já realizados sobre este tema.

Comentário: Além das obras utilizadas como referência básica neste material, esta obra adicional pode agregar mais conteúdo a quem esteja estudando o desenvolvimento de algoritmos.

Autor: Thomas Cormen

Editora: Campus



Filme

Os Estagiários

Ano: 2013

Sinopse: Dois vendedores veteranos são demitidos e começam a buscar emprego em diversas áreas, mas em um estalo, resolvem se candidatar à vaga do programa de contratação da Google.

Comentário: Comédia leve que mostra o perfil da empresa e dos que nela trabalham de forma descontraída.

[Web](#)

Todo mundo deveria aprender a programar

Vídeo bastante interessante com relatos de alguns das maiores personalidades da área dentre outros que comentam a importância da programação e como ela pode ser importante na educação desde a infância.

[Acesse o link](#)



Referências

ASCENCIO, Ana Fernanda Gomes; CAMPOS, Edilene Aparecida Verenuchi de.

Fundamentos da programação de computadores: algoritmos, pascal, C/C++ (padrão ANSI) e java. 3ª ed., São Paulo: Pearson, 2012.

FORBELLONE, A. L. **Lógica de Programação.** 3ª ed., São Paulo: Pearson, 2005.

MANZANO, José Augusto N. G. e Oliveira. Jayr Figueiredo de. **Estudo Dirigido de Algoritmos.** São Paulo: Érica, 1997.