e-Tec Brasil
Escola Técnica Aberta do Brasil

Lógica de Programação

Victorio Albani de Carvalho

Curso Técnico em Informática




INSTITUTO FEDERAL
ESPÍRITO SANTO

UNIVERSIDADE FEDERAL
DE SANTA CATARINA

Ministério da Educação

e-Tec Brasil
Escola Técnica Aberta do Brasil

Lógica de Programação

Victorio Albani de Carvalho

Colatina - ES
2010

Presidência da República Federativa do Brasil

Ministério da Educação

Secretaria de Educação a Distância

Este Caderno foi elaborado em parceria entre o Instituto Federal de Educação, Ciência e Tecnologia do Espírito Santo e a Universidade Federal de Santa Catarina para o Sistema Escola Técnica Aberta do Brasil – e-Tec Brasil.

Equipe de Elaboração
Instituto Federal do Espírito Santo – IFES

Coordenação do Curso
Joao Henrique Caminhas Ferreira/IFES

Professor-autor
Victorio Albani de Carvalho/IFES

Comissão de Acompanhamento e Validação
Universidade Federal de Santa Catarina – UFSC

Coordenação Institucional
Araci Hack Catapan/UFSC

Coordenação do Projeto
Silvia Modesto Nassar/UFSC

Coordenação de Design Instrucional
Beatriz Helena Dal Molin/UNIOESTE e UFSC

Coordenação de Design Gráfico
Carlos Antônio Ramirez Righi/UFSC

Design Instrucional
Antonio Jonas Pinotti/IFES

Web Master
Rafaela Lunardi Comarella/UFSC

Web Design
CEAD/IFES

Diagramação
Andre Rodrigues/UFSC
Andréia Takeuchi/UFSC
Caroline Ferreira da Silva/UFSC
Guilherme Ataide Costa/UFSC
Juliana Tonietto/UFSC

Revisão
Maria Isolina de Castro Soares/IFES

Projeto Gráfico
e-Tec/MEC

C331L **Carvalho, Victorio Albani de**

Lógica de programação : Curso Técnico em Informática / Victorio Albani de Carvalho. – Colatina: CEAD / Ifes, 2010.
104 p. : il.

1. C (Linguagem de programação de computador). 2. Algoritmos de computador. 3. Material didático. I. Instituto Federal do Espírito Santo. II. Título.

CDD: 005.369

Apresentação e-Tec Brasil

Prezado estudante,

Bem-vindo ao e-Tec Brasil!

Você faz parte de uma rede nacional pública de ensino, a Escola Técnica Aberta do Brasil, instituída pelo Decreto nº 6.301, de 12 de dezembro 2007, com o objetivo de democratizar o acesso ao ensino técnico público, na modalidade a distância. O programa é resultado de uma parceria entre o Ministério da Educação, por meio das Secretarias de Educação a Distancia (SEED) e de Educação Profissional e Tecnológica (SETEC), as universidades e escolas técnicas estaduais e federais.

A educação a distância no nosso país, de dimensões continentais e grande diversidade regional e cultural, longe de distanciar, aproxima as pessoas ao garantir acesso à educação de qualidade, e promover o fortalecimento da formação de jovens moradores de regiões distantes, geograficamente ou economicamente, dos grandes centros.

O e-Tec Brasil leva os cursos técnicos a locais distantes das instituições de ensino e para a periferia das grandes cidades, incentivando os jovens a concluir o ensino médio. Os cursos são ofertados pelas instituições públicas de ensino e o atendimento ao estudante é realizado em escolas-polo integrantes das redes públicas municipais e estaduais.

O Ministério da Educação, as instituições públicas de ensino técnico, seus servidores técnicos e professores acreditam que uma educação profissional qualificada – integradora do ensino médio e educação técnica, – é capaz de promover o cidadão com capacidades para produzir, mas também com autonomia diante das diferentes dimensões da realidade: cultural, social, familiar, esportiva, política e ética.

Nós acreditamos em você!

Desejamos sucesso na sua formação profissional!

Ministério da Educação
Janeiro de 2010

Nosso contato
etecbrasil@mec.gov.br

Indicação de ícones

Os ícones são elementos gráficos utilizados para ampliar as formas de linguagem e facilitar a organização e a leitura hipertextual.

Atenção: indica pontos de maior relevância no texto.

Saiba mais: oferece novas informações que enriquecem o assunto ou "curiosidades" e notícias recentes relacionadas ao tema estudado.

Glossário: indica a definição de um termo, palavra ou expressão utilizada no texto.

Mídias integradas: sempre que se desejar que os estudantes desenvolvam atividades empregando diferentes mídias: vídeos, filmes, jornais, ambiente AVEA e outras.

Atividades de aprendizagem: apresenta atividades em diferentes níveis de aprendizagem para que o estudante possa realizá-las e conferir o seu domínio do tema estudado.

Sumário

Palavra do professor-autor

Caro Aluno,

Você talvez se pergunte por que estudar programação logo no primeiro módulo do curso. Há alguns anos, na área de informática, você poderia optar por três a quatro especializações apenas; hoje estas possibilidades se multiplicaram, pelo fato de a informática permear praticamente todas as atividades da sociedade moderna. Entretanto, dentre estes caminhos possíveis, a programação ainda apresenta uma das maiores demandas de técnicos. Além disto, o estudo de programação desenvolve um tipo de raciocínio (sequencial e lógico) base para resolver problemas de várias outras ramificações da informática, além de ajudar em outras disciplinas do curso. A formação deste tipo de raciocínio, necessário para você prosseguir com sucesso na programação, é o principal objetivo desta disciplina.

Para desenvolver um programa de computador é necessário analisar e entender o problema, inventar ou escolher uma solução, escrever um algoritmo, testá-lo e depurá-lo até que esteja completamente correto. Isto exige tempo, persistência e disciplina. As etapas mais importantes do trabalho são exatamente as primeiras, onde se cria a solução; por isto os maiores amigos do programador são papel e lápis e não o computador.

Se você não tem experiência com computadores, ainda assim você poderá estudar a disciplina e concluí-la com bom aproveitamento. Esta afirmativa é feita baseada em nossa experiência com vários alunos de cursos presenciais, que chegam sem nenhuma experiência, enfrentando algumas dificuldades no começo, mas que conseguem se superar e se tornam bons programadores. Como o aluno a distância é especial, acreditamos que também conseguirá superar as dificuldades iniciais. Se for preciso, peça a ajuda da equipe de apoio e também o auxílio de seus colegas.

Lembre-se: a melhor forma de aprender é praticando! Esta frase é ainda mais verdadeira para a programação. Assim, organize seu tempo, dedique-se ao curso e aproveite: você deverá divertir-se durante o processo de aprendizado.

Um abraço!
Prof. Victorio Albani de Carvalho

Apresentação da disciplina

A disciplina de Lógica de Programação será seu primeiro passo no aprendizado da programação de computadores. Assim, ela é de fundamental importância para o curso, pois é nela que você aprenderá os fundamentos básicos que servirão de alicerce para todas as demais disciplinas da área de programação.

Nesta disciplina você aprenderá os principais conceitos de programação, como variáveis, operadores lógicos e aritméticos, estruturas de laço e estruturas de decisão e terá a oportunidade de aplicar esses conceitos na prática, criando algoritmos e programas em linguagem C. Dessa forma, ao final desta disciplina você estará capacitado a criar algoritmos completos e funcionais utilizando a linguagem de programação C.

A formação do raciocínio, necessário para você prosseguir com sucesso na programação, é o principal objetivo desta disciplina. Por isso, nas primeiras aulas, você estudará o Portugol, uma sintaxe formal para escrever programas, mas que não é uma linguagem de programação (no sentido restrito de ser compilada e interpretada). Na sequência do curso você estudará também a linguagem C.

Por que escolhemos a linguagem C para você iniciar com a programação? Porque ela é a base da sintaxe de diversas outras linguagens de programação, inclusive para Internet (Java, PHP, Java Script...), com as quais você lidará durante o curso e depois, ao longo de sua vida profissional.

Lembre-se de que, apesar de se tratar de um curso a distância, você não está sozinho nesta jornada. Qualquer dúvida que tenha poderá acionar o tutor a distância e, sempre que necessário, poderá solicitar que o tutor entre em contato com o professor. Além disso, temos à nossa disposição um ambiente virtual cheio de recursos para nos auxiliar neste processo.

Bons estudos e sucesso!

Projeto instrucional

Disciplina: Lógica de Programação (carga horária: 90 horas)**.**

Ementa: Lógica de programação. Algoritmos. Estruturas de controle.

AULA	OBJETIVOS DE APRENDIZAGEM	MATERIAIS	CARGA HORÁRIA (horas)
1. Introdução à Lógica de Programação	Compreender os conceitos de lógica de programação e de algoritmos. Conhecer os primeiros exemplos de algoritmos e algumas técnicas para construção de algoritmos. Entender os conceitos de estrutura sequencial, estrutura de seleção e estruturas de repetição no contexto de algoritmos.	Caderno impresso. Ambiente Virtual de Ensino-Aprendizagem (Moodle).	10
2. Conceitos básicos para a construção de algoritmos para computadores	Iniciar-se na utilização de uma linguagem formal para a construção de algoritmos para computadores. Conhecer os conceitos de variável e tipos de dados. Compreender como funciona a alocação de memória em computadores e a declaração de variáveis e constantes em Portugol. Conhecer os conceitos e necessidades de comandos de atribuição e de entrada e saída de dados além dos operadores aritméticos e sua ordem de precedência.	Caderno impresso. Ambiente Virtual de Ensino-Aprendizagem (Moodle).	10
3. Expressões lógicas e estruturas de decisão	Conhecer os operadores relacionais e os operadores lógicos. Entender a Tabela-verdade dos operadores lógicos. Compreender a ordem de precedência entre operadores. Conhecer a formalização de uma estrutura de decisão em Portugol.	Caderno impresso. Ambiente Virtual de Ensino-Aprendizagem (Moodle).	10
4. Estruturas de repetição	Aplicar o conceito de estruturas de repetição em Portugol. Conhecer dois tipos de estruturas de repetição e avaliar quando utilizá-las.	Caderno impresso. Ambiente Virtual de Ensino-Aprendizagem (Moodle).	10

continua

AULA	OBJETIVOS DE APRENDIZAGEM	MATERIAIS	CARGA HORÁRIA (horas)
5. Introdução à linguagem C	Conhecer os conceitos de linguagem de programação, compilador e ambiente de desenvolvimento. Conhecer a ferramenta de programação a ser utilizada. Visualizar a transformação de um algoritmo simples de Portugol para C. Compreender como são declaradas variáveis e constantes e como fazer atribuições em linguagem C. Conhecer os comandos de saída de dados (printf) e de entrada de dados (scanf). Aprender como escrever comentários em linguagem C. Entender como utilizar expressões aritméticas em C.	Caderno impresso. Ambiente Virtual de Ensino-Aprendizagem (Moodle). Ambiente de desenvolvimento DevC++ instalado nos computadores do Polo.	10
6. Estruturas de decisão em linguagem C	Conhecer os operadores lógicos e relacionais da linguagem C. Conhecer as estruturas de decisão da linguagem C: if...else e switch. Entender quando utilizar cada uma das estruturas de decisão da linguagem C.	Caderno impresso. Ambiente Virtual de Ensino-Aprendizagem (Moodle). Ambiente de desenvolvimento DevC++ instalado nos computadores do Polo.	10
7. Estrutura de repetição for	Conhecer as estruturas de repetição for. Entender quando utilizar estruturas de repetição da linguagem C.	Caderno impresso. Ambiente Virtual de Ensino-Aprendizagem (Moodle). Ambiente de desenvolvimento DevC++ instalado nos computadores do Polo.	10
8. Estrutura de repetição while, do...while	Conhecer as estruturas de repetição while, do...while. Entender quando utilizar diferentes estruturas de repetição da linguagem C.	Caderno impresso. Ambiente Virtual de Ensino-Aprendizagem (Moodle). Ambiente de desenvolvimento DevC++ instalado nos computadores do Polo.	10
9. Vetores	Conhecer o conceito de vetor. Aprender como utilizar vetores em linguagem C. Aprender a utilizar vetores de caracteres. Construir programas em linguagem C utilizando vetores.	Caderno impresso. Ambiente Virtual de Ensino-Aprendizagem (Moodle). Ambiente de desenvolvimento DevC++ instalado nos computadores do Polo.	10
			conclusão

Aula 1 – Introdução à Lógica de Programação

Objetivos

Compreender os conceitos de lógica de programação e de algoritmos.

Conhecer os primeiros exemplos de algoritmos.

Entender os conceitos de estrutura sequencial, estrutura de decisão e estrutura de repetição no contexto de algoritmos.

1.1 Conceitos básicos

Nesta disciplina, iniciaremos nossos estudos sobre **Lógica** de Programação. Mas, antes de começarmos, seria útil uma reflexão sobre o significado da palavra "Lógica". Assim, o que é Lógica?

A-Z

Lógica
Pode ser vista como a arte de pensar corretamente. A lógica visa a colocar ordem no pensamento (FARRER, 1999).

Utilizamos a lógica de forma natural em nosso dia a dia. Por exemplo:

a) Sei que o livro está no armário.

Sei que o armário está fechado.

Logo, concluo que tenho de abrir o armário para pegar o livro.

b) Sei que sou mais velho que João.

Sei que João é mais velho que José.

Então, concluo que eu sou mais velho que José.

Atividade 1.1 – Sejam os seguintes fatos:

- Todos os filhos de João são mais altos do que Maria.
- Antônio é filho de João.

Então, o que podemos concluir logicamente?

Atividade 1.2 – Considere os fatos abaixo:

- José é aluno do IFES.
- Para ser aprovado, um aluno do IFES precisa obter nota maior ou igual a 60 e comparecer a mais de 75% das aulas.
- José compareceu a todas as aulas e obteve nota igual a 80.

Então, o que podemos concluir?

1.2 Construção de algoritmos

A lógica de programação é essencial para pessoas que desejam trabalhar com desenvolvimento de programas para computadores. Lógica de programação pode ser definida como um conjunto de técnicas para encadear pensamentos a fim de atingir determinado objetivo.

O objetivo fundamental de toda programação é construir **algoritmos**. Mas, afinal, o que é um algoritmo?

Algoritmo
Formalmente é uma sequência finita de passos que levam à execução de uma tarefa. Podemos pensar em algoritmo como uma receita, uma sequência de instruções que dão cabo de uma meta específica (FORBELLONE et al., 2005).

Em outras palavras, quando criamos um algoritmo, apenas apontamos uma sequência de atividades que levam à solução de um problema. Até mesmo as soluções para os problemas cotidianos mais simples podem ser descritas por sequências lógicas de atividades, ou seja, por algoritmos:

Problema: Trocar uma lâmpada.

Sequência de Passos para Solução:

1. Pegue uma escada;
2. Posicione a escada embaixo da lâmpada;
3. Pegue uma lâmpada nova;
4. Suba na escada;
5. Retire a lâmpada velha;
6. Coloque a lâmpada nova.

Esta solução é **apenas uma** das muitas soluções possíveis para o problema apresentado. Assim, ao criarmos um algoritmo, indicamos uma dentre várias possíveis sequências de passos para solucionar o problema.

Por exemplo, o problema acima poderia ser resolvido mesmo se alterássemos a sequência de passos para:

1. Pegue uma lâmpada nova;
2. Pegue uma escada;
3. Posicione a escada embaixo da lâmpada;
4. Suba na escada;
5. Retire a lâmpada velha;
6. Coloque a lâmpada nova.

Atividade 1.3 – Escreva um algoritmo (sequência de passos) para trocar um pneu de um carro.

Atividade 1.4 – Descreva um algoritmo que defina como fazer um bolo.

Atividade 1.5 – Descreva um algoritmo que defina como preparar um ovo frito.

1.2.1 Algoritmos com estruturas de decisão

Os algoritmos que construímos até agora apresentam uma sequência de passos que devem ser seguidos para atingir um objetivo bem definido. Note que todos os passos dos algoritmos devem ser executados a fim de que o objetivo seja alcançado.

Porém, há algoritmos nos quais a execução de alguns passos pode depender de **decisões** a serem tomadas. Dessa forma, algum fato indicará se um ou mais passos do algoritmo serão executados ou não.

A-Z

Testes
Determinam quais ações serão executadas; são chamados de estruturas de seleção ou estruturas de decisão.

Por exemplo, o nosso primeiro algoritmo define uma sequência de passos para trocar uma lâmpada. Em momento algum perguntamos se a lâmpada está queimada. Simplesmente trocamos a lâmpada sem fazer qualquer teste. Para resolver esse problema, podemos acrescentar ao nosso algoritmo um **teste** que verifique se a lâmpada deve ser trocada:

1. Ligue o interruptor
2. Se a lâmpada não acender:
    - **2.1.** Pegue uma escada;
    - **2.2.** Posicione a escada embaixo da lâmpada;
    - **2.3.** Pegue uma lâmpada nova;
    - **2.4.** Suba na escada;
    - **2.5.** Retire a lâmpada velha;
    - **2.6.** Coloque a lâmpada nova.

Agora, estamos ligando os passos de efetuar a troca da lâmpada a uma condição. Assim, só executamos os passos definidos de 2.1 a 2.6 caso a condição definida do passo 2 seja verdadeira, ou seja, caso a lâmpada não acenda.

1.2.2 Algoritmos com estruturas de repetição

Note que, apesar de nosso novo algoritmo estar verificando a necessidade de trocar a lâmpada antes de fazê-lo, em momento algum verificamos se a lâmpada nova que foi instalada funciona. Assim, vamos tentar alterar o nosso algoritmo a fim de garantir que ao fim de sua execução tenhamos uma lâmpada funcionando. Para isso, vamos incluir um novo teste em seu final:

1. Ligue o interruptor

2. Se a lâmpada não acender:

2.1. Pegue uma escada;

2.2. Posicione a escada embaixo da lâmpada;

2.3. Pegue uma lâmpada nova;

2.4. Suba na escada;

2.5. Retire a lâmpada velha;

2.6. Coloque a lâmpada nova;

2.7. Se a lâmpada não acender:

2.7.1. Retire a lâmpada;

2.7.2. Coloque uma outra lâmpada;

2.7.3. Se a lâmpada ainda não acender:

2.7.3.1. Retire a lâmpada;

2.7.3.2. Coloque uma outra lâmpada;

(Até quando ficaremos nesses testes???)

Pelo nosso novo algoritmo, caso a nova lâmpada não acenda, devemos trocá-la novamente e repetir esse procedimento indefinidamente até que uma lâmpada funcione. Note que não sabemos quantas vezes teremos de repetir o teste até acharmos uma lâmpada que funcione.

Em casos como esse, devemos utilizar estruturas de repetição. Essas estruturas definem um fluxo de ações que se repete enquanto uma determinada situação acontece.

Dessa forma, substituímos nossa sequência indefinida de estruturas de decisão por uma estrutura de repetição:

1. Ligue o interruptor;

2. Se a lâmpada não acender:

2.1. Pegue uma escada;

2.2. Posicione a escada embaixo da lâmpada;

2.3. Pegue uma lâmpada nova;

2.4. Suba na escada;

2.5. Retire a lâmpada velha;

2.6. Coloque a lâmpada nova.

2.7. Enquanto a lâmpada não acender:

2.7.1. Retire a lâmpada;

2.7.2. Coloque uma outra lâmpada.

Assim, neste novo algoritmo, enquanto a condição definida na linha 2.7 for verdadeira (ou seja, enquanto a lâmpada não acender), as ações definidas em 2.7.1 e 2.7.2 serão repetidas. Abstraímos os fatos de ter de descer da escada para pegar uma lâmpada nova e subir novamente.

Resumo

Nesta aula você conheceu o conceito de algoritmo e começou a desenvolver sua lógica de programação a partir de exemplos de algoritmos presentes em nosso cotidiano. Este é apenas o início dessa grande viagem pelo mundo da programação!

Atividades de aprendizagem

1. Elabore um algoritmo que indique como fazer uma prova. Faça o algoritmo pensando que o aluno não deve deixar questões em branco; assim, deve continuar fazendo a prova enquanto existir questão em branco e o tempo de prova não tiver acabado. Além disso, o aluno só deve resolver uma questão se souber resolvê-la, senão pula para a próxima.

2. Suponha que você tenha uma caixa cheia de bolas. Nessa caixa existem bolas azuis e bolas vermelhas. Além disso, você tem também duas caixas vazias. Vamos chamar a caixa que contém as bolas de "caixa 1" e as duas caixas vazias de "caixa 2" e "caixa 3". Neste contexto, escreva um algoritmo que defina como tirar todas as bolas da "caixa 1", colocando as bolas azuis na "caixa 2" e as bolas vermelhas na "caixa 3".

3. José trabalha no departamento de recursos humanos de uma empresa. A empresa de José definiu que os salários dos empregados serão aumentados seguindo a seguinte regra: caso o salário seja menor que R$ 1.000,00, o aumento será de 10%; caso contrário, será de 8%. José recebeu uma lista contendo os nomes e salários de todos os funcionários da empresa e foi solicitado que calculasse o novo salário desses funcionários. Assim, escreva um algoritmo para que José calcule corretamente os novos salários.

4. Desafio de lógica: Três missionários e três canibais encontram-se na margem esquerda de um rio. Nessa margem também existe um bote que pode transportar uma ou duas pessoas. As seis pessoas pretendem todas passar para a margem direita (usando o bote). No entanto, os missionários têm de arranjar um plano para consegui-lo de modo que, em nenhuma circunstância, existam missionários numa margem em minoria

relativamente aos canibais, pois têm receio do que lhes possa acontecer. Quando o bote chega à margem, os elementos do bote são contados como estando na margem. Assim, se houver apenas um canibal em uma margem, não podemos enviar a essa margem o bote com um canibal e um missionário, pois, ao chegar à outra margem, serão dois canibais contra um missionário. Lembre-se de que para o bote ir de uma margem a outra é necessário que alguém esteja remando, ou seja, o bote nunca atravessa vazio. Faça um algoritmo que exiba, passo a passo, como efetuar esta travessia de forma segura.

Aula 2 – Conceitos básicos para a construção de algoritmos para computadores

Objetivos

Entender a necessidade de se utilizar uma linguagem formal para construir algoritmos a serem interpretados por computadores.

Compreender os conceitos de variáveis, constantes e tipos de dados.

Conhecer os conceitos de comandos de atribuição, entrada e saída de dados.

Conhecer os operadores aritméticos e a ordem de precedência.

Construir nossos primeiros algoritmos em Portugol.

2.1 Formalizando a escrita de algoritmos

Para que os computadores sejam capazes de interpretar os algoritmos que desenvolvemos, precisamos transformar a sequência de passos que escrevemos em linguagem natural para uma linguagem que possa ser "entendida" pelo computador. Essas linguagens são chamadas de **linguagens de programação**. Existem diversas linguagens de programação em uso atualmente. Em nosso curso, a partir da aula 5, utilizaremos a linguagem C.

Nesta aula, a fim de facilitar o aprendizado dos principais conceitos de programação, utilizaremos o Portugol. O Portugol é uma linguagem que une o formalismo das linguagens de programação à facilidade de compreensão da linguagem natural.

Para entender a construção de algoritmos nessas linguagens, vamos iniciar estudando alguns conceitos básicos como variáveis e constantes.

2.2 Variáveis

O primeiro passo para que um programa seja executado em um computador é o carregamento desse programa para a memória. A **memória** é utilizada

A-Z

Memória
Meio físico para armazenar dados temporariamente ou permanentemente (TANENBAUM, 1997, p.212).

para armazenar tanto as instruções dos programas quanto os dados utilizados pelos mesmos. Qualquer programa, para ser executado, tem de estar na memória (FARRER, 1999).

Ao desenvolvermos nossos algoritmos, frequentemente precisamos armazenar dados referentes ao problema, como um nome, um número ou mesmo o resultado de uma operação. Mas, para armazenar esses dados, precisamos solicitar ao computador que ele reserve uma área da memória para nosso uso. A forma de solicitar ao computador que reserve memória é chamada de **declaração de variáveis.**

A seguir exibe-se como declarar variáveis em Portugol:

Sintaxe:

```
var
    nome_da_variavel : tipo_da_variavel
```

var
A palavra **var** é utilizada para indicar o início do bloco de declaração de variáveis de um algoritmo. Essa palavra é escrita apenas uma vez, independente do número de variáveis a serem declaradas. Mas, não se preocupe com isso agora. Nesse momento o importante é entender os conceitos de variável e de tipo de variável.

nome_da_variavel
Quando solicitamos que o computador reserve espaço de memória, temos de informar como vamos nos referir a essa área de memória reservada, ou seja, **qual nome** daremos a esse espaço de memória. Assim, toda variável tem um nome através do qual é referenciada. A seguir (item 2.2.1) apresentaremos um conjunto de regras que devem ser seguidas para dar nomes às variáveis. Caso queiramos declarar mais de uma variável de um mesmo tipo, basta escrever os nomes desejados para as variáveis separados por vírgula.

tipo_da_variavel
Precisamos informar o **tipo de dados** que armazenaremos na variável para que o computador saiba o tamanho do espaço de memória que reservará. A seguir (item 2.2.2) serão apresentados os tipos que podem ser utilizados.

2.2.1 Por que declarar variáveis e como nomeá-las?

Sempre que criamos uma variável, nós o fazemos com o objetivo de armazenar algum tipo de valor específico. Por exemplo, se estivermos desenvolvendo um algoritmo que calcule o imposto de renda a ser pago por um assalariado, precisamos de variáveis para armazenar o valor do salário e para os resultados dos cálculos. Assim, o nome dado à variável deve deixar claro o objetivo da mesma; devemos utilizar nomes sugestivos. Apesar de esta ser a principal diretriz quanto à atribuição de nomes a variáveis, algumas outras regras são apresentadas no Quadro 2.1.

Quadro 2.1: Regras para nomear variáveis

REGRA	EXEMPLO
Inicie sempre por um caractere alfabético, nunca por um número.	Nome (correto) - 1nome (errado)
Não utilize caracteres especiais como " , ( ) / * ; + .	Nome(M); N*B
Não coloque espaços em branco ou hífen entre nomes.	salario-bruto
Utilize, se necessário, *underline*.	salario_bruto
Crie suas variáveis com nomes sugestivos.	Se vai guardar salário de funcionários, dê à variável o nome **salario**.

2.2.2 O que são tipos de variáveis?

Quando declaramos uma variável, devemos ter em mente quais valores serão armazenados naquele espaço de memória. É essa observação que definirá o tipo da variável a ser declarado. Uma variável pode ser de um dos seguintes tipos:

- **Tipo inteiro:** Declararemos variáveis do tipo **numérico inteiro** quando precisarmos armazenar valores inteiros, positivos ou negativos (0, 1, 5, 7, -10, -5...). Por exemplo, se precisarmos de uma variável para armazenar o número de filhos de um funcionário, o tipo ideal para essa variável seria **inteiro**.

- **Tipo real**: Declararemos variáveis do tipo **numérico real** para armazenar valores reais; em outras palavras, valores com ponto decimal (5.7, 3.2, -8.5). Esse seria o tipo ideal para armazenar, por exemplo, o salário de funcionários.

- **Tipo caractere**: Declararemos variáveis do tipo **literal caractere** para armazenar um único caractere, que pode ser uma letra ou um símbolo. Por exemplo, para identificar o sexo do indivíduo, armazenaremos apenas o caractere 'F' ou 'M'.

Variável
É uma posição nomeada de memória, que é usada para guardar um valor que pode ser modificado pelo programa (LAUREANO, 2005, p. 12

- **Tipo cadeia**: Declararemos variáveis do **tipo literal cadeia** para armazenar uma sequência de caracteres, ou seja, uma palavra, uma mensagem, um nome. Assim, se precisarmos de uma **variável** para armazenar o nome de uma pessoa, esse seria o tipo ideal.

- **Tipo lógico:** Declararemos variáveis do tipo lógico para armazenar valores lógicos. O valor de variáveis desse tipo será sempre VERDADEIRO ou FALSO.

Lembrando a sintaxe:

```
var
     nome_variavel1, nome_variavel2 : tipo_variaveis1e2
     nome_variavel3 : tipo_variavel3
```

Assim, suponha que precisemos declarar uma variável para armazenar a idade de uma pessoa, uma para o peso e uma terceira para o nome. Para o peso e a idade utilizaremos variáveis do tipo inteiro e para o nome o tipo será cadeia. Logo, essas declarações ficariam assim:

```
var
      peso, idade : inteiro
          nome : cadeia
```

Atividade 2.1. Aprendemos algumas regras que devem ser seguidas para dar nomes a variáveis. Assinale os nomes de variáveis que obedecem a essas regras:

a) ( ) nome

b) ( ) telefone-celular

c) ( ) nome+sobrenome

d) ( ) 2taxa

e) ( ) telefone_celular

f) ( ) conta1

Atividade 2.2. Para cada valor dado abaixo foi definido um tipo de variável. Marque os pares "valor e tipo" definidos corretamente:

a) ( ) valor = 2.5 tipo= real

b) ( )valor = 'F' tipo= inteiro

c) ( ) valor = -2 tipo= inteiro

d) ( ) valor = 'M' tipo= caractere

e) ( ) valor = 5 tipo= cadeia

f) ( ) valor = -10.35 tipo= real

g) ( ) valor = 38 tipo= real

h) ( )valor = 'Jose' tipo= cadeia

i) ( ) valor = 135 tipo= inteiro

j) ( )valor = 7.5 tipo= inteiro

2.3. Você está fazendo um algoritmo para calcular a média dos alunos a partir das notas de duas provas. Assim, precisará de três variáveis: uma para a nota da primeira prova, uma para a nota da segunda prova e uma para a média. Segundo as normas da instituição, as notas das provas devem ser números inteiros de 0 a 10. Já para a média podem ser atribuídos valores com casas decimais. Utilizando a sintaxe de declaração de variáveis em Portugol e as regras para definição de tipos e de nomes, indique como você declararia essas 3 variáveis. **Dica: lembre-se de escolher nomes sugestivos para as variáveis.**

2.3 Constantes

Como aprendemos, o valor de uma variável pode ser alterado ao longo de seu algoritmo. Mas, às vezes, precisamos armazenar valores que não se alteram. Para isso existem as constantes.

As constantes são criadas obedecendo às mesmas regras de nomenclatura já vistas em variáveis. Diferem apenas no fato de

armazenar um valor constante, ou seja, que não se modifica durante a execução de um programa.

A sintaxe para a declaração de constantes em Portugol é dada abaixo:

Sintaxe:

const

nome_da_constante ← valor

const
A palavra const é utilizada para indicar o início do bloco de declaração de constantes de um algoritmo. A exemplo da palavra var, const também é escrita apenas uma vez, independente do número de constantes a serem declaradas.

nome_da_constante
É o nome pelo qual vamos nos referir à constante. Deve obedecer às mesmas regras que os nomes de variáveis. Apenas para ficar fácil a diferenciação entre variáveis e constantes em seus programas, aconselha-se que todas as letras dos nomes das constantes sejam maiúsculas.

A-Z

Constante
É uma variável com valor pré-definido que não pode ser modificado por nenhuma função de um programa (LAUREANO, 2005, p.16)

Como exemplo, considere um algoritmo que calcule o valor da contribuição de FGTS: 8% sobre o salário, independentemente do valor do salário. Assim, a taxa de 8% será constante durante a execução do programa. Logo, poderia declarar minha **constante** da seguinte forma:

```
const TAXA_FGTS ← 0.08
```

2.4 Comandos de atribuição, entrada e saída de dados

2.4.1 Comando de atribuição

Na construção de algoritmos, depois que declaramos nossas variáveis e constantes, geralmente precisamos indicar que elas armazenarão um determinado valor durante a execução do programa. Para isso, utilizamos o comando de atribuição que, em Portugol, é representado por uma seta (←), conforme sintaxe abaixo:

Sintaxe:

identificador ← expressão

identificador
Nome da variável ou constante a ser utilizada.

expressão
Valor ou expressão a ser armazenado (veremos ainda neste capítulo que podemos ter expressões aritméticas e expressões lógicas).

Exemplos:

```
var
      nota : inteiro  /*criamos a variável inteira nota*/
      sexo :caractere /*criamos a variável caractere sexo*/
      nota ← 10  /*atribuímos o valor 10 à variável nota*/
      sexo ← 'F' /*atribuímos o caractere 'F' à variável sexo*/
```

Os sinais /* e */ indicam o início e o fim de um comentário, que pode ocupar mais de uma linha. Os sinais // indicam um comentário que vai até o fim da linha.

Em programação utilizamos comentários para explicar a lógica utilizada em nossos algoritmos.

2.4.2 Comando de entrada de dados

Frequentemente, na construção de algoritmos, precisamos solicitar que usuários informem, por meio do teclado, alguns valores a serem utilizados durante a execução. Por exemplo: se fizermos um algoritmo para calcular a média das notas de um aluno, precisaremos solicitar quais foram as notas, para depois calcularmos a média. Esses valores informados devem ser armazenados em variáveis para que sejam utilizados quando necessário.

O comando de entrada de dados será responsável pela leitura e armazenamento desses dados na variável que indicarmos. A sintaxe do comando de entrada de dados em Portugol é exibida abaixo:

Sintaxe: leia(variavel)

leia( )
Função responsável por ler o que o usuário digitou e armazenar o valor na variável indicada.

variavel
Nome da variável utilizada para armazenar o valor digitado.

Exemplo:

```
var
  nota : inteiro //criamos a variável inteira nota
  leia(nota)  //atribuímos à variável nota o valor que o usuário digitar
```

2.4.3 Comando de saída de dados

Por meio da utilização do comando de saída de dados, conseguimos exibir mensagens ou valores para o usuário de nossos programas. É através desse comando que nosso algoritmo consegue se comunicar com o usuário para solicitar a entrada de dados ou para fornecer saídas de dados.

O comando de saída de dados exibe no monitor valores de constantes, variáveis ou expressões. A sintaxe do comando de saída de dados em Portugol é exibida abaixo:

Sintaxe: escreva(expressao)

escreva( )
Função responsável por escrever no monitor uma mensagem para o usuário.

expressao
Indica o que será escrito no monitor. É normalmente composta por um texto fixo seguido por uma vírgula e um nome de variável.

Para exemplificar a utilização dos comandos de entrada e de saída de dados, vamos criar nosso primeiro algoritmo completo em Portugol. Mas, antes disso, vamos ver a estrutura geral de um algoritmo em Portugol:

Estrutura Geral de um algoritmo em Portugol:

algoritmo nome_do_algoritmo

const
/*Aqui devem ser declaradas todas as constantes do nosso algoritmo seguindo as regras de sintaxe já estudadas. Caso nosso algoritmo não necessite de constantes, esta seção não deve existir.*/

var

/*Aqui devem ser declaradas todas as variáveis do nosso algoritmo seguindo as regras de sintaxe já estudadas. Caso nosso algoritmo não necessite de variáveis, esta seção não deve existir.*/

inicio

/*Aqui ficará o corpo do algoritmo, ou seja, o conjunto de comandos que formará o algoritmo em si.*/

fim

A seguir, nosso primeiro exemplo de algoritmo completo em Portugol. Neste exemplo solicitamos que o usuário digite seu nome e depois imprimimos na tela uma saudação ao mesmo. Note que, como não utilizamos constantes neste algoritmo, ele não faz uso do bloco const.

Exemplo:

```
algoritmo exemplo
var
  nome:cadeia /*criamos a variável nome do tipo cadeia*/
inicio
  escreva("Digite seu Nome")  /*solicitamos que o
                        usuário digite seu nome*/
  leia (nome)  /*lemos para a variável nome o valor
               digitado pelo usuário*/
  escreva ("Bom dia", nome) /*imprimimos na tela a
               mensagem Bom dia acompanhada pelo nome
               digitado pelo usuario*/
fim
```

Assim, caso o usuário tenha digitado o nome Maria, será exibida no monitor a seguinte mensagem: Bom dia Maria.

Atividade 2.4. Assinale os comandos de atribuição realizados corretamente:

a) ( ) var cadeia SEXO ← 'F'

b) ( ) var inteiro ALTURA ← 1.80

c) ( ) var real SALÁRIO ← 3000.00

d) ( ) var cadeia ← "NOME"

Atividade 2.5. No programa abaixo, dois valores inteiros são lidos e somados e o resultado dessa soma é mostrado no final da execução. Analise as linhas do programa e assinale as afirmações corretas:

```
linha 1 ... Algoritmo soma
linha 2 ... var
linha 3 ...    num1, num2, soma : inteiro
linha 4 ... inicio
linha 5 ...    escreva( "Digite o primeiro número")
linha 6 ...    leia(num1)
linha 7 ...    escreva( "Digite o segundo número")
linha 8 ...    leia(num2)
linha 9 ...    soma ← num1 + num2
linha 10...    escreva("A soma dos números digitados é: ", soma)
linha 11 ... fim.
```

a) ( ) linha 6 → O primeiro valor digitado no teclado está sendo lido e armazenado em num1,

b) ( ) linha 7 → O segundo valor digitado no teclado está sendo lido e armazenado em num2,

c) ( ) linha 9 → O resultado da soma dos valores digitados está sendo atribuído à variável soma,

d) ( ) linha 10 → No monitor serão exibidas a mensagem que está entre aspas e a soma dos números digitados.

Atividade 2.6. Faça um algoritmo que solicite que o usuário digite seu nome e a seguir solicite que seja digitada sua idade. Depois que o usuário digitar o nome e a idade, o programa deve exibir na tela duas mensagens: uma com o nome e outra com a idade do usuário. Suponha que o usuário seja o Pedro e tenha 32 anos. Assim, após a digitação dos dados, seu programa deve exibir as seguintes mensagens: **"Seu nome é Pedro"** e **"Você tem 32 anos".**

2.5 Operadores aritméticos e expressões aritméticas

Os operadores aritméticos são símbolos que representam operações aritméticas, ou seja, as operações matemáticas básicas. O Quadro 2.2 apresenta os operadores aritméticos que utilizaremos neste curso.

Quadro 2.2: Operadores aritméticos

Operador	Operação matemática
+	Adição
-	Subtração
*	Multiplicação
/	Divisão
%	Resto da Divisão Inteira

Os operadores aritméticos são utilizados para formar expressões aritméticas. As expressões aritméticas são formadas por operadores aritméticos que agem sobre operandos. Os operandos podem ser variáveis ou constantes do tipo numérico, ou seja, inteiros ou reais. Abaixo temos dois exemplos de expressões aritméticas:

nota/2

x*2+y/2

Para resolver expressões aritméticas formadas por mais de um operador, deve-se utilizar uma ordem de precedência entre os mesmos. O Quadro 2.3 exibe essa ordem de precedência.

Quadro 2.3: Ordem de precedência entre operadores aritméticos

Prioridade	Operador	Operação
1ª	* / %	multiplicação, divisão, resto da divisão
2ª	+ -	adição, subtração

Além da ordem de prioridades definida acima, podemos utilizar parênteses. Assim, resolvemos primeiro as expressões contidas nos parênteses mais internos, seguindo a ordem de precedência entre operadores, passando depois para os parênteses mais externos. Por exemplo, na expressão:

nota1 + (nota2 + nota3) / 2

Primeiro somamos nota2 a nota3. O resultado é divido por 2 e só depois

somamos com nota 1.

Um fato interessante é que o resultado da execução de expressões aritméticas é sempre um valor numérico (inteiro ou real) que pode então ser atribuído a uma variável numérica por meio do uso do comando de atribuição estudado anteriormente.

Resumindo:

- expressões aritméticas: uma conta a ser feita;
- operadores aritméticos: os sinais + - * / %;
- operandos aritméticos: constantes ou variáveis (inteiras ou reais), ou outra expressão aritmética;
- precedência na ordem dos cálculos: a mesma da matemática, podendo haver vários níveis de parênteses; não há colchetes [ ] ou chaves { }.

Como exemplo, vamos criar um algoritmo para ler e multiplicar dois números inteiros e exibir o resultado.

É importante observar cada linha desta sequência:

```
linha 1 ... Algoritmo multiplicacao
linha 2 ... var
linha 3 ...   num1, num2, mult : inteiro
linha 4 ... inicio
linha 5 ...   escreva ( "Digite o primeiro número: ")
linha 6 ...   leia (num1)
linha 7 ...   escreva ( "Digite o segundo número: ")
linha 8 ...   leia (num2)
linha 9 ...   mult ← num1 * num2
linha 10...   escreva( "O resultado da multiplicação é: " , mult)
linha 11 ..fim
```

Vamos entender todas as linhas do nosso algoritmo:

linha 1 ... Nome do programa.

linha 2 ... Indica o início do bloco de declaração de variáveis.

linha 3 ... Declaração das três variáveis do tipo inteiro necessárias ao programa.

linha 4 ... Indica o início do bloco de instruções do algoritmo.

linha 5 ... O comando **escreva** exibirá na tela do computador uma mensagem que solicita a digitação do primeiro número.

linha 6 ... O primeiro número digitado será lido e armazenado na variável num1.

linha 7 ... O comando **escreva** exibirá na tela do computador uma mensagem que solicita a digitação do segundo número.

linha 8 ... O segundo número digitado será lido e armazenado na variável num2.

linha 9 ... A variável mult receberá o resultado da multiplicação do primeiro número pelo segundo número.

linha 10 ... O comando **escreva** exibirá na tela do computador uma mensagem com o resultado da multiplicação.

linha 11 ... Indica o fim do algoritmo.

Resumo

Nesta aula você viu alguns conceitos básicos de programação, variáveis, constantes e tipos de dados e conheceu uma linguagem formal para escrever algoritmos: o Portugol.

Você implementou seus primeiros algoritmos em Portugol, utilizando comandos de atribuição, entrada e saída de dados e operadores aritméticos.

Atividades de aprendizagem

Guarde esses algoritmos, você os utilizará mais tarde. Aliás, isto é um trabalho de engenharia: você sempre deve aproveitar o que desenvolveu antes e não ficar reinventando a roda.

1. Resolva as seguintes expressões aritméticas considerando A=2, B=5 e C=10:

a) A+B*C/A

b) B+C%A*(B-A/2)

c) (B+C)%2+A*(B+(C*4))

2. O algoritmo abaixo deverá ler duas notas, calcular a média e mostrar o resultado. Para que o algoritmo seja executado corretamente, complete-o com os comandos que faltam:

```
linha 1 ... Algoritmo media
linha 2 ... var
linha 3 ...       nota1, nota2, media : ____________
linha 4 ... inicio
linha 5 ...    __________ ( "Digite a primeira nota")
linha 6 ...    _____(nota1)
linha 7 ...    escreva( "_____________________")
linha 8 ...    leia(________)
linha 9 ...    media ← (_______ + _______ )/2
linha 10...  escreva("A média das notas é: ", _______)
linha 11 ..fim.
```

3. Faça o mesmo no algoritmo abaixo, cuja finalidade é calcular 8% de aumento sobre um salário:

```
linha 1 ... Algoritmo reajuste
linha 2 ... var
linha 3 ...   salario, salario_novo:____________
linha 4 ... inicio
linha 5 ...   __________ ( "Digite o salário")
linha 6 ...   __________(salario)
linha 7 ...   salario_novo ← ____________* 1.08
linha 8 ...   _______("O valor do novo salário é: ",____
linha 9 ... fim.
```

4. Faça um algoritmo que leia um número inteiro e imprima seu antecessor e seu sucessor.

5. Faça um algoritmo que leia dois números reais e imprima a soma e a média aritmética desses números.

6. Faça um algoritmo que receba como entrada as medidas dos dois catetos de um triângulo retângulo e calcule e exiba a medida da hipotenusa e a área do triângulo.

Aula 3 - Expressões lógicas e estruturas de decisão

Objetivos

Conhecer os operadores relacionais.

Conhecer os operadores lógicos e suas Tabelas-verdade.

Compreender a ordem de precedência entre os operadores em expressões lógicas.

Conhecer a formalização de uma estrutura de decisão no contexto de um algoritmo em Portugol.

Construir algoritmos em Portugol com estruturas de decisão.

3.1 Operadores relacionais, operadores lógicos e expressões lógicas

Para entender o conceito e o uso de expressões lógicas, primeiro precisamos conhecer os operadores lógicos e os operadores relacionais, pois as expressões lógicas são formadas a partir da utilização desses operadores.

3.1.1 Operadores relacionais

Os operadores relacionais são utilizados para realizar comparações entre dois valores de um mesmo tipo. Esses valores podem ser representados por variáveis ou constantes. Os operadores relacionais são apresentados no Quadro 3.1.

Quadro 3.1: Operadores relacionais

Descrição	Símbolo
igual a	=
maior que	>
menor que	<
maior ou igual a	>=
menor ou igual a	<=
diferente de	!=

A uma comparação realizada utilizando um operador relacional dá-se o nome de relação. O resultado obtido de uma relação é sempre um valor lógico, ou seja, **verdadeiro** ou **falso**.

No Quadro 3.2 temos exemplos de relações e seus resultados. Para tais exemplos, considere duas variáveis inteiras, **A** e **B**, onde A = 5 e B = 8.

Quadro 3.2: Exemplos de relações e seus resultados (para A = 5 e B = 8)

Relação	Resultado
A = B	Falso
A < B	Verdadeiro
A >= B	Falso
B != 6	Verdadeiro
A >= 5	Verdadeiro

3.1.2 Operadores lógicos

Os operadores lógicos retornam **verdadeiro (V)** ou **falso (F)** de acordo com seus operandos. Os operadores lógicos mais comuns são listados no Quadro 3.3.

Quadro 3.3: Operadores lógicos

E
OU
NÃO

Os operadores lógicos também são conhecidos como **conectivos**, pois são utilizados para formar novas proposições a partir da junção de duas outras. Para entender o funcionamento de operadores lógicos, vamos recorrer ao nosso exemplo das variáveis inteiras, **A** e **B** onde A = 5 e B = 8. O Quadro 3.4 exibe exemplos de expressões que usam operadores lógicos.

Quadro 3.4: Expressões utilizando operadores lógicos e seus resultados (para A = 5 e B = 8)

Relação	Resultado
A<6 E B > 7	Verdadeiro: o valor de A é menor que 6 E o valor de B é maior que 7.
A = 5 E B < 5	Falso: apesar de o valor de A ser igual a 5, o valor de B não é menor que 5.
A = 5 OU B < 5	Verdadeiro: usando o operador OU, se ao menos uma das condições for verdadeira (A = 5), o resultado da expressão é verdadeiro.

Você deve ter notado, pelos exemplos anteriores:

- quando utilizamos o operador lógico E, o resultado só será verdadeiro se as duas condições relacionadas forem verdadeiras;

- para o operador OU, basta que uma das condições seja verdadeira para que o resultado seja verdadeiro;

- em consequência: com o operador OU, para que o resultado seja falso as duas condições devem ser falsas.

Para visualizar todas as opções possíveis ao combinar operadores lógicos, utilizamos as **Tabelas-verdade**.

O Quadro 3.5 apresenta a Tabela-verdade para o operador lógico **OU**; o Quadro 3.6 para o operador lógico **E** e o Quadro 3.7 para o operador lógico **NÃO**. Os operadores **OU** e **E** trabalham com duas expressões: nos Quadros 3.5 e 3.6 essas expressões são representadas por **P** e **Q**. Já o operador **NÃO** trabalha sobre apenas uma expressão, representada no Quadro 3.7 por **P**.

Quadro 3.5: Tabela-verdade do operador OU

P	Q	P ou Q
V	V	V
V	F	V
F	V	V
F	F	F

Quadro 3.6: Tabela-verdade do operador E

P	Q	P e Q
V	V	V
V	F	F
F	V	F
F	F	F

Quadro 3.7: Tabela-verdade do operador NÃO

P	Não P
V	F
F	V

3.1.3 Expressões lógicas

Como foi dito no início desta aula, as **expressões lógicas** são expressões formadas a partir do uso de variáveis e constantes, operadores relacionais e operadores lógicos. As expressões lógicas são avaliadas e retornam sempre um valor lógico: **verdadeiro** ou **falso**.

As relações utilizadas nos Quadros 3.2 e 3.4 para explicar a utilização dos operadores lógicos e relacionais são bons exemplos de expressões lógicas.

Outros Exemplos: (x < y) E (y < z)

(y + z < x) OU (x>10) E (y < 5)

Como podemos ver nos exemplos acima, podem ser combinados, em uma mesma expressão, operadores relacionais, lógicos e aritméticos. Assim, é importante compreender a ordem de precedência entre eles, pois isso irá definir a forma de solução da expressão (Quadro 3.8).

Quadro 3.8: Ordem de precedência entre operadores em expressões lógicas

Prioridade	Operador
1ª	Operadores aritméticos seguindo a ordem de precedência indicada na seção 2.5 (Quadro 2.3)
2ª	Operadores relacionais
3ª	Operador lógico **NÃO**
4ª	Operador lógico **E**
5ª	Operador lógico **OU**

Exemplo: Dadas as variáveis e as seguintes atribuições:

var NUM1, NUM2, NUM3, NUM4: inteiro

inicio
NUM1⇐10
NUM2⇐5
NUM3⇐200
NUM4⇐200

Vamos verificar se a expressão (NUM1+ NUM2 > 10 **E** NUM1+ NUM3 > NUM4) é VERDADEIRA (**V**) ou FALSA (**F**):

→

Vamos analisar todas as etapas necessárias:

1. NUM1+ NUM2 > NUM1 é o mesmo que (10+5>10). Pela Tabela de precedências, vimos que primeiro resolvemos os operadores aritméticos; assim, temos (15 >10). Logo, a resposta é **V**, já que 15 é maior que 10.

2. NUM1+ NUM3 > NUM4 é o mesmo que (10+200>200). Assim, a resposta é **V**, já que 10+200 é maior que 200.

3. Assim, nossa expressão se resumirá em **V E V**.

4. Na Tabela-verdade aprendemos que numa proposição **V E V**, o resultado será **V**.

5. Portanto, o resultado final é: **V**, ou seja, Verdadeiro.

Hum! Isto pode ficar confuso.

- uma forma de você evitar confusão é definir as precedências usando parênteses. As expressões dentro dos parênteses mais internos são resolvidas primeiro;

- você pode testar o uso de parênteses também usando uma planilha de cálculo como o Excel.

Observe as seguintes declarações de variáveis e suas respectivas atribuições e responda às questões abaixo:

```
var
num1, num2, num3, num4 : inteiro
inicio
num1←10
num2←2
num3←200
num4←200
```

Atividade 3.1 Coloque F ou V nas expressões abaixo:

Exemplo: (F) NUM4 > NUM3
a) ( ) NUM1 > NUM2
b) ( ) NUM1 < NUM3
c) ( ) NUM1 < NUM4
d) ( ) NUM3 = NUM4

Atividade 3.2 Coloque F ou V nas expressões abaixo:

Exemplo: (F) NUM1-NUM2 < NUM2
a) ( ) NUM1+ NUM2 > NUM3
b) ( ) NUM1* NUM2 < NUM4
c) ( ) NUM3 - NUM4 != NUM4
d) ( ) NUM3 / NUM1 < NUM4

Atividade 3.3 Coloque F ou V nas expressões abaixo:

Exemplo: (F) NUM1+ NUM2 > 10 e NUM3 - NUM4 = NUM3
a) ( ) NUM1 / NUM2 > 0 **e** NUM1 + NUM3 > NUM4
b) ( ) NUM1 * NUM2 > 40 **e** NUM3 - NUM1 > NUM4
c) ( ) NUM1 - NUM2 = 10 **e** NUM2 + NUM3 > NUM4
d) ( ) NUM1 + NUM2 < 10 **e** NUM3 - NUM4 = NUM1

Atividade 3.4 Coloque F ou V nas expressões abaixo:

Exemplo: (V) NUM3 / NUM2 >55 ou NUM1+ NUM3 > NUM4
a) ( ) NUM3 / NUM2 > 0 **ou** NUM1 + NUM3 > NUM4
b) ( ) NUM2 * NUM1 = 50 **ou** NUM3 - NUM1 > NUM4
c) ( ) NUM1 - NUM2 > 10 **ou** NUM2 + NUM3 > NUM4
d) ( ) NUM1 + NUM2 > 10 **ou** NUM3 / NUM1 > NUM4

Atividade 3.5 Coloque F ou V nas expressões abaixo:

Ex.: (V) NUM1 > NUM2 **e** NUM2 < NUM3 **ou** NUM3 < NUM4
a) ( ) NUM1 > NUM2 **e** NUM2 < NUM3 **ou** NUM3 < NUM4
b) ( ) NUM1 * NUM2 > 10 **e** NUM1 > NUM4 **ou** NUM3 - NUM1 > NUM4
c) ( ) NUM1 > 10 **ou** NUM1> NUM4 **e** NUM3 - NUM1 > NUM4
d) ( ) NUM1 + NUM2 > 10 **ou** NUM1 / NUM3 > NUM4 **e** NUM3 < NUM4

3.2 Estrutura de decisão

Como vimos na aula 1, muitas vezes precisamos tomar decisões que podem interferir diretamente no andamento do algoritmo. A representação dessas decisões em nossos programas é feita através do uso de **estruturas de seleção**, ou estruturas de decisão.

A estrutura de seleção que utilizaremos em Portugol será a estrutura **SE... ENTÃO... SENÃO**. A sintaxe dessa estrutura é exibida abaixo:

Sintaxe: SE <expressão lógica>
ENTÃO
<bloco de instruções verdade>
SENÃO
<bloco de instruções falso>
FIM SE

Como funciona?

A palavra reservada **SE** indica o início da estrutura de seleção. Após essa palavra, vem a condição que definirá o bloco a ser executado. Qualquer expressão lógica poderá ser utilizada como condição, pois deverá retornar verdadeiro ou falso. Caso a expressão da condição seja verdadeira, o bloco de instruções **ENTÃO** será executado. Caso contrário, o bloco **SENÃO** o será. A palavra reservada **FIM SE** indica o final da estrutura de seleção.
Vale ressaltar que o bloco **SENÃO** não é obrigatório: caso só queiramos executar algumas instruções se a expressão lógica for verdadeira, podemos omitir o bloco **SENÃO**.

Como primeiro exemplo, faremos um programa que solicita que o usuário preencha suas duas notas, tire a média das duas e, caso a média seja maior ou igual a 60, apresente uma mensagem parabenizando-o pela aprovação:

A-Z

Estruturas de seleção
Os comandos de seleção ou de decisão são técnicas de programação que conduzem a estruturas de programas que não são totalmente sequenciais. Uma estrutura de seleção permite a escolha de um grupo de ações a ser executado quando determinadas condições, representadas por expressões lógicas, são ou não satisfeitas.

```
Algoritmo Media
var
   nota1, nota2, media: real  /*criamos 3 variáveis do tipo
                              real*/
inicio
   leia(nota1)  /*esperamos que o usuário digite a nota1*/
   leia(nota2)  /*esperamos que o usuário digite a nota2*/
   media ← (nota1 + nota2)/2  /*atribuímos o resultado da
            expressão que faz a média das notas*/
   se media >= 60  /*verificamos se a média é maior que 60*/
      então
         escreva("Parabéns! Aprovado!") ../*se média maior que
                  60, apresenta mensagem de aprovação*/
   fim se   /*finaliza a estrutura de seleção*/
   fim.
```

Atenção: Note, no algoritmo acima, que não utilizamos o bloco SENÃO da estrutura de seleção. Com isso, caso a média seja menor que 60, nada acontecerá no programa.

Assim, a seguir apresentamos uma estrutura de decisão acrescentando o bloco SENÃO para que mostre uma mensagem de reprovação.

```
Algoritmo Media
var
   nota1, nota2, media: real  /*criamos 3 variáveis do tipo
                              real*/
inicio
   leia(nota1)  /*esperamos que o usuário digite a nota1*/
   leia(nota2)  /*esperamos que o usuário digite a nota2*/
   media ← (nota1 + nota2)/2  /*atribuímos o resultado da
            expressão que faz a média das notas*/
   se media >= 60  /*verificamos se a média é maior que 60*/
      então
         escreva("Parabéns! Aprovado!") ../*se média maior que
                  60, apresenta mensagem de aprovação*/
      senão
         escreva("Aluno Reprovado!") ../*se média menor que 60
                  apresenta mensagem de reprovação*/
   fim se   /*finaliza a estrutura de seleção*/
fim.
```

Note que, nessa nova versão do algoritmo, caso a média seja maior ou igual a 60 continua sendo exibida uma mensagem informando que o aluno foi

aprovado. Mas, caso o aluno não atinja a nota 60, passa a ser exibida uma mensagem informando a reprovação.

Você deve ter notado que, quando escrevemos algoritmos, algumas linhas têm um recuo maior em relação à margem. Essa técnica é utilizada para melhorar a organização do código e é chamada de identação ou endentação. Note no algoritmo acima que a linha **leia(nota1)** está com uma distância maior para a margem do que a linha **inicio**. Isso é feito para deixar claro que a linha **leia(nota1)** está dentro do bloco de programa iniciado pela linha **inicio.** Note que na estrutura de decisão também fazemos uso da endentação para facilitar a visualização do código.

Resumo

Nesta aula você conheceu os operadores relacionais e os operadores lógicos. Aprendeu a utilizar a estrutura de decisão **SE...ENTÃO...SENÃO** e construiu seus primeiros algoritmos em Portugol, utilizando essa estrutura.

Atividades de aprendizagem

1. O algoritmo abaixo deve ler o salário bruto e calcular o salário líquido. Neste exemplo, o salário líquido será o salário bruto menos os descontos de INSS e IR, calculados segundo as seguintes regras: caso o salário seja menor que R$ 1.500,00 não devemos descontar IR e descontaremos 8% de INSS; para salários a partir de R$ 1.500,00 descontaremos 5% de IR e 11% de INSS. Ao final deve ser exibido o novo salário. Para que o algoritmo seja executado corretamente, complete-o com os comandos que faltam.

Obs.: Essas faixas de cálculo são fictícias, apenas para exemplo, não condizendo com as leis em vigor no país.

```
linha 1 .. Algoritmo calcula_liquido
linha 2 .. var
linha 3 ..    bruto, liquido, inss, ir: __________
linha 4 .. inicio
linha 5 ..    __________ ( "Digite o salário")
linha 6 ..    _______(bruto)
linha 7 ..    se __________ < 1500
linha 8 ..      então
linha 9 ..         inss ← ___________* 0.08
linha 10..         ir ← _____________
linha 11..      senão
linha 12..         ______ ← bruto * 0.11
linha 13..         ir ← bruto * _____
linha 14..    fim se
linha 15..    liquido ← bruto - _____ - _____
linha 16..    ______("O valor do salário líquido é:",_____)
linha 17.. fim.
```

2. Sabendo que triângulo é uma Figura geométrica de três lados onde cada um dos lados é menor que a soma dos outros dois, queremos fazer um algoritmo que receba três valores e verifique se eles podem ser os comprimentos dos lados de um triângulo. Neste contexto, complete o algoritmo abaixo para que funcione:

```
linha 1.. Algoritmo verifica_triangulo
linha 2.. var
linha 3..   lado1, lado2, lado3: real
linha 2.. inicio
linha 4..   __________ ( "Digite os valores dos 3 lados.")
linha 5..   __________(lado1)
linha 6..   __________(lado2)
linha 7..   leia(____________)
linha 8..   se lado1+lado2 < _____ e lado2+lado3<lado1 ___
                          lado1+_____< lado2
linha 9..     então
linha 10.       ________("Podemos construir um triângulo com
                         estas dimensões!")
linha 11.     senão
linha 11.       escreva("_____________________________")
linha 12.     fim _____
linha 13. fim.
```

3. Escreva um algoritmo que leia um número inteiro e diga:

- Se ele é par ou ímpar. Dica: utilize o operador % (resto da divisão inteira).
- Se ele é positivo, negativo ou nulo (zero).

4. Escreva um algoritmo que leia a idade de um atleta e escreva na tela em que categoria ele se enquadra, seguindo o quadro abaixo:

Faixa Etária	Categoria
de 5 a 10 anos	Infantil
de 11 a 17 anos	Juvenil
de 18 a 30 anos	Profissional
acima de 30 anos	Sênior

Aula 4 –Estruturas de repetição

Objetivos

Compreender a utilização de estruturas de repetição em algoritmos.

Conhecer dois tipos de estruturas de repetição em Portugol e avaliar quando utilizar cada uma.

Construir algoritmos em Portugol com estruturas de repetição.

Conforme vimos na primeira aula, são comuns as situações nas quais precisamos repetir determinadas ações enquanto não atingimos um objetivo. Da mesma forma, ao desenvolver nossos algoritmos, deparamos com situações nas quais precisamos repetir um conjunto de instruções até que uma determinada condição ocorra. Nessas situações, utilizaremos os comandos de repetição, também conhecidos como **laços** ou *loops*.

A-Z

Laços
As estruturas de repetição ou laços são técnicas de programação que permitem que um bloco de instruções seja executado várias vezes até que uma determinada condição seja satisfeita. Assim essas estruturas são compostas por uma condição e um conjunto de instruções a serem executados enquanto a condição não for satisfeita.

Na aula anterior desenvolvemos um algoritmo que lê duas notas de um aluno, calcula sua média e indica se o mesmo foi aprovado ou reprovado. Caso seja necessário calcular a média de dois alunos utilizando apenas as estruturas que já estudamos, teríamos de praticamente duplicar todo o algoritmo. Veja o exemplo a seguir:

```
Algoritmo Media
var
  nota1, nota2, media: real  /*criamos 3 variáveis do tipo
                               real*/
inicio
   //Trecho para analisar o primeiro aluno
  escreva("Digite a nota 1") /*solicitamos a primeira nota*/
   leia(nota1) /*esperamos que o usuário digite a nota1*/
   escreva("Digite a nota 2")/*solicitamos a segunda nota*/
   leia(nota2) /*esperamos que o usuário digite a nota2*/
   media ← (nota1 + nota2)/2 /*atribuímos o resultado da
                  expressão que faz a média das notas*/
   se media >= 60 /*verificamos se a média é maior que 60*/
      então
            escreva("Parabéns! Aluno Aprovado!) ../*se média
                  maior que 60 apresenta mensagem de
                  aprovação*/
      senão
           escreva("Aluno Reprovado!) ../*se média menor que
         60 apresenta mensagem de reprovação*/
   fim se  /*finaliza a estrutura de seleção*/

  //Agora repetimos todos os passos para o segundo aluno


    escreva("Digite a nota 1") /*solicitamos a primeira nota*/
   leia(nota1) /*esperamos que o usuário digite a nota1*/
   escreva("Digite a nota 2")/*solicitamos a segunda nota*/
   leia(nota2) /*esperamos que o usuário digite a nota2*/
   media ← (nota1 + nota2)/2 /*atribuímos o resultado da
                  expressão que faz a média das notas*/
   se media >= 60 /*verificamos se a média é maior que 60*/
      então
            escreva("Parabéns! Aluno Aprovado!) ../*se média
                  maior que 60 apresenta mensagem de
                  aprovação*/
      senão
           escreva("Aluno Reprovado!) ../*se média menor que
         60 apresenta mensagem de reprovação*/
   fim se  /*finaliza a estrutura de seleção*/
fim.
```

No exemplo acima, nota-se que, para executar as mesmas ações duas vezes, tivemos de duplicar o código. Agora, imagine se nosso objetivo fosse calcular a média de uma turma de 20 alunos! Teríamos de repetir por 20 vezes o bloco de instruções que calcula a média.

É em situações assim que as estruturas de repetição são úteis. Com elas, um bloco de instruções do seu algoritmo pode ser repetido diversas vezes sem

ter de duplicar o código.

Aprenderemos a utilizar duas estruturas de repetição: **para...faça e enquanto...faça.**

4.1 Estrutura de repetição para...faça

A estrutura de repetição **para...faça** é utilizada quando um determinado bloco de instruções deve ser repetido um número fixo conhecido de vezes.

Sintaxe:

para <**variável de controle**> de <**valor inicial**> ate <**valor final**> passo <**incremento**> faca

<**bloco de instruções**>

fim para

Como funciona?

A palavra reservada **para** indica o início da estrutura de repetição. Após essa palavra vem o nome de uma variável que irá controlar a quantidade de repetições do laço. Essa variável deve ser do tipo **inteiro** e deve ter sido declarada no bloco de declarações de variáveis do seu algoritmo (**bloco var**). Depois de definida a variável de controle, aparece a palavra reservada **de**, que define o valor inicial que será atribuído à variável de controle. Em seguida aparece a palavra reservada **ate** definindo o valor que irá encerrar o laço, ou seja, as instruções serão repetidas enquanto a **variável de controle** tiver valor menor ou igual ao **valor final**. Por fim é definido o valor de incremento através da palavra reservada **passo**. Esse valor é utilizado para incrementar a variável de controle após cada ciclo de execução.

Como primeiro exemplo, vamos supor que queiramos apenas imprimir na tela do computador todos os números inteiros de 1 até 10. Vamos fazer isso utilizando a estrutura para...faça.

```
linha 1. Algoritmo contagem_de_1_a_10
linha 2. var
linha 3.    contador : inteiro /*essa variável fará o controle do
                               laço*/
linha 4. inicio
linha 5.    para contador de 1 ate 10 passo 1 faca
linha 6.        escreva(contador)
linha 7.    fim para
linha 8. fim.
```

Agora vamos entender nosso algoritmo. Nas 4 primeiras linhas não temos novidades: apenas nomeamos o algoritmo, definimos uma variável e iniciamos o programa.

Na linha 5 estamos definindo um laço utilizando a variável **contador** como controle. Note que iniciamos nosso laço atribuindo o valor **1** à variável **contador** e indicamos que o laço irá até **10** com passo **1**, ou seja, a primeira vez que o laço for executado a variável **contador** será igual a 1. Então, o comando **escreva** presente na linha 6 imprimirá 1 na tela.

Após a execução da escrita, chegamos ao final do laço (linha 7 ... **fim para**). Nesse ponto, a variável de controle é incrementada pelo valor de **passo** (no nosso exemplo o valor de **passo** é 1). Assim, a variável **contador** passa a ter o valor 2. Como 2 é menor ou igual a 10 (valor final do laço), o laço é repetido. Assim, o comando **escreva** que aparece na linha 6 será executado novamente e imprimirá **2** na tela.

Novamente a variável **contador** é incrementada em 1, passando a valer 3. Como 3 ainda é menor ou igual a 10 (valor final do laço), o laço é repetido, imprimindo 3 na tela. E assim a execução segue até que a variável de controle atinja um valor maior que 10.

Dessa maneira, ao executar esse algoritmo, seria impresso na tela o seguinte: **12345678910**

Agora que entendemos como utilizar uma estrutura **para...faça,** vamos voltar ao nosso exemplo do cálculo da média de alunos. Suponha que queiramos fazer um algoritmo para calcular as médias de 20 alunos. Nesse exemplo, sabemos que o bloco responsável pelo cálculo das médias terá de se repetir por 20 vezes. Assim, vamos fazer um laço utilizando a estrutura **para...faça** tendo como valor inicial **1**, valor final **20** e incremento **1**. Assim, a variável de controle começará com **1** e, a cada vez que o laço for executado, ela será incrementada em **1**, até chegar ao valor **20**. Ou seja, o laço será executado 20 vezes. Esse algoritmo está implementado no Exemplo 2.

```
linha 1.. Algoritmo media_com_repeticao
linha 2.. var
linha 3..    nota1, nota2, media : real
linha 4..    contador : inteiro //essa variável será o controle do
                                laço
linha 5.. inicio
linha 6..    para contador de 1 ate 20 passo 1 faca
linha 7..        escreva ("Digite a primeira nota do aluno: ",
                         contador)
linha 8..        leia (nota1)
linha 9..        escreva ("Digite a segunda nota do aluno: ",
                          contador)
```

```
linha 10.        leia (nota2)
linha 11.        media ← (nota1 + nota2)/2
linha 12.        se media >= 60..... /*verificamos se a média é
                                      maior que 60*/
linha 13.            então
linha 14.                escreva("Aluno Aprovado!")
linha 15.            senão
linha 16.                escreva("Aluno Reprovado!")
linha 17.        fim se
linha 18.    fim para
linha 19. fim.
```

Na linha 6 deste exemplo, estamos iniciando um laço que termina na linha 18. O trecho que vai da linha 7 até a linha 17 está dentro do laço, ou seja, se repetirá 20 vezes, pois, pela definição da linha 6, nosso laço vai de 1 a 20 com passo 1.

Note que o trecho que está dentro do laço é idêntico ao algoritmo para cálculo de média que vimos na seção anterior. Assim, por 20 vezes, nosso programa repetirá o procedimento de solicitar duas notas, calcular a média e indicar se o aluno foi aprovado ou reprovado. Na primeira vez que o laço for executado a variável contador terá o valor 1. Na segunda vez, 2, e assim sucessivamente, até exceder o valor 20.

Atividade 4.1 O que será exibido na tela ao executar o algoritmo abaixo?

```
Algoritmo contagem
var
        c : inteiro
inicio
        para c de 0 ate 10 passo 2 faca
             escreva(c)
        fim para
fim.
```

Atividade 4.2 Faça um algoritmo que exiba na tela todos os números ímpares entre 100 e 200.

Atividade 4.3 Faça um algoritmo que leia 15 números inteiros e, para cada um deles, exiba o antecessor, o sucessor, o dobro e o triplo.

Atividade 4.4 Faça um algoritmo que imprima na tela a tabuada de multipli-

cação por 6. O programa deve imprimir na primeira linha a multiplicação de 6 por 1; na segunda, de 6 por 2; e assim sucessivamente, até a décima (6 vezes 10):

6 x 1 = 6
....
6 x 10 = 60

4.5 Em 2010, uma pequena cidade brasileira tem 20.000 habitantes. A previsão do IBGE é que esta cidade cresça a uma taxa de 5% ao ano. Sabendo disso, faça um algoritmo que imprima na tela o ano e a população prevista para a cidade em tal ano, com o ano variando de 2011 até 2030.

4.6 Foi dado como exemplo um algoritmo que lê duas notas de vinte alunos e indica se cada um deles foi aprovado ou reprovado. Altere aquele algoritmo de forma que leia as notas dos vinte alunos e, ao final, apenas imprima as quantidades de aprovados e de reprovados.

4.2 Estrutura de repetição enquanto...faça

A estrutura de repetição **enquanto...faça** é utilizada quando um determinado bloco de instruções deve ser repetido enquanto uma determinada condição for verdadeira.

Sintaxe: enquanto <condição de repetição> faca

<bloco de instruções >

fim enquanto

Como funciona?

A palavra reservada **enquanto** indica o início da estrutura de repetição. Após essa palavra, vem a **condição de repetição**. Qualquer expressão lógica poderá ser utilizada como condição de repetição, pois deverá retornar verdadeiro ou falso. Caso a condição seja verdadeira, o bloco de instruções será executado. Ao final da execução do bloco de instruções, a **condição de repetição** é testada novamente e, caso continue verdadeira, todo o bloco será executado novamente e assim sucessivamente até que a **condição de repetição** se torne falsa.

Como primeiro exemplo, vamos fazer uma nova implementação do algoritmo que imprime os números de 1 a 10; agora, porém, utilizaremos a estrutura **enquanto...faça** no lugar da estrutura **para...faça** utilizada anteriormente.

```
linha 1.. Algoritmo contagem_de_1_a_10
linha 2.. var
linha 3..       contador : inteiro /*essa variável será o controle
                                    do laço*/
linha 4.. inicio
linha 5..       contador ← 1
linha 6..       enquanto contador <= 10 faca
linha 7..            escreva(contador)
linha 8..            contador ← contador + 1
linha 9..       fim enquanto
linha 10. fim.
```

Vamos analisar nosso exemplo.

Da linha 1 até a linha 4 apenas nomeamos o algoritmo, declaramos as variáveis e indicamos o início do bloco de instruções.

Na linha 5 iniciamos a variável **contador** com o valor 1. Quando fizemos este exemplo, utilizando a estrutura **para...faça**, essa linha não era necessária, pois, como vimos, a própria estrutura coloca na variável um **valor inicial**.

Na linha 6 iniciamos a estrutura do laço **enquanto...faça** com a condição de repetição definindo que o laço será executado enquanto a variável **contador** for menor ou igual a 10.

Na linha 7 o valor da variável **contador** é impresso na tela e na linha 8 o valor dessa mesma variável é incrementado em 1. Assim, na primeira vez que o laço for executado, será impresso na tela o valor 1 e a variável terá seu valor incrementado para 2. Como 2 é menor que 10, a condição de repetição continuará verdadeira, fazendo com que o laço seja executado novamente, e assim sucessivamente, até que o valor da variável **contador** atinja **11** e a condição de repetição seja quebrada, encerrando o laço.

Note que, nesse caso, o algoritmo utilizando a estrutura **para...faça** ficou menor que o algoritmo utilizando a estrutura **enquanto..faça**.

Mas, vamos voltar ao exemplo do cálculo de média de alunos. Quando falamos da estrutura **para...faça** desenvolvemos um algoritmo que calcula

as médias de 20 alunos indicando se cada um foi aprovado ou não. Agora vamos supor uma situação na qual não se saiba a quantidade de alunos da turma e se queira calcular a média de todos os alunos. Nesse caso a saída seria calcular a média de um aluno e depois perguntar ao usuário se ele deseja digitar as notas de mais um aluno. Caso a resposta seja positiva, todo o processo de digitar notas, calcular a média e perguntar se deseja digitar para mais um aluno será repetido. E assim esse processo ocorrerá até que o usuário informe que não deseja calcular média de outro aluno. Nesse caso, utilizaremos a estrutura **enquanto...faça**. Veja o exemplo a seguir:

Exemplo 2:

```
linha 1.. Algoritmo media_com_repeticao
linha 2.. var
linha 3..    nota1, nota2, media : real
linha 4..    resposta : caracter /*essa variável armazenará a
                                   resposta*/
linha 5.. inicio
linha 6..    resposta ← 'S'
linha 7..    enquanto resposta = 'S' faca
linha 8..       escreva ("Digite a primeira nota do aluno ")
linha 9..       leia (nota1)
linha 10.       escreva ("Digite a segunda nota do aluno ")
linha 11.       leia (nota2)
linha 12.       media ← (nota1 + nota2)/2
linha 13.       se media >= 60..... /*verificamos se a média é
                                      maior que 60*/
linha 14.          então
linha 15.             escreva("Aluno Aprovado!")
linha 16.          senão
linha 17.             escreva("Aluno Reprovado!")
linha 18.       fim se
linha 19.       escreva ("Quer calcular média de outro aluno? S/N
                  ")
linha 20.       leia (resposta)
linha 21.    fim enquanto
linha 22. fim.
```

Como sempre fazemos, vamos analisar nosso exemplo:
Na linha 4 declaramos uma variável do tipo caracter com o nome **resposta** que será utilizada para informar se o usuário deseja ou não digitar as notas de mais um aluno. Na linha 6 armazenamos um "**S**" nessa variável. Fazemos isso para que a condição de repetição da linha 7 seja verdadeira na primeira vez que for executada.

A linha 7 define o início da nossa estrutura de repetição que termina na linha 21. Note que a condição de repetição é que o valor da variável **resposta** seja

igual a **"S".** Assim, o trecho que vai da linha 8 à linha 20 se repetirá enquanto o valor da variável **resposta** for "**S**".4

O trecho que vai da linha 8 à linha 18 nós já conhecemos, pois o utilizamos em vários outros algoritmos. É nesse trecho que solicitamos a digitação das duas notas do aluno, calculamos a média e imprimimos se ele foi aprovado ou não.

- Na linha 19 perguntamos ao usuário se ele deseja digitar as notas de outro aluno ou não. Esperamos que, caso deseje digitar as notas de outro aluno, o usuário digite um "**S**" e, caso contrário, digite um "**N**". Na linha 20 a resposta digitada pelo usuário é armazenada na variável **resposta**.

Esperamos que o usuário digite "S" ou "N". Porém, no nosso algoritmo, a resposta para continuar tem que ser "S" – maiúsculo. Qualquer outro caractere encerrará o laço.

Na linha 21 é encerrado o bloco de repetição. Assim, o fluxo volta à linha 7 onde é testada a resposta dada pelo usuário. Se a resposta for "**S**", a condição de repetição será verdadeira e todo o bloco de repetição é executado novamente. Caso contrário, o laço é finalizado e a execução continua na linha 22, que determina o fim do programa.

Resumo

Nesta aula você conheceu dois tipos de estruturas de repetição em Portugol: o **para...faça** e o **enquanto...faça**. Aprendeu a avaliar quando utilizar cada um deles e colocou esse conhecimento em prática construindo algoritmos.

Atividades de aprendizagem

1. O algoritmo abaixo deve ficar lendo números inteiros até que o número zero seja informado. Quando o número zero for informado, o algoritmo deve exibir na tela a quantidade de números digitados (contando inclusive com o zero digitado) e a média dos valores digitados. Assim, complete as lacunas de forma que o algoritmo funcione corretamente.

```
Algoritmo exercicio1
var
  numero, quantidade, soma : inteiro
  media : real
inicio
  numero ← 1
  quantidade ← 0
```

```
  soma ← 0
  enquanto numero != ____  faca
      escreva("Digite um numero")
      leia(________)
      quantidade ← _____________ + 1
      soma ← soma + _____________
  fim enquanto
  media ← __________ / _____________
  escreva ("Quantidade de numeros digitados:",
_____________)
  escreva ("Media dos numeros digitados:",
_____________)
fim.
```

2. Faça um algoritmo que solicite a digitação da idade e do sexo de uma pessoa (o sexo deve ser F ou M) e depois pergunte se o usuário deseja informar uma nova pessoa. Esse processo deve se repetir até que o usuário informe que não deseja mais informar novas pessoas. Quando isso acontecer, o algoritmo deve imprimir na tela a quantidade de pessoas do sexo masculino informadas; a quantidade de pessoas do sexo feminino informadas; a média das idades informadas para pessoas de sexo masculino; e a média das idades informadas para pessoas de sexo feminino.

3. Faça um programa que fique em laço solicitando a digitação do estado civil (S para solteiro, C para Casado, V para viúvo ou D para Divorciado) e da idade de pessoas. O programa só deve parar de solicitar a digitação de dados de novas pessoas quando for informado um estado civil inválido (diferente de S, C, V e D) ou uma idade inválida (idade menor que 0). Quando isso acontecer, devem ser exibidas as quantidades de pessoas de cada estado civil e a respectiva média de idade. A condição de repetição deste laço é um pouco mais complexa: você deverá montar uma expressão lógica usando OU ou E (reveja o item 3.1.3).

4. Foi dado como exemplo um algoritmo que solicita duas notas de alunos e indica se cada um deles foi aprovado ou reprovado até que o usuário responda que não deseja mais informar notas de alunos. Altere aquele algoritmo de forma que, ao final da execução, informe a quantidade total de alunos analisados, a quantidade de alunos aprovados e a quantidade de alunos reprovados.

Aula 5 – Introdução à linguagem C

Objetivos

Aprender os conceitos de linguagem de programação e compilador.

Instalar e conhecer a ferramenta de programação que será utilizada no curso.

Aprender os conceitos básicos da linguagem C, bem como sua sintaxe.

Conhecer os comandos de atribuição, entrada e saída de dados da linguagem C.

Criar programas em linguagem C.

5.1 Conceitos básicos

No capítulo anterior, construímos nossos algoritmos utilizando uma linguagem conhecida como Portugol. Esta linguagem é muito utilizada para iniciar o ensino de programação por ter regras formais e rígidas como uma linguagem de programação e, ao mesmo tempo, ser muito parecida com a linguagem natural humana.

Mas, quando queremos construir algoritmos que computadores possam entender e executar, é necessário que utilizemos uma **linguagem de programação** que disponha de um **compilador** que transforme o algoritmo em um programa a ser executado.

O arquivo contendo o algoritmo que desenvolvemos é chamado de "fonte", pois é a partir dele que o compilador vai criar o **programa** a ser executado.

Em nosso curso foi escolhida a linguagem C. Para compilar e executar nossos programas utilizaremos o ambiente Bloodshed Dev-C++, disponível gratuitamente no *link* **http://superdownloads.uol.com.br/download/199/bloodshed**-dev-c/, ou no ambiente Moodle do nosso curso.

A-Z

Linguagem de programação
É um método padronizado para expressar instruções para um computador (LAUREANO, 2005, p. 4).

Compilador
É um programa que traduz algoritmos construídos em uma determinada linguagem de programação para arquivos em linguagem de máquina, ou seja, possíveis de serem executados em computadores.

Programa
É uma coleção de instruções que descrevem uma tarefa a ser realizada por um computador (LAUREANO, 2005, p. 4).

5.2 Conhecendo o Bloodshed DEV-C++

5.2.1 Primeiro passo – Janela 1

Assim que entrarmos no ambiente Dev-C++, a tela da Figura 5.1 será a primeira a que teremos acesso.

Clique no botão <Fechar> da janela "Dica do dia".

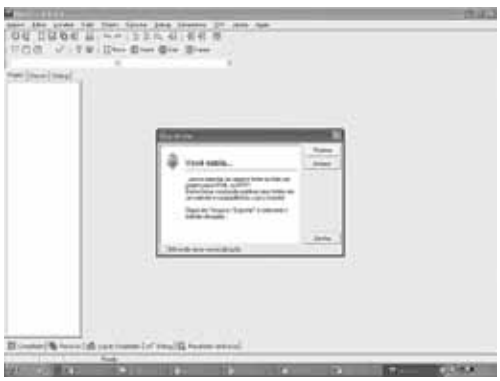

Figura 5.1: Tela inicial

5.2.2 Segundo passo – Janela 2

Clique no menu Arquivo>Novo>Arquivo Fonte, como apresentado na Figura 5.2.

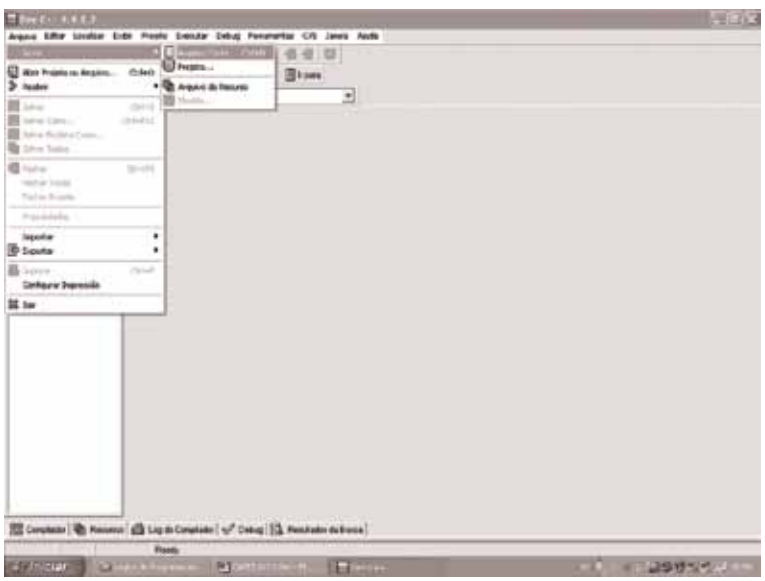

Figura 5.2: Apresentação do menu para criação de novo arquivo

5.2.3 Terceiro passo – Janela 3

Será aberta uma janela como a exibida na Figura 5.3. Digitaremos nossos algoritmos na área branca do lado direito dessa janela, onde aparece o cursor e uma linha destacada em azul.

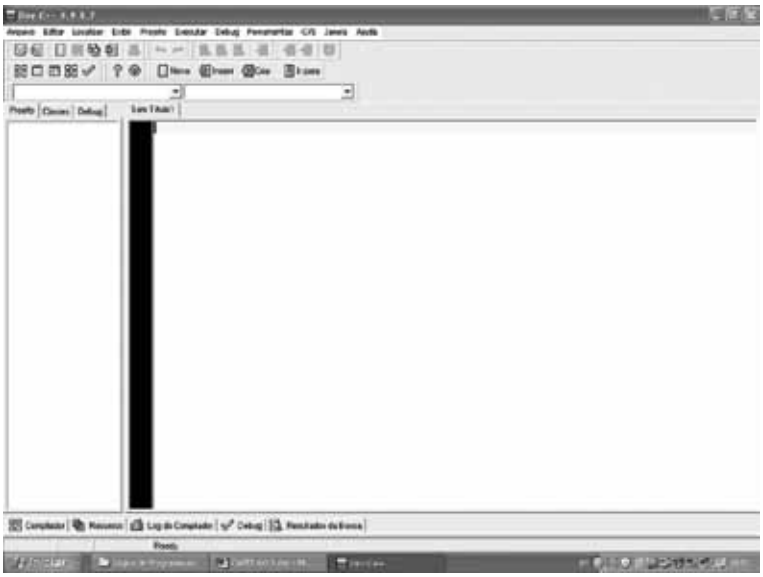

Figura 5.3: Apresentação da área de trabalho

5.3 Visão geral da linguagem C e do Dev-C++

Para termos uma visão geral da linguagem que usaremos no desenvolvimento dos programas, vamos analisar como ficaria, na linguagem C, nosso algoritmo **Multiplicacao** apresentado como exemplo na aula 2.

Quadro 5.1: Comparação Portugol & linguagem C

Linguagem – Portugol	Linguagem – C
linha 1 ... Algoritmo multiplicacao	linha 1 ...#include <stdio.h>
linha 2 .. var	inha 2 ... #include <stdlib.h>
linha 3 .. num1, num2, mult:inteiro	linha 3 ...int main ()
linha 4 .. inicio	linha 4 ...{
linha 5 ... escreva ("Digite o primeiro número")	linha 5 ...int num1, num2, mult;
linha 6 ... leia (num1)	linha 6 ...printf ("Digite o primeiro numero: ");
linha 7 ...escreva ("Digite o segundo número")	linha 7 ...scanf ("%d", &num1);
linha 8 ... leia (num2)	linha 8 ... printf ("Digite o segundo numero: ");
linha 9 ... mult ← num1 * num2	linha 9 ... scanf ("%d", &num2);
linha 10 ... escreva ("A multiplicação é:", mult)	linha 10..mult = num1 * num2;
linha 11 .fim	linha 11.. printf ("A multiplicacao e: %d\n",mult);
	linha 12..system("PAUSE");
	linha 13..return(0);
	linha 14..}

A seguir cada linha da função **Multiplicacao** em linguagem C é explicada, aproveitando para comentar sobre os fundamentos básicos da linguagem C:

- **A primeira e a segunda linha** – #include <stdio.h> #include <stdlib.h>

As duas linhas indicam a inclusão de bibliotecas que possuem as funções de entrada e saída de dados necessários à execução do nosso programa **Multiplicacao**. Veremos mais adiante que outras bibliotecas serão necessárias. Quando isso acontecer, vamos incorporá-las. Para evitar problemas, **sempre** inicie seus programas com essas duas linhas.

- **A terceira linha** - int main( )

A função main( ) é sempre a primeira a ser executada no programa C. Em todo programa desenvolvido em C, existirá uma função main( ).

- **A quarta linha** - {

É o início de um bloco de comandos no programa. Para toda chave { que inicia um bloco de comandos, teremos uma chave } que será responsável por informar o fechamento desse bloco.

- **A quinta linha** - int num1, num2, soma;

Foram declaradas três variáveis necessárias à execução do programa. Iniciamos a declaração informando que as variáveis seriam do tipo inteiro (na linguagem C o tipo inteiro é chamado de ***int***). Note que em linguagem C não há a palavra **var** indicando o bloco de declaração de variáveis. Preste atenção também à identação do código! Como essa linha está dentro do bloco iniciado pela chave ({) da linha anterior, ela tem um recuo em relação à mesma.
Observe a existência de um ponto-e-vírgula ";". Seu emprego indica o final do comando. **Toda instrução em C é finalizada por um ponto-e-vírgula.**

- **A sexta linha** - printf("Digite o primeiro numero:");

A função *printf*( ) é uma função de **saída de dados**. Permite que uma mensagem seja exibida na tela do computador (equivalente ao **escreva**() de Portugol). As mensagens devem ser escritas entre aspas.

- **A sétima linha -** scanf ("%d", &num1);

A função *scanf*( ) é responsável por **ler os dados** que forem digitados pelo teclado (equivalente ao **leia**() de Portugol). Nessa linha a função lerá o primeiro número que for digitado e o armazenará no endereço da variável num1, conforme indicado ("%d", &num1). O "%d" indica que se trata da leitura de um número inteiro. Para ler dados de outros tipos serão utilizados outros códigos, conforme veremos mais à frente.

- **A décima linha -** mult = num1 * num2;

O comando de atribuição (=) atribui à variável mult o resultado da multiplicação dos valores contidos nos endereços de num1 e num2. **É importante notar que o comando de atribuição que em portugol era representado por uma seta, em C é representado pelo sinal de igual (=).**

- **A décima primeira linha -** printf("A soma e: %d \n", soma);

Já vimos que a função *printf*( ) permite a exibição de mensagens no monitor. Porém, nesse comando, o conteúdo da variável *soma* também é exibido. Isso é possível porque incluímos na mensagem o código para impressão de variáveis do tipo inteiro: %d. O código especial \n é responsável por fazer saltar uma linha. Mais à frente aprofundaremos os estudos da função *printf*().

- **A décima segunda linha -** system("PAUSE");

Possibilita uma pausa no programa a fim de visualizarmos o resultado na tela. Caso contrário, ele seria exibido tão rapidamente que não conseguiríamos vê-lo.

- **A décima terceira linha -** return (0);

Indica o número inteiro que está sendo retornado pela função; em nosso caso, o número zero. O comando return (0) será detalhado adiante.

- **A décima quarta linha -** }

Indica o fim do programa. O fim de main( ). Essa chave está fechando o bloco aberto na linha 4.

Um detalhe importante sobre a linguagem C é que, ao contrário de algumas outras linguagens, em C **há distinção entre caracteres maiúsculos e minúsculos.** Assim, em C, é diferente chamar uma variável de **num** ou **Num.** Portanto, para evitar erros, por padrão, costumamos utilizar apenas caracteres minúsculos nos nomes de variáveis.

Observe também que todos os comandos da linguagem C são escritos apenas **com caracteres minúsculos.**

Agora que compreendemos cada linha do nosso primeiro programa em C, vamos abrir o ambiente Dev-C++ seguindo os passos apresentados no início do capítulo e, então, digitar esse programa no ambiente.

Para salvar o nosso arquivo fonte, devemos acessar o menu Arquivo > Salvar conforme exibido na Figura 5.4.

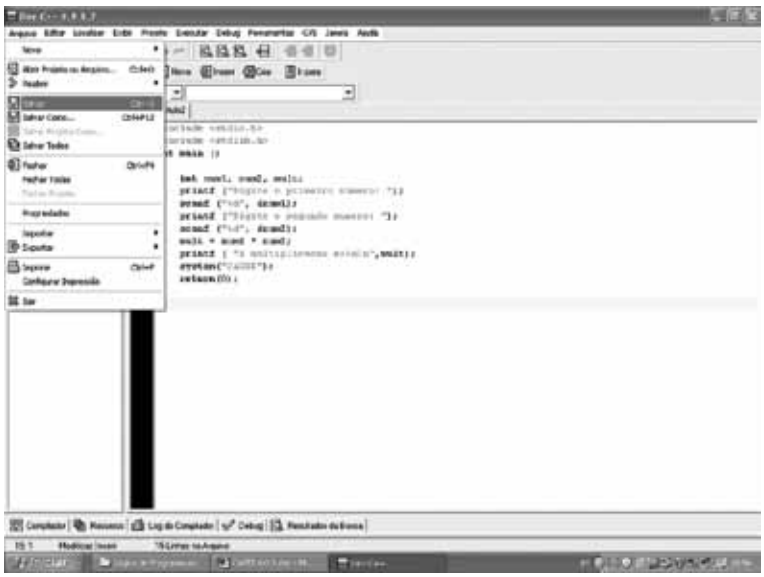

Figura 5.4: Apresentação do menu para salvar arquivo

Então será exibida a janela "Salvar Arquivo". Nessa janela deve ser informado o nome para o arquivo e indicado o tipo do arquivo. No nosso caso, devemos salvar como arquivos fontes de C (C *source files*). Essa janela com suas opções é exibida na Figura 5.5.

Figura 5.5: Apresentação da janela "salvar arquivo"

Uma boa prática é salvar periodicamente o arquivo, ou seja, não espere finalizar toda a digitação para então salvar. Assim, caso ocorra algum problema, você não perderá todo o trabalho.

Note que a janela para nomear o arquivo só aparece na primeira vez em que o mesmo é salvo. Nas demais vezes o arquivo será apenas atualizado, não sendo necessário informar novamente seu nome e tipo.

Depois de salvar o arquivo, devemos compilar e executar o programa a fim de visualizarmos seu resultado. Para compilar e executar o programa, podemos utilizar a tecla **F9** ou acessar o menu Executar > Compilar & Executar. Caso você solicite a compilação antes de salvar o arquivo, automaticamente aparecerá a janela de Salvar arquivo para depois o ambiente compilar seu programa. Neste caso, siga as instruções dadas anteriormente para salvar arquivo. O resultado da compilação e execução deste programa é exibido na Figura 5.6. No exemplo apresentado nesta figura, o usuário digitou 2 para o valor do primeiro número e 3 para o segundo número.

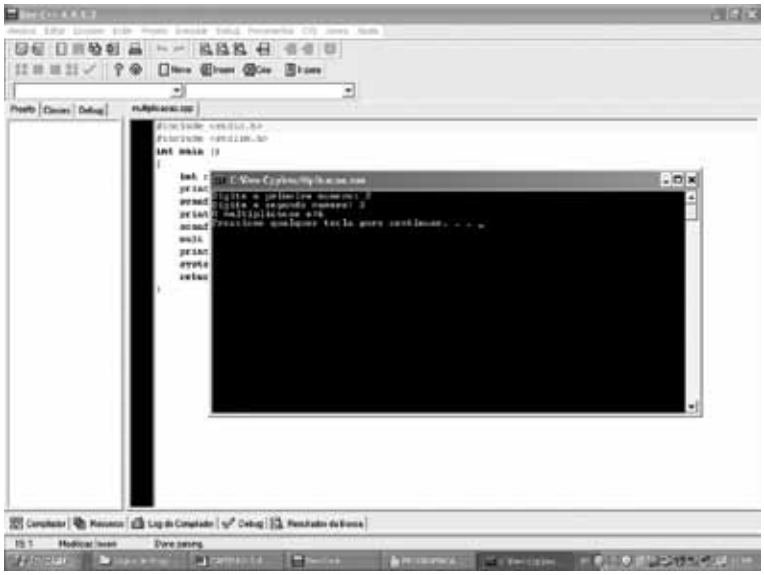

Figura 5.6: Apresentação do resultado da compilação e execução do programa multiplicacao

Quando compilamos um programa e o ambiente encontra algum erro no mesmo, a linha que contém o erro fica sombreada em destaque e, na parte inferior da janela do ambiente, são exibidas mensagens indicando o erro encontrado. Essas mensagens são muito úteis para que possamos compreender o motivo do erro e corrigi-lo. É muito importante ficar atento a tais mensagens. A Figura 5.7 exibe a tela do ambiente ao tentar executar o programa de multiplicação com um erro.

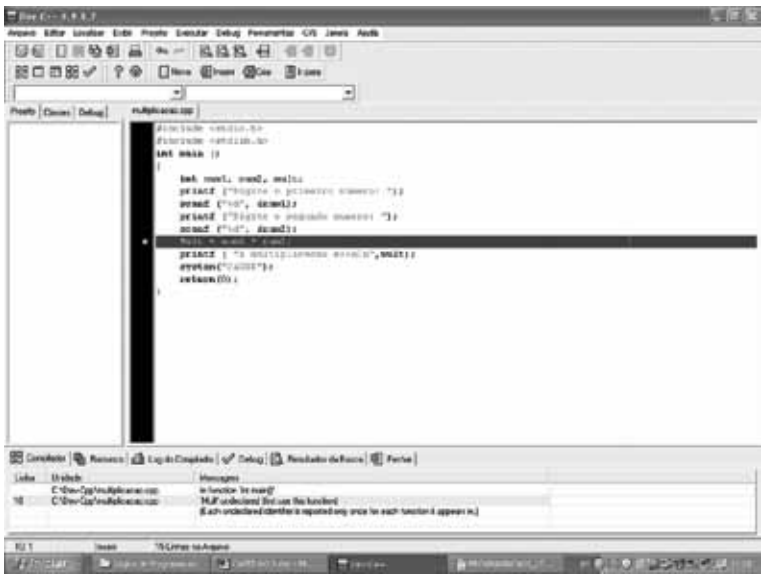

Figura 5.7: Apresentação de erro de compilação

No caso apresentado pela Figura 5.7, o erro está no fato de **mult** ter sido declarada com letras minúsculas, mas ao utilizar a variável, colocamos seu

nome com a primeira letra maiúscula. Assim, o ambiente está exibindo uma mensagem de erro na linha 10 que diz: "'**Mult**' *undeclared (first use in this function)*". Ou seja, ele está dizendo que a variável "**Mult**" não foi declarada e que a primeira vez que ela está sendo usada é nessa linha 10.

Atividade 5.1 Utilizando o ambiente Dev-C++ digite, salve, compile e execute o exem- plo do programa de multiplicação conforme apresentado nesta seção.

Atividade 5.2 Classifique as afirmativas como verdadeiras ou falsas:

a) Toda instrução em C é terminada por um ponto-e-vírgula.( )

b) Em C não há diferenciação entre letras maiúsculas e minúsculas.( )

c) Todo programa C deve ter uma função main(). Esta é a primeira função do programa a ser executada. ( )

As explicações dadas na sequência do conteúdo serão acompanhadas de exemplos que você deverá digitar, compilar e executar no DEV-C++.

Depois de executá-los, o código fonte deverá ser analisado e entendido.

A fim de facilitar o estudo, mesmo longe do computador, a partir daqui duas telas serão sempre apresentadas abaixo do exemplo. São elas:

- A tela branca, que contém o código do programa citado como exemplo, devidamente digitado no DEV-C++.

- A tela preta, que é o resultado da compilação e da execução.

Todos os programas desenvolvidos nas atividades também deverão ser digitados, compilados e executados no DEV-C++.
Não avance se as dúvidas permanecerem.
Bom estudo!!

5.4 Variáveis em linguagem C

Já aprendemos que constantes e variáveis alocam espaço em memória e são utilizadas para armazenar valores necessários à execução do programa. A diferença entre constantes e variáveis está no fato de que o valor de uma constante nunca se altera enquanto o valor de uma variável pode mudar

durante a execução do programa.

Assim como em Portugol, para declarar variáveis em C é necessário definir um nome e o tipo da variável. Porém em C indicamos o tipo da variável e, em seguida, o nome da mesma. A quinta linha do exemplo apresentado no Quadro 5.1 exibe a declaração de 3 variáveis do tipo inteiro. A seguir são apresentados outros exemplos.

```
int idade;                    //declaração da variável idade do tipo inteiro

float salario, desconto;      //declaração das variáveis salário e desconto
do tipo real
```

O tipo da variável define, além do tipo de dado que ela pode armazenar, o tamanho do espaço de memória que deve ser alocado para a mesma. O tamanho do espaço de memória é medido em uma unidade chamada *byte*. O Quadro 5.2 traz três tipos de variáveis existentes em C, informando para cada um o tipo de dados que pode ser armazenado (fazendo uma comparação com o Portugol) e o tamanho do espaço de memória reservado (considerando arquiteturas de 32 *bits*).

Quadro 5.2: Tipos de variáveis da linguagem C

Tipo de variável em C	Valores a serem armazenados	Tamanho em *bytes*
char	Permite armazenar um **caractere** alfabético. Equivalente ao tipo caractere de Portugol.	1
int	Permite armazenar números inteiros positivos ou negativos. Equivalente ao tipo **inteiro** de Portugol.	4
float	Permite armazenar valores numéricos reais, ou seja, números com ponto decimal. Equivalente ao tipo **real** de Portugol.	4

Quanto aos nomes de variáveis, valem as mesmas regras apresentadas quando estudamos Portugol, ou seja, o primeiro caractere do nome deve ser uma letra e os demais podem ser letras, números ou o caractere *underline* (_).

Vale lembrar que em C há distinção entre caracteres maiúsculos e minúsculos (rever exemplo na Figura 5.7). Assim, para evitar erros desse tipo, aconselhamos evitar o uso de caracteres maiúsculos nos nomes das variáveis, apesar de seu uso ser permitido.

Após declarar uma variável, pode-se atribuir um valor a ela através da utilização do comando de atribuição **igual**(=). Em C, a atribuição pode ser feita em qualquer ponto do programa após a criação da variável, mas também é

permitido fazer uma atribuição na mesma linha em que é feita a declaração.

Os valores atribuídos a variáveis do tipo char devem estar entre aspas simples. Também é importante ressaltar que o separador decimal utilizado em variáveis do tipo *float* é o ponto (.) e não a vírgula (,) como se costuma utilizar no Brasil. Assim, se queremos atribuir a uma variável o valor 552,35 devemos utilizar 552.35. Veja os exemplos:

float salario = 552.35; //a variável **salario** foi declarada e recebeu o valor 552,35

char sexo = 'F'; /*a variável **sexo** do tipo *char* foi declarada e recebeu o valor F (note as aspas simples)*/

salario = 1625.23; /*a variável **salario** recebeu o valor 1.625,23 (note que o separador de decimais é o ponto "." e não a vírgula como utilizamos naturalmente no Brasil!*/

5.5 Comando de saída de dados – *printf*()

Como vimos em nosso exemplo do programa multiplicacao, a função *printf* é a função de saída de dados em C. O *printf* funciona em C como a função **escreva** em Portugol, ou seja, é através dessa função que imprimimos mensagens na tela.

Ainda em nosso exemplo anterior vimos que a função *printf( )* usa o caractere de percentual (%) seguido de uma letra para identificar o formato de impressão. Naquele exemplo utilizamos o %d, pois estávamos imprimindo um número inteiro. No Quadro 5.3 são exibidos os principais códigos de formatação utilizados no *printf()*.

Quadro 5.3: Códigos de formatação *printf()*

CÓDIGO	SIGNIFICADO
%c	Usado quando a função for exibir apenas um caractere (tipo *char*).
%f	Usado quando a função for exibir número com ponto flutuante (tipo *float*). Exemplo: 1.80
%s	Usado quando a função for exibir uma cadeia de caracteres, ou seja, uma ou várias palavras.
%d	Usado quando a função for exibir um número inteiro (tipo *int*).

A Figura 5.8 exibe um programa em C exemplificando o uso de *printf()* com vários tipos de dados.

```c
#include <stdio.h>
#include <stdlib.h>
   int main( )
 {
   int matricula;
   float media_final;
   char turma;
   turma='A';
   matricula= 12;
   media_final=85.0;
   printf ("O aluno matricula = %d \n", matricula);
   printf ("Turma = %c \n", turma);
   printf ("Ficou com media = %f \n\n", media_final);
   system("pause");
   return(0);
 }
```

Figura 5.8 – Exemplo de programa em C

A Figura 5.9 apresenta o resultado da execução desse programa.

Figura 5.9: Resultado da execução do programa que exemplifica o uso de *printf()*

Nesse exemplo, utilizamos o %d quando imprimimos a variável **matricula**, que é do tipo *int*; %c para a variável **turma**, que é do tipo *char* e %f para a variável **media_final**, do tipo *float*. Os caracteres \n que aparecem no final de cada printf são utilizados para pular uma linha; caso não tivéssemos utilizado \n, todas as mensagens seriam impressas na mesma linha.

5.6 Comando de entrada de dados *scanf()*

No exemplo do programa **multiplicacao** também pudemos observar a utilização do comando *scanf( )*. O *scanf( )* funciona em C como a função **leia** em Portugol, ou seja, é através desta função que lemos entradas de dados através do teclado.

A exemplo do *printf()*, o *scanf()* também utiliza os códigos de formatação. Enquanto no *printf()* esses códigos eram utilizados para indicar o formato dos dados a serem escritos, no *scanf()* esses mesmos códigos indicam o formato dos dados a serem lidos. O Quadro 5.4 exibe os códigos de formatação utilizados no *scanf()*. Note a semelhança com os códigos do *printf()*.

Quadro 5.4: Códigos de formato de leitura da função *scanf()*	
CÓDIGO	**FUNÇÃO**
%c	Usado quando a função for armazenar um caractere (tipo char).
%f	Usado quando a função for armazenar um número com ponto flutuante, aquele valor com vírgula (tipo *float*).
%s	Usado quando a função for armazenar uma cadeia de caracteres, ou seja, uma ou várias palavras.
%d	Usado quando a função for armazenar um número inteiro (tipo int).

5.7 Comentários

Quando desenvolvemos programas, devemos colocar textos que expliquem o raciocínio seguido durante seu desenvolvimento para que outras pessoas, ou nós mesmos, ao ler o programa mais tarde, não tenhamos dificuldades em entender sua lógica. Esses textos são chamados de comentários.

Os comentários podem aparecer em qualquer lugar do programa. Em C, há dois tipos de comentários: os comentários de linha e os comentários de bloco.

Os comentários de linha são identificados pelo uso de //. Assim, quando usamos // em uma linha, tudo o que estiver nessa linha depois do // são considerados comentários.

Os comentários de bloco são iniciados por /* e finalizados por */. Tudo o que estiver entre esses dois símbolos são considerados comentários. Os comentários de bloco podem ocupar várias linhas.

Veja o exemplo da Figura 5.10 a seguir.

```
#include <stdio.h>
#include <stdlib.h>
int main( )
{
  int matricula= 2233;  /* podemos escrever cometários desta forma */
  float media_final=80.5; // ou apenas com duas barras no início do comentário
  char discipl[10]="Prog I";// a var discipl[10], pode armazenar até 10 caracteres
  printf ("O aluno matrícula = %d \n", matricula);
  printf ("Disciplina = %s \n", discipl);
  printf ("Ficou com media = %f \n\n", media_final);
  system("pause");
  return(0);
}
```

Figura 5.10: Exemplo de código em C com comentários

A Figura 5.11 mostra a execução do programa acima. Note que o comentário só aparece no código fonte, não influenciando na execução do programa.

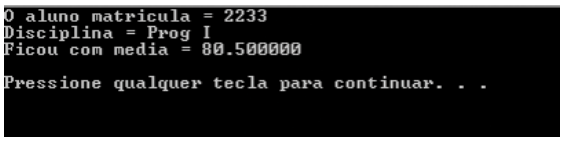

Figura 5.11: Execução do exemplo de uso de comentários em C

5.8 Expressões aritméticas

Como estudamos na aula 2, os operadores aritméticos são símbolos que representam operações aritméticas, ou seja, as operações matemáticas básicas. A maior parte dos operadores aritméticos de C são os mesmos que vimos em Portugol. Conforme podemos ver no Quadro 5.5, apenas acrescentamos o incremento unário (++) e o decremento unário (--):

Quadro 5.5: Operadores aritméticos da linguagem C

Operador	Operação matemática
+	Adição
-	Subtração
*	Multiplicação
/	Divisão
--	**Decremento Unário**
++	**Incremento Unário**
%	Resto da Divisão Inteira

Para aprofundar seus conhecimentos, recomendo a leitura dos capítulos 1 e 2 do livro **Treinamento em Linguagem C – Curso Completo – Módulo 1**, de Victorine Mizrahi ou os capítulos 1 e 2 do livro **C Completo e Total,** do autor Herbert Schildt.

O operador de incremento unário (++) incrementa de **1** o seu operando. Ou seja, se eu quiser aumentar em **1** o valor de uma variável **x**, posso fazer **x=x+1**; ou escrever simplesmente **x++**;. De forma análoga, o operador de decremento unário (--) decrementa de **1** o seu operando. Ou seja, se eu quiser diminuir de **1** o valor de uma variável **x**, posso fazer **x=x-1**; ou escrever simplesmente **x--**;.

Devemos evitar a utilização de operadores unários em expressões aritméticas, pois seu uso pode dificultar o entendimento da expressão. Assim, recomendo a utilização desses operadores apenas em ocasiões em que se deseja apenas incrementar ou decrementar o operando; nunca utilizá-los em meio a expressões.

A ordem de precedência entre os operadores em expressões aritméticas é a mesma já estudada, ou seja, primeiro as multiplicações e divisões e só depois as somas e subtrações. Em C também podemos utilizar os parênteses em expressões aritméticas, como fizemos em Portugol.

Resumo

Nesta aula você aprendeu os conceitos de linguagem de programação e de compilador, aprendeu os conceitos básicos da linguagem C relacionando-os ao Portugol e conheceu a ferramenta de programação Dev-C++. Para praticar, você implementou seus primeiros programas em C.

Atividades de aprendizagem

1. Faça um programa que:

a) peça ao usuário para digitar um número inteiro;

b) armazene esse número numa variável chamada num1;

c) peça ao usuário para digitar outro número inteiro;

d) armazene esse número numa variável chamada num2;

e) some os valores e guarde o resultado numa variável chamada soma;

f) exiba os valores digitados;

g) exiba o resultado da soma.

Obs.: **Lembre-se de comentar seu código! Caso encontre dificuldades para fazer esse programa, consulte o exemplo Multiplicacao.**

2. Transforme para linguagem C os algoritmos desenvolvidos em Portugol nos exercícios 4, 5 e 6 (aula 2).

Aula 6 – Estruturas de decisão em linguagem C

Objetivos

Conhecer os operadores lógicos e relacionais da linguagem C.

Conhecer a estrutura de decisão *if...else*.

Conhecer a estrutura de decisão *switch*.

Criar programas em linguagem C utilizando estruturas de decisão.

6.1 Expressões lógicas

Como já estudamos na aula 3, as expressões lógicas são expressões formadas a partir do uso de variáveis e constantes, operadores relacionais e operadores lógicos. As expressões lógicas são avaliadas e retornam sempre um valor lógico: **verdadeiro** ou **falso**.

A teoria sobre operadores lógicos, operadores relacionais e Tabelas-verdade foi estudada na aula 2 e, por isso, não será repetida aqui. Dessa forma, vale a pena revisar tais conteúdos.

O Quadro 6.1 exibe a representação dos operadores lógicos em C.

Quadro 6.1: Operadores lógicos em linguagem C

Operador Lógico	Representação em C
E	&&
OU	|| (duas barras verticais)
NÃO	! (exclamação)

No Quadro 6.2 são listados os operadores relacionais em C.

Quadro 6.2: Operadores relacionais em linguagem C	
Descrição	**Símbolo**
igual a	== (dois sinais de igual)
maior que	>
menor que	<
maior ou igual a	>=
menor ou igual a	<=
diferente de	!=

Dentre os operadores relacionais, a única alteração que temos em C em relação ao que aprendemos em Portugol refere-se ao operador **igual a**. Em C, esse operador é representado por dois sinais de =, ou seja, por ==. Isso acontece para diferenciar o operador relacional (==) do comando de atribuição (=).

A Figura 6.1 exibe um exemplo de utilização do operador relacional ==.

```
#include <stdio.h>
#include <stdlib.h>
int main ()
{
    int num1;
    num1 = 5; //Atribui o valor 5 à variável num1
    if (num1 == 5) //comparei o valor de num1 com 5. Note que utilizei ==
       printf("num1 vale 5 \n");
    system("PAUSE");
    return(0);
}
```

Figura 6.1 - Exemplo de utilização do operador relacional ==

O exemplo apresentado na Figura 6.1 é bem simples. É declarada uma variável com nome **num1**, atribuído o valor **5** a ela e depois utilizamos uma estrutura de decisão para verificar se o valor dessa variável é igual a **5**. Obviamente o teste será verdadeiro e a linha de *printf()* que se encontra dentro do bloco de decisão será executada. Ainda nessa aula vamos estudar a estrutura de decisão *if...else* utilizada nesse exemplo. A Figura 6.2 exibe o resultado da execução desse programa.

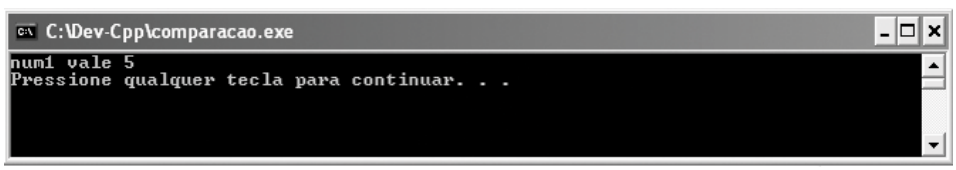

Figura 6.2: Resultado da execução do programa da Figura 6.1

6.2 Estruturas de decisão

Como vimos na aula 3, ao desenvolver programas deparamos com situações nas quais o fluxo de execução do programa depende de determinadas condições, ou seja, parte do nosso programa só é executada se a condição para essa execução for verdadeira. Para isso existem os comandos de seleção ou decisão.

Em Portugol estudamos a estrutura de decisão **se...então...senão**. Para realizar essa tomada de decisão na linguagem C, temos os comandos de seleção if e *switch*.

6.2.1 Comando *if*

O comando *if* deve ser utilizado quando a execução de uma ou mais instruções do programa depender de uma ou mais condições. O comando *if* é equivalente ao comando **se...então** do Portugol.

Sintaxe: if **(expressão lógica)**
instrução;

Como funciona?

Se a **expressão lógica** que se encontra entre os parênteses for verdadeira, a instrução da linha subsequente será executada; caso contrário, não será.

No caso de termos mais de uma instrução que dependa do resultado da condição para ser executada, essas instruções devem ficar entre chaves, conforme exibido na sintaxe abaixo:

if **(expressão lógica)**

```
{
      instrução 1;
. . .
      instrução n;
}
```

A Figura 6.3 exibe um exemplo em que o programa solicita a digitação de dois números pelo usuário, soma os dois valores digitados e verifica o resultado. Caso o resultado da soma seja maior que 2, é exibida uma mensagem; caso contrário, nada é feito.

```
#include <stdio.h>
#include <stdlib.h>

main()
{
        int n1, n2, s;
        printf("Digite primero numero");
        scanf("%d", &n1);
        printf("Digite segundo numero");
        scanf("%d", &n2);
        s=n1 + n2;
        if(s>2)
                printf("\O resultado da soma dos valores digitados é maior
que dois: %d \n", s);
        system("pause");

}
```

Figura 6.3: Exemplo de utilização do comando *if*

A Figura 6.4 mostra a execução do programa em um caso em que a soma dos números é maior que 2. O programa solicitou a digitação de dois números, o usuário digitou **1** para o primeiro número e **3** para o segundo. Como a soma entre os dois foi maior que **2**, o *printf()* é executado, exibindo a mensagem.

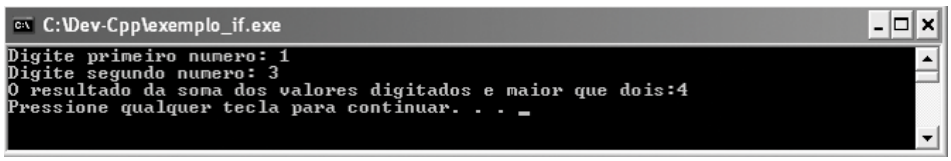

Figura 6.4: Execução do programa para uma soma maior que 2

Já no exemplo exibido pela Figura 6.5, o resultado da soma dos números digitados não é maior que 2; nenhuma mensagem é exibida. A linha "Pressione qualquer tecla para continuar..." é exibida porque utilizamos em nosso programa o comando *system*("*pause*").

Figura 6.5: Execução do programa para uma soma menor que 2

É importante que você digite, compile e execute os exemplos vistos, conforme orientação dada no início desta aula.

Para montar a expressão lógica das atividades 6.2 e 6.3, você deverá combinar expressões usando || ou && (OU ou E – reveja o item 3.1.3 e Quadro 6.1).

Atividade 6.1- Desenvolva um programa que leia a matrícula e a nota final de um aluno de uma escola. Se a nota final for maior ou igual a 60, o programa deve exibir a mensagem "Aluno aprovado".

Atividade 6.2 - Faça um programa que leia o sexo do usuário e apresente a mensagem "O sexo é válido", se o caractere digitado for 'M' ou 'F'.

Atividade 6.3 - Faça um programa que leia um número dado como entrada e apresente a mensagem "O número está na faixa de 20 a 90" se o valor fornecido estiver entre 20 e 90.

Atividade 6.4 - Faça um programa que leia o valor do salário bruto de um funcionário. Se o salário for menor ou igual a R$ 500,00, o programa deve aplicar um aumento de 0.10 (10%).

6.2.2 Comando *if...else*

Como vimos, o comando *if* deve ser utilizado em situações nas quais um bloco de instruções só deve ser executado se uma determinada situação for verdadeira. Mas, muitas vezes nos deparamos com situações nas quais o programa deve seguir um fluxo caso uma determinada condição seja verdadeira; e outro fluxo caso essa condição seja falsa. Nessas situações, devemos utilizar a estrutura *if...else*. O comando *if...else* é equivalente ao comando **se...então...senão** de Portugol, estudado na terceira aula.

Sintaxe:
if **(expressão lógica)**
{
    <bloco de instruções a ser executado caso a expressão seja verdadeira>
}

else

{
    <bloco de instruções a ser executado caso a expressão seja falsa>
}

Como funciona?

Se a **expressão lógica** que se encontra entre os parênteses for verdadeira, o bloco de instruções logo abaixo do *if* será executado. Caso contrário, o bloco de instruções do *else* é que será executado.

Caso tenhamos apenas uma instrução no bloco do *if* ou no bloco do *else*, as chaves poderão ser omitidas.

Por exemplo, a Figura 6.3 apresentou um programa que exibe uma mensagem caso a soma de dois números seja maior que **2**. Caso a soma não atenda a essa condição, nenhuma ação é executada pelo programa. Agora, vamos alterar aquele exemplo, utilizando o *if...else*. Em nosso novo exemplo, vamos efetuar a soma e, caso a soma seja maior que **2**, será exibida uma mensagem informando isso. Caso contrário, será exibida uma mensagem informando que a soma não é maior que **2**. Ou seja, vamos apenas acrescentar uma cláusula *else* ao nosso exemplo anterior. O novo exemplo é apresentado na Figura 6.6.

```
#include <stdio.h>
#include <stdlib.h>
int main ()
{
    int num1, num2, s;
    printf ("Digite primeiro numero: ");
    scanf ("%d", &num1);
    printf ("Digite segundo numero: ");
    scanf ("%d", &num2);
    s = num1 + num2;
    if (s>2)
        printf ( "O resultado da soma dos valores digitados e maior que dois:%d\n",s);
    else
        printf ( "O resultado da soma dos valores digitados nao e maior que dois:%d\n",s);
    system("PAUSE");
    return(0);
}
```

Figura 6.6: Exemplo do comando *if...else*

A Figura 6.7 exibe o resultado desse programa em um caso onde a soma dos dois números digitados pelo usuário é maior que 2, enquanto a Figura 6.8 exibe o resultado da execução quando essa soma não é maior que 2.

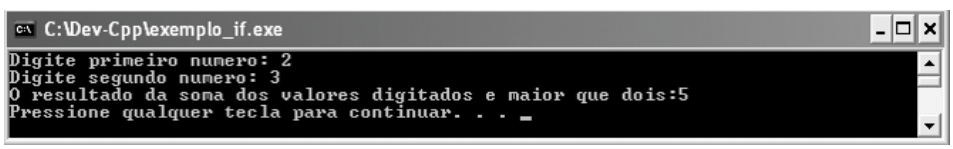

Figura 6.7: Execução do programa para uma soma maior que 2

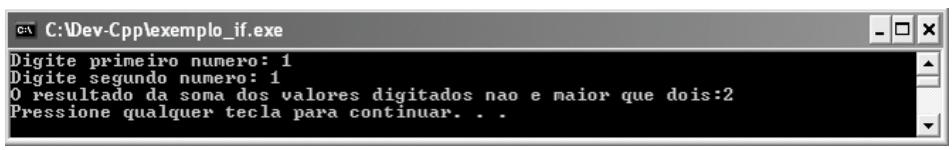

Figura 6.8: Execução do programa para uma soma menor ou igual a 2

Vale ressaltar que, no exemplo apresentado na Figura 6.6, apenas uma instrução é executada tanto no bloco do *if* quanto no bloco do *else*. Por isso, não foi necessário o uso das chaves **{ }**. Caso tivéssemos mais de uma instrução em algum desses blocos, o uso das chaves seria obrigatório!

A Figura 6.9 apresenta um exemplo no qual o uso das chaves foi necessário. Nesse exemplo, uma empresa dará um aumento para os funcionários de acordo com o salário atual de cada um. Caso o funcionário receba até R$1.500,00, ele terá um aumento de 10%. Caso o salário seja maior que R$1.500,00, o aumento será de 8%. Assim, o programa solicita a digitação do salário e, de acordo com o valor atual, calcula o novo valor e exibe uma mensagem.

```
#include <stdio.h>
#include <stdlib.h>
int main ()
{
    float salario_atual, novo_salario;
    printf ("Digite o salario atual: ");
    scanf ("%f", &salario_atual);
    if (salario_atual<=1500)
    {
       novo_salario = salario_atual * 1.1;
       printf ( "Voce tera um aumento de 10%. Seu novo salaraio sera:%f\n",novo_salario);
    }
    else
    {
       novo_salario = salario_atual * 1.08;
       printf ( "Voce tera um aumento de 8%. Seu novo salaraio sera:%f\n",novo_salario);
    }
    system("PAUSE");
    return(0);
}
```

Figura 6.9: Exemplo do uso de chaves em comando *if...else*

A Figura 6.10 exibe o resultado desse programa em um caso onde o salário digitado foi de R$1320,00, ou seja, menor ou igual a R$1.500,00. Assim, o programa aplicou um reajuste de 10% e exibiu o novo salário.

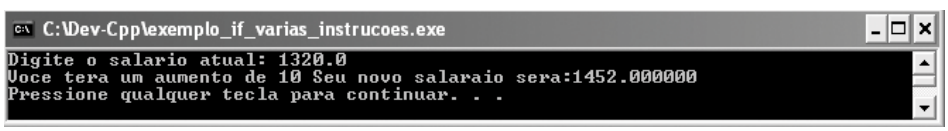

Figura 6.10: Resultado da execução para um salário de até R$1.500,00

Já na execução exibida pela Figura 6.11, o salário digitado foi de R$1.600,00.

Figura 6.11: Resultado da execução para um salário maior que R$1.500,00

Os primeiros exercícios desta lista são complementos aos exercícios da lista anterior. Assim, utilize as soluções da lista anterior como ponto de partida para esta.

Atividade 6.5 - Como complemento ao exercício 6.1, o programa deverá exibir também a mensagem "Aluno reprovado", caso a nota final do aluno seja menor que 60.

Atividade 6.6 - Como complemento ao exercício 6.2, o programa deverá exibir também a mensagem "Sexo inválido", se o caractere digitado for diferente de 'M' ou 'F'.

Atividade 6.7 - Como complemento ao exercício 6.3, o programa deverá exibir também a mensagem "O número está fora da faixa de 20 a 90", caso o valor fornecido não esteja entre 20 e 90.

Atividade 6.8 - Como complemento ao exercício 6.4, o programa deverá aplicar também um aumento de 0.05 (5%), se o salário for maior do que R$ 500,00.

Atividade 6.9 - Construa um programa que leia um número inteiro e imprima a informação se este número é ou não divisível por 5. Dica: Utilize o operador % (resto de divisão inteira).

6.2.3 Comandos *if...else* aninhados

Podemos aninhar construções *if...else*, em outras palavras, podemos colocar comandos *if...else* ou comandos *if* dentro de outros comandos *if...else*.

Veja o exemplo apresentado na Figura 6.12 a seguir.

```
#include <stdio.h>
#include <stdlib.h>
int main ()
{
    float salario_atual, novo_salario;
    char tem_filhos;
    printf ("O funcionario tem filhos? Digite s ou n.");
    scanf ("%c", &tem_filhos);
    printf ("Digite o salario atual: ");
    scanf ("%f", &salario_atual);
        if (salario_atual<=1500)
    {
       novo_salario = salario_atual * 1.1; //Se salario ate 1500 aumento de 10%
       if (tem_filhos == 's')
          novo_salario = novo_salario + 80;//Se salario até 1500 e tem filhos aumenta R$80,00
    }
    else
    {
       novo_salario = salario_atual * 1.08;//Se salario maior que 1500 aumento de 8%
       if (tem_filhos == 's')
          novo_salario = novo_salario + 50;//Se salario maior que 1500 e tem filhos aumenta R$50,00
    }
    printf ( "Seu novo salario sera:%f\n",novo_salario);
    system("PAUSE");
    return(0);
}
```

Figura 6.12: Exemplo de comandos *if...else* aninhados

Nesse exemplo, além do aumento percentual sobre os salários, os empregados também receberão uma ajuda caso tenham filhos. Os empregados com salários até R$1.500,00 receberão o aumento de 10% e, se tiverem filhos, receberão mais R$80,00. Já os funcionários com salários maiores que R$1.500,00 receberão o aumento de 8% e, se tiverem filhos, receberão mais R$50,00. Assim, além de informar o salário, deverá ser informado também se o funcionário tem filho ou não, digitando 's' para sim e 'n' para não. Note que foi acrescentada uma condição if dentro dos blocos if...else existentes para somar a gratificação no caso de ter filho. Note também que, nesse exemplo, utilizamos um printf único, fora das estruturas de condição que exibe o salário final.

A Figura 6.13 mostra o resultado da execução deste programa para um funcionário cujo salário é R$1.300,00 e que tem filhos. Foi dado um reajuste de 10% (R$130,00) mais a gratificação de R$80,00, chegando ao salário reajustado de R$1.510,00 exibido.

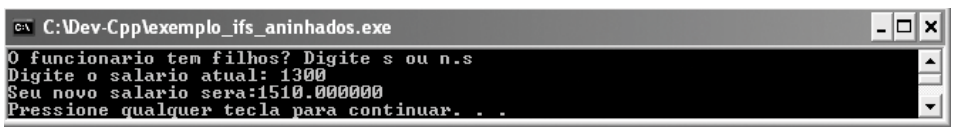

Figura 6.13: Resultado da execução para um salário de R$1.300,00 de quem tem filhos

Crie, compile e execute esse programa, testando outros valores de salário, variando a resposta à pergunta se tem ou não filhos.

Teste sempre seus programas com vários valores ou várias situações diferentes para poder ter certeza de que eles funcionam.

Pesquise os exercícios em Portugol. Caso não os tenha, resolva-os antes no papel; só depois de ter a solução pronta, ou ao menos delineada, você deve digitar o código. Lápis e papel são ainda os melhores amigos do programador.

Atividade 6.10 - Faça um programa que leia três valores distintos a serem digitados pelo usuário, determine e exiba o menor deles.

Atividade 6.11 - Sabendo que triângulo é uma figura geométrica de três lados em que cada um dos lados é menor que a soma dos outros dois, faça um algoritmo que receba três valores e verifique se eles podem ser os comprimentos dos lados de um triângulo.

Atividade 6.12 - Refaça, agora em linguagem C, o algoritmo desenvolvido

no exercício 3 (aula 3).

Atividade 6.13 - Faça um programa que leia o salário bruto e calcule o salário líquido. Para esse programa, o salário líquido será o salário bruto menos os descontos de INSS e IR, seguindo as regras:

- caso o salário seja menor que R$1.500,00, não devemos descontar IR e descontaremos 8% de INSS;

- para salários a partir R$1.500,00, descontaremos 5% de IR e 11% de INSS.

Obs.: Essas faixas de cálculo são fictícias, apenas para exemplo, não condizendo com as leis em vigor no país.

6.2.4 Comando *switch*

Assim como o comando *if*, o comando *switch* é uma estrutura de decisão. Devemos utilizar o comando *switch* quando o programa tiver que escolher uma entre várias alternativas para um determinado valor.

Sintaxe:

```
switch (condição de teste)
{
    case constante 1:
        bloco de instruções 1
        break;
    case constante n:
        bloco de instruções n
        break;
    default: bloco de instruções padrão.
}
```

Como funciona:

A **condição de teste** deve ter alguns valores possíveis. Para cada valor possível, fazemos um bloco *case* contendo as instruções a serem executadas naquele caso. A instrução *break* sai do bloco e do *switch*. Por último, podemos utilizar uma condição *default*: o bloco de instruções desta condição só é executado caso a condição de teste não satisfaça a nenhum dos valores previstos anteriormente.

A Figura 6.14 apresenta um exemplo no qual a estrutura *switch* é utilizada em um programa para simular uma calculadora de quatro operações. Solicitamos a digitação do primeiro operando **num1**, do **operador** (+ - * /) e do segundo operando **num2**. Utilizamos, então, um *switch* de forma que, de acordo com a operação solicitada, imprimimos o resultado. O bloco *default* só será executado caso seja digitado um valor inválido para o operador.

```
#include <stdio.h>
#include <stdlib.h>
int main ()
{
    float num1, num2;
    char operador;

    printf("Digite o primeiro numero, o operador e o segundo numero\n");
    scanf("%f %c %f", &num1, &operador, &num2);
    switch (operador){
           case '+':
                printf("O resultado e %f\n", num1+num2);
                break;
           case '-':
                printf("O resultado e %f\n", num1-num2);
                break;
           case '*':
                printf("O resultado e %f\n", num1*num2);
                break;
           case '/':
                printf("O resultado e %f\n", num1/num2);
                break;
           default:
                printf("Operador Invalido\n", num1+num2);

    }
    system("pause");
    return(0);
}
```

Figura 6.14: Exemplo de utilização do comando *switch*

A Figura 6.15 exibe o resultado de uma execução desse programa, na qual o usuário informou os valores 20.5 * 3.

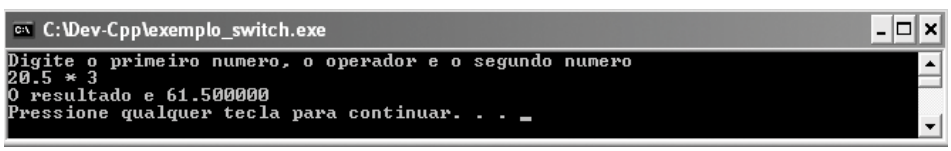

Figura 6.15: Resultado de uma execução do programa da Figura 6.14

Note que o programa apresentado na Figura 6.14 poderia ter sido implementado utilizando uma estrutura de *if...else*. De fato, em todos os casos nos quais o switch é aplicável, é possível também resolver utilizando *if... else*. Trata-se de uma opção do programador. De forma geral, optamos pelo *switch* quando temos um conjunto determinado de valores como opção.

O que aprendemos até aqui?

- Que há três comandos de seleção em C.
- Que o comando *if* é utilizado para decisão simples.
- Que o comando *if...else* é utilizado quando, com base em uma condição, o programa pode executar um ou outro bloco de comandos.
- Que podemos utilizar comandos *if...else* **aninhados**, ou seja, dentro de um bloco de comandos executados em um *if...else* podemos ter outra estrutura *if...else* e assim sucessivamente. Também podemos aninhar uma estrutura switch dentro de uma *if...else* e vice-versa.
- Que o comando *switch* é utilizado quando temos condições que não sejam expressões e temos uma lista de valores possíveis para a condição.

Para aprofundar seus conhecimentos, recomendo a leitura do capítulo 4 do livro **Treinamento em Linguagem C – Curso Completo – Módulo 1** de Victorine Mizrahi e do capítulo 3 (da página 61 à 74) do livro **C Completo e Total**, do autor Herbert Schildt.

Resumo

Nesta aula você conheceu os operadores lógicos e relacionais da linguagem C, bem como as estruturas de decisão dessa linguagem: *if..else* e *switch*. Você aprendeu que pode utilizar as estruturas de decisão de forma aninhada (uma dentro de outra) para construir lógica mais complexa. Para praticar, vários programas foram desenvolvidos, utilizando as estruturas estudadas.

Atividades de aprendizagem

1. Uma empresa dará aumento aos seus funcionários, de acordo com sua classe:

a) Classe A = 0,10 (10%) de aumento;

b) Classe B = 0,15 (15%) de aumento;

c) Classe C = 0,20 (20%) de aumento.

Usando o comando *switch*, faça um programa que leia o salário e a classe do funcionário, calcule e exiba os salários com os devidos aumentos.

2. Precisamos fazer um programa para uma biblioteca que receba o tipo do usuário e a classificação do livro e responda se o usuário pode ou não locar o livro seguindo as seguintes regras: Existem dois tipos de usuários: o tipo 'A' (aluno) e o tipo 'P' (professor). Existem duas classificações de livros: A e B. Livros do tipo A podem ser locados por qualquer usuário enquanto livros do tipo B só podem ser locados por professores.

Aula 7 - Estruturas de repetição em linguagem C

Objetivos

Conhecer a estrutura de repetição *while*.

Conhecer a estrutura de repetição *do...while.*

Conhecer a estrutura de repetição *for*.

Entender quando se devem utilizar cada uma das estruturas de repetição.

Criar programas em linguagem C utilizando estruturas de repetição.

Conforme vimos na primeira aula, são normais situações nas quais nós repetimos determinadas ações enquanto não atingimos um objetivo. Já na aula 4 vimos que, ao desenvolver nossos programas, deparamos com situações nas quais precisamos que um determinado bloco de instruções seja repetido enquanto uma determinada condição é válida. Nessas situações, utilizamos os comandos de repetição, também conhecidos como laços ou *loops*. Quando estudamos Portugol, aprendemos duas estruturas de repetição: **para...faça** e **enquanto...faça**. Agora vamos estudar as estruturas de repetição da linguagem C.

A linguagem C conta com três comandos de repetição: *for, while* e *do while.*

7.1 Comando *for*

O comando *for* é ideal para situações nas quais um bloco de instruções deve ser repetido um número fixo ou conhecido de vezes. Esse comando da linguagem C é equivalente ao **para...faça** que utilizamos em Portugol.

Sintaxe:

```
for (inicialização; teste; incremento)
{
      bloco de instruções;
}
```

Como funciona:

Os parênteses que seguem a palavra *for* contêm 3 expressões separadas por ponto-e-vírgula: expressão de inicialização, expressão de teste e expressão de incremento.

A expressão de **inicialização** é uma instrução de atribuição executada apenas uma vez, no início do laço. É geralmente utilizada para inicializar uma variável que irá controlar o número de repetições do laço.

A expressão de **teste** é a condição que controla o laço. Normalmente é uma expressão lógica que utiliza a variável de controle do laço. Essa expressão é verificada antes da execução do laço. Se for verdadeira, o laço é executado mais uma vez. Caso contrário, o laço é finalizado.

A expressão de **incremento** define a maneira como a variável de controle do laço será alterada a cada vez que o laço for repetido. Ela é executada ao final da execução de cada repetição do corpo do laço.

Para exemplificar o uso do comando *for*, vamos fazer um programa que leia a nota de 10 alunos e no final exiba a média da turma.

```
linha 1...#include <stdio.h>
linha 2...#include <stdlib.h>
linha 3...int main()
linha 4...{
linha 5... float nota, soma=0, media;
linha 6... int conta;
linha 7... for(conta=0;conta<=9;conta++)
linha 8... {
linha 9...          printf( "Digite a nota ");
linha 10.          scanf("%f", &nota);
linha 11...        soma=soma+nota;
linha 12... }//esta chave encerra o comando de repetição for
linha 13... media= soma/conta;
linha 14... printf( "A media da turma e: %f \n ", media);
linha 15... system("PAUSE");
linha 16... return 0;
linha 17...}
```

Vamos entender melhor algumas linhas do código acima.

linha 5... *float* nota, soma=0, media;

Houve necessidade de iniciarmos a variável **soma** com zero, pois terá valor cumulativo.

Já vimos que, ao declararmos uma variável, estamos reservando um espaço na memória, o qual não é necessariamente um espaço limpo. Isso significa que nossa variável no momento da declaração armazena apenas lixo. Ao atribuirmos o valor zero para ela, garantimos que seu valor se inicia com zero para que a soma seja acumulada corretamente.

linha 7... **for**(conta=0;conta<=9;conta++)

A linha do comando ***for*** controla a quantidade de vezes que o *loop* será executado. Observe que ele inicia a variável **conta** de zero (conta=0;), controla o loop para ser executado 10 vezes (conta<=9) e finalmente incrementa a variável **conta** (conta++).

É importante notar que o comando **conta** ++ é o mesmo que: **conta** = **conta** + **1**.

linha 11... soma=soma+nota;

Nessa linha acumula-se a soma das notas da turma.

linha 13... media= soma/conta;

Observe que essa linha de comando foi colocada após encerramento do ***for***, pois só nos interessa calcular a média depois que todas as notas forem somadas.

Como a variável **conta** guarda o número de vezes que o loop foi executado, que é igual à quantidade de alunos estipulada no programa, em vez de dividirmos a soma por 10, fazemos a divisão utilizando a variável **conta**. Lembre-se de que o loop é encerrado quando conta atinge o valor 10.

linha 14... **printf**("A media da turma e: %f \n ", media);

Para melhorarmos a exibição dessa mensagem, basta trocar %f por %.2f: serão exibidas apenas 2 casas depois da vírgula.

A Figura 7.1 apresenta nosso programa e o resultado de uma execução do mesmo na qual a média da turma foi de 7,09 pontos.

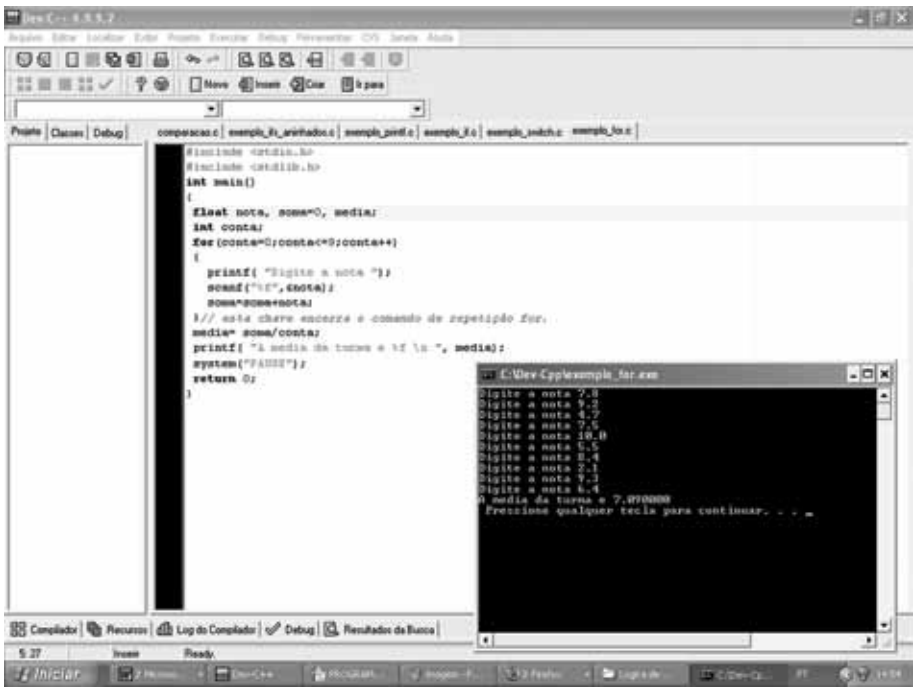

Figura 7.1: Código e resultado da execução do programa exemplo com *for*

Atividade 7.1 - Faça um programa que leia cinco valores reais e imprima o quadrado de cada um deles. Ao fim, imprima também o somatório dos cinco.

Atividade 7.2 - Faça um programa que calcule a média de 5 números inteiros dados como entrada e imprima o resultado.

Atividade 7.3 - Faça um programa que imprima todos os números pares no intervalo de 1 a 100.

Atividade 7.4 - Faça um programa que receba como entrada um valor inicial e um final de temperatura em graus Celsius e imprima, variando do valor inicial até o final, a temperatura em Celsius seguida do seu equivalente em Fahrenheit. Ou seja, faça uma conversão entre as duas medidas. A fórmula de conversão de Celsius para Fahrenheit é dada por: F=1,8C + 32, onde F é a temperatura em Fahrenheit e C a temperatura em Celsius. Por exemplo, suponha que o programa receba 8 como temperatura inicial e 10 como final. Ele deve imprimir:

8 Celsius = 46,4 Fahrenheit
9 Celsius = 48,2 Fahrenheit
10 Celsius = 50,0 Fahrenheit

Atividade 7.5 - Na matemática, o fatorial de um número natural n é dado pelo produto de todos os números inteiros e positivos menores ou iguais a n. Por exemplo, o fatorial de 5 é dado por 5 * 4 * 3 * 2 * 1. Desenvolva um programa que calcule o fatorial de um número dado como entrada.

Atividade 7.6 - Refaça, agora utilizando a linguagem C, o exercício 4.2. Esse exercício já foi implementado em Portugol, utilizando a estrutura **para... faça**.

Atividade 7.7 - Refaça, agora utilizando a linguagem C, o exercício 4.3. Esse exercício já foi implementado em Portugol, utilizando a estrutura **para... faça**.

Atividade 7.8 - Refaça, agora utilizando a linguagem C, o exercício 4.4. Esse exercício já foi implementado em Portugol, utilizando a estrutura **para... faça.**

Atividade 7.9 - Refaça, agora utilizando a linguagem C, o exercício 4.5. Esse exercício já foi implementado em Portugol, utilizando a estrutura **para... faça.**

7.2 Comando *while*

O comando *while* é ideal para situações nas quais não sabemos o número exato de vezes que o bloco de instruções deve ser repetido, mas também pode ser utilizado para substituir laços *for*.

Sintaxe:
while (**condição**)
{
    bloco de instruções;
}
Como funciona:
Enquanto a condição especificada no cabeçalho do laço for satisfeita, o bloco de instruções é executado. Assim, antes de cada execução do bloco, a condição é avaliada: caso seja verdadeira, o bloco é executado; caso seja falsa, o laço é finalizado.

O comando *while* é o equivalente na linguagem C à estrutura **enquanto... faça** do Portugol.

Vamos utilizar o mesmo exemplo do comando *for*, porém, ao invés de predefinir que serão entradas dez notas, leremos a primeira nota e, daí em diante, perguntaremos ao usuário se deseja digitar mais notas.

Nosso código ficará assim:

```
linha 1..#include <stdio.h>
linha 2..#include <stdlib.h>
linha 3..int main()
linha 4..{
linha 5..  float nota, soma=0, media;
linha 6..  int resp=1, contador=0;
linha 7..  while(resp==1)
linha 8..  { // esta chave inicia o bloco de repetição while
linha 9..     printf( "Digite a nota ");
linha 10.     scanf("%f",&nota);
linha 11.     soma=soma+nota;
linha 12.     printf("Digite 1 para continuar ou digite outra
                    tecla para finalizar.... ");
linha 13.     scanf("%d",&resp);
linha 14.     contador++; /* essa linha é igual a
                          contador=contador + 1*/
linha 15.. }
linha 16.. media= soma/contador;
linha 17.. printf( "A media da turma e: %.2f \n ", media);
linha 18.. system("PAUSE");
linha 19.. return 0;
linha 20..}
```

Vamos entender melhor algumas linhas do código acima.

linha 6... int resp=1, contador=0;

A variável **resp** será responsável por armazenar a resposta do usuário. Observe que ela é inicializada = 1, para que o programa execute o laço a primeira vez.

A variável **contador** guardará a quantidade de vezes que o usuário digitou uma nota, o que corresponderá à quantidade de alunos. Precisaremos desse total para calcular a média da turma.

linha 7... *while* (resp==1)

Observe que enquanto **resp** for igual a 1(um) o laço será executado. Assim podemos entender o motivo pelo qual iniciamos a variável **resp** no momento da sua declaração. Se o valor 1 não fosse atribuído à variável no início, o laço nunca seria executado.

linha 14... contador++;

A variável **contador** está sendo incrementada cada vez que o laço é executado, ou seja, está contando a quantidade de notas digitadas.

A Figura 7.2 apresenta nosso programa e o resultado de uma execução na qual foram digitadas seis notas.

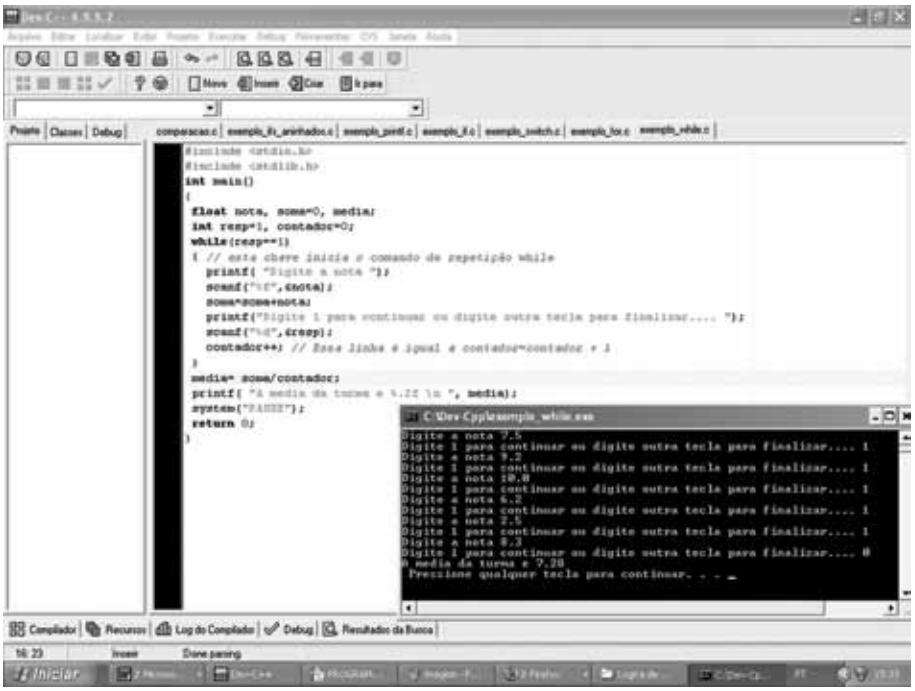

Figura 7.2 – Código e resultado da execução do programa exemplo com *while*

Atividade 7.10 - Faça um programa que leia os valores dos salários atuais dos funcionários de uma empresa e imprima os valores com aumento. Se o salário for menor ou igual a R$ 500,00, o programa deve aplicar um aumento de 0.10 (10%) e se for maior que R$ 500,00, o aumento deve ser de 0.08 (8%). Assim como no nosso último exemplo, o usuário é que deve informar quando deseja sair. Ou seja, ele deve digitar o salário de um funcionário e o programa vai exibir o valor com o aumento. Depois ele deve responder se deseja digitar outro salário ou não. Se ele responder que sim, deve-se solicitar o novo salário e exibir o valor com aumento. Então, torna-se a perguntar se ele deseja informar um novo salário e repetir o laço até que ele não queira informar novos salários.

Atividade 7.11 - Faça um programa que fique em um laço solicitando a digitação de números inteiros e só pare de solicitar a digitação de novos números quando o usuário informar o número 0. Quando o número 0 for informado, o programa deve exibir a quantidade de números digitados, a quantidade de números pares, a quantidade de números ímpares e a média

dos valores dos números digitados.

Atividade 7.12 - No exercício 7.5 fizemos um programa para calcular o fatorial de um dado número. Refaça tal exercício, utilizando um laço *while* em lugar do laço *for*.

Atividade 7.13 - Refaça, agora utilizando a linguagem C, o exercício 2, que já foi implementado em Portugol.

Atividade 7.14 - Refaça, agora utilizando a linguagem C, o exercício 3, que já foi implementado em Portugol.

7.3 Comando *do...while*

O comando do *while* é muito parecido com o comando *while* que acabamos de aprender. A única diferença é que, com o comando *do...while*, assegura-mos que o bloco de instruções do laço seja executado ao menos uma vez. Depois da primeira execução, o bloco de instruções continua sendo executado enquanto a condição permanecer verdadeira.

Sintaxe:
do
{
    bloco de instruções;
} while (**condição**)

Como funciona:
A primeira execução do bloco de instruções ocorre sem fazer a avaliação da condição. Ao final da execução a condição é avaliada e o bloco de instruções é repetido se a condição permanecer verdadeira.

Para mostrar na prática a utilização *do...while*, vamos refazer o mesmo exemplo utilizado para o laço *while.* Nosso programa ficará assim:

```
linha 1..#include <stdio.h>
linha 2..#include <stdlib.h>
linha 3..int main()
linha 4..{
linha 5..  float nota, soma=0, media;
linha 6..  int resp, contador=0;
linha 7..  do
linha 8..  {
linha 9..     printf( "Digite a nota ");
linha 10.     scanf("%f",&nota);
linha 11.     soma=soma+nota;
linha 12.     printf("Digite 1 para continuar ou digite outra
                    tecla para finalizar.... ");
linha 13.     scanf("%d",&resp);
linha 14.     contador++; /* Esse comando é igual a
                          contador=contador + 1;*/
linha 15.  } while(resp==1);
linha 16.  media= soma/contador;
linha 17.  printf( "A media da turma e %.2f \n ", media);
linha 18.  system("PAUSE");
linha 19.  return 0;
linha 20.}
```

Vamos entender melhor o código acima, analisando algumas linhas principais:

linha 6... **int resp, contador=0;**

Note que nesse exemplo não precisamos iniciar a variável **resp** com 1, pois a condição só será testada após a primeira execução. Assim, na primeira vez em que a condição for testada, o usuário já terá respondido à pergunta. Aliás, essa é a única diferença entre o *while* e o *do...while.* No *while* o teste é feito antes da execução do laço; já no *do...while* primeiro executa-se o laço e só depois o teste é feito para verificar se o laço continuará a ser executado.

linha 7... **do**

Início do comando *do...while*. O bloco será executado ao menos uma vez, pois, como vimos, o teste é feito no final do laço.

linha 8... **{**

Delimita o início do bloco de instruções do laço.

linha 15... } while(resp==1);

Enquanto essa condição for verdadeira, o programa continuará em execução. Delimita o fim do bloco de instruções do laço.

O que aprendemos até aqui?

- Os comandos *for*, *while* e *do...while* são responsáveis por executar laços em um programa em linguagem C;

- Os três comandos podem ser usados para resolver o mesmo problema. Cabe ao programador decidir qual deles melhor responderá às necessidades para a solução de cada problema;

- Mas, de uma forma geral o *for* é aconselhável em casos onde se sabe o número de vezes que o laço se repetirá, enquanto o *while* e o *do...while* são mais adequados em situações onde há uma condição lógica que define o momento de parada do laço;

- A diferença do *while* para o *do...while* é que no primeiro a condição é testada antes de se executar o bloco de comandos do laço; já no segundo o teste só é feito após a execução do bloco para se definir se será exe cutado novamente. Assim, a estrutura *do...while* é recomendada para casos em que se quer garantir que o bloco de repetição seja executado ao menos uma vez.

Para aprofundar seus conhecimentos, recomendo a leitura do capítulo 3 do livro **Treinamento em Linguagem C – Curso Completo – Módulo 1**, de Victorine Mizrahi e do capítulo 3 (da página 74 à 85) do livro **C Completo e Total**, do autor Herbert Schildt.

Resumo

Nesta aula você conheceu as três estruturas de repetição da linguagem C: *for*, *while* e *do...while*. Você aprendeu em quais situações deve utilizar cada uma delas. Para praticar, vários programas foram desenvolvidos utilizando as estruturas estudadas.

Atividades de aprendizagem

1. Construa um programa capaz de ler uma série de números até que apareça um número entre 1 e 5. Ao final, exiba a quantidade de números digitados, o valor da soma dos números digitados e a média dos valores dos números digitados.

2. Refaça o exercício 7.10, porém, agora utilize *do...while* em lugar do *while* utilizado anteriormente.

Aula 8 – Vetores

Objetivos

Entender o conceito de vetor.

Aprender a utilizar vetores em linguagem C.

Entender como utilizar vetores para simular o tipo cadeia em linguagem C.

Implementar algoritmos utilizando vetores.

Um vetor é uma estrutura de dados utilizada para representar certa quantidade de variáveis de valores homogêneos, ou seja: um conjunto de variáveis, todas do mesmo tipo.

Na aula anterior, fizemos um programa que calculava a média de notas de uma turma. Nesse programa sempre líamos a nota de cada aluno utilizando a mesma variável para armazenar o valor. Ou seja, a cada rodada do laço, a variável assumia a nota de um aluno e essa nota era acumulada numa variável *soma*.

Assim, se no final do programa quiséssemos exibir as notas de todos os alunos, não seria possível, pois sempre substituíamos a nota de um aluno pela nota do aluno seguinte. Para que fosse possível armazenar as notas de todos os alunos, teríamos duas alternativas: a primeira, menos inteligente, seria declarar uma variável para armazenar a nota de cada aluno; e a segunda seria declarar um vetor de tamanho igual à quantidade de alunos.

Sintaxe para declaração de vetores:

tipo_do_vetor nome_do_vetor [tamanho];

Como funciona:

A única diferença entre a declaração de vetores e a declaração de variáveis simples é que, ao declarar vetores, especificamos ao fim da declaração e entre colchetes [ ] o tamanho do vetor, ou seja, sua capacidade de armazenamento.

Por exemplo: se precisarmos armazenar a nota de 10 alunos podemos declarar um vetor de 10 posições do tipo float. A declaração do nosso vetor ficaria assim: float notas[10];

8.1 Referenciando elementos e armazenando dados em vetores

Um vetor é como uma coleção de caixinhas numeradas. Cada caixinha é capaz de armazenar um valor e tem o seu "endereço". O "endereço" de cada caixinha é conhecido como índice e serve para identificar qual posição do vetor queremos acessar. Abaixo temos a representação gráfica no nosso vetor de notas de dez posições declarado acima:

Índice	0	1	2	3	4	5	6	7	8	9
Valor	8.5	7.6	9.3	10.0	6.3	4.7	8.8	9.1	3.4	10.0

No vetor representado acima temos, por exemplo, armazenado no índice "2" o valor "9.3". Note que, como declaramos um vetor com 10 posições, os índices variam de 0 a 9.

Mas, resta uma dúvida: como referenciar as posições do vetor e armazenar dados nas mesmas? A sintaxe para utilizar uma posição do vetor é: **nome_do_vetor[indice];**

Assim, para armazenar o valor 8.5 na primeira posição do nosso vetor, seria utilizado o comando: **notas[0] = 8.5;**

A Figura 8.1 apresenta um exemplo simples que utiliza um vetor de 5 posições e atribui um valor a cada posição. A Figura 8.2 exibe o resultado da sua execução.

```
#include <stdio.h>
#include <stdlib.h>
int main ()
{
    int numeros[5];
    numeros[0] = 10;
    numeros[1] = 20;
    numeros[2] = 30;
    numeros[3] = 40;
    numeros[4] = 50;
    printf ("O valor %d foi armazenado na posicao zero do vetor.\n", numeros[0]);
    printf ("O valor %d foi armazenado na posicao um do vetor.\n", numeros[1]);
    printf ("O valor %d foi armazenado na posicao dois do vetor.\n", numeros[2]);
    printf ("O valor %d foi armazenado na posicao tres do vetor.\n", numeros[3]);
    printf ("O valor %d foi armazenado na posicao quatro do vetor.\n", numeros[4]);
    system("pause");
}
```

Figura 8.1: Exemplo de programa utilizando vetor

Figura 8.2: Resultado da execução do programa da Figura 8.1

Note que a única diferença entre a atribuição de valores quando utilizamos variáveis e quando utilizamos vetores é que, usando vetores, precisamos indicar o índice a ser utilizado.

Para utilizar vetores para armazenar valores obtidos através do comando *scanf*, a única diferença é também a informação do índice do vetor a ser utilizado. Assim, por exemplo, caso queiramos ler um número e armazená-lo no índice zero do nosso vetor, o comando ficaria assim: *scanf("%d",* **&numeros[0]);.**

Agora, suponha que queiramos fazer um programa para solicitar a digitação das notas de 10 alunos. Será que teríamos de escrever as dez atribuições uma a uma? Imagine então um caso onde fossem 500 valores... O exemplo 2 mostra como utilizar um comando de repetição para não ter de escrever todas essas atribuições:

Exemplo 2:

```
linha1..#include <stdio.h>
linha2..#include <stdlib.h>
linha3..int main ()
linha4..{
linha5..    float notas[10];
linha6..    int indice;
linha7..    printf("Lendo as notas:\n");
linha8..    for (indice=0; indice<10; indice++){
linha9..       printf("Digite a nota do proximo aluno: ");
linha10.       scanf("%f", &notas[indice]);
linha11.    } //Finalizando o bloco do for
linha12.    printf("Exibindo as notas digitadas: \n");
linha13.    for (indice=0; indice<10; indice++){
linha14.       printf("A nota %f foi armazenada na posicao %d
                    do vetor.\n",notas[indice], indice);
linha15.    }//Finalizando o bloco do for
linha16.    system("pause");
linha17.}
```

Vamos entender o código acima analisando as principais linhas:

linha5..... float notas[10];

O vetor **notas** foi declarado com capacidade para armazenar dez valores do tipo *float*.

linha6..... int indice;

Foi declarada a variável **indice** do tipo inteira para armazenar o índice do vetor a ser utilizado.

linha8..... for (indice=0; indice<10; indice++){

Será utilizado um comando *for* para variar o valor da variável **indice** de zero até nove, de forma que, cada vez que o laço for executado, uma posição diferente do vetor será utilizada.

linha10... scanf("%f", &notas[indice]);

A nota digitada será armazenada no vetor de notas na posição indicada pela variável **indice.**

linha13... for (indice=0; indice<10; indice++){

Um novo laço for será utilizado, agora para imprimir o conteúdo do vetor.

linha14... printf("A nota %f foi armazenada na posicao %d do vetor.\n",notas[indice], indice);

Será impressa na tela a nota armazenada em determinada posição, bem como o índice dessa posição.

A Figura 8.3 exibe o resultado de uma execução desse programa.

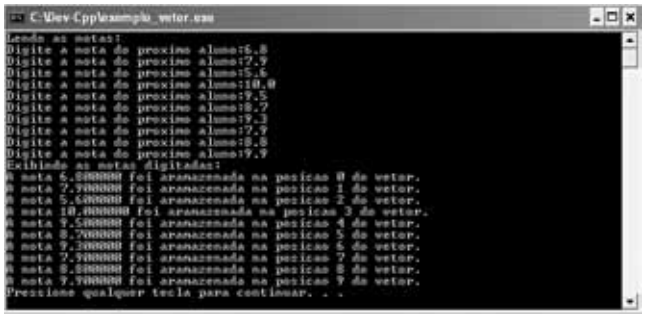

Figura 8.3: Resultado da execução do exemplo 2

- Note que o fato de termos declarado o vetor com 10 posições não significa que estejamos livres do controle do índice;

- A linguagem C não verifica se o índice que você usou está dentro dos limites válidos. Você é quem deverá ter o cuidado de controlar os limites, como fizemos nos laços do exemplo anterior;

- O primeiro índice de um vetor é o índice zero. Assim, se tenho um vetor com "n" posições, seus índices vão de zero a "n – 1".

8.2 Utilizando vetores de caracteres

Quando estudamos Portugol, na aula 2, vimos que um dos tipos de variáveis existentes era o tipo **cadeia de caracteres**. Esse tipo de variável era utilizado para armazenar textos como palavras e nomes. Como vimos até agora, não há um tipo primitivo de variáveis em C que seja equivalente ao tipo **cadeia de caracteres**. Assim, como armazenaremos textos em linguagem C?

A resposta está na utilização de vetores. Para trabalhar com cadeias de caracteres em C, devemos utilizar vetores de caracteres, ou seja, vetores do tipo *char.*

Em textos técnicos de programação, muito comumente, cadeias de caracteres também são conhecidas como *string*. Uma *string* em C é um vetor do tipo *char* terminado pelo caractere nulo ('/0'). Este detalhe "toda *string* é finalizada pelo caractere nulo" é importante, pois, se desejarmos armazenar uma cadeia de 8 caracteres, deveremos declarar um vetor de, pelo menos, 9 posições, pois a última posição conterá o caractere nulo.

Veja na Figura 8.4 uma demonstração de um vetor de 9 posições, para armazenar a *string* "Educação".

NOME

E	D	U	C	A	C	A	O	\0
0	1	2	3	4	5	6	7	8

Figura 8.4: Demonstração do vetor NOME

A declaração do nosso vetor ficará assim: **char** NOME[9]. Observe que declaramos uma posição a mais, garantindo uma posição para o caractere nulo.

Para ler uma ***string*** digitada por um usuário, devemos utilizar a função ***gets***( ). Essa função colocará o terminador nulo na ***string***, assim que a tecla *enter* for pressionada.

Vejamos um exemplo de leitura de uma cadeia de caracteres utilizando ***gets().***

Exemplo 3:

```
linha 1..#include <stdio.h>
linha 2..#include <stdlib.h>
linha 3..int main ( )
linha 4..{
linha 5..    char nome[20];
linha 6..    printf("Digite seu nome: ");
linha 7..    gets(nome);
linha 8..    printf("O nome digitado foi: %s \n",nome);
linha 9..    system("pause");
linha 10.}
```

Vamos entender algumas linhas do código:

linha 5... char nome[20];

Declaramos um vetor com 20 posições. Isso significa que esse vetor poderá armazenar até 19 caracteres, pois um caractere será utilizado para armazenar o caractere nulo que indica o fim da *string*.

linha 7... gets(nome);

Observe que para leitura não utilizamos a função ***scanf***( ); a função utilizada para leitura da *string* foi a *gets*( ).

linha 8... printf("O nome digitado foi %s \n",nome);

O que nos chama a atenção nessa linha é o caractere de impressão %s. Esse caractere, como já vimos, imprime uma cadeia de caracteres.

A Figura 8.5 exibe o resultado da execução desse programa.

Figura 8.5: Resultado da execução do exemplo 3

Para aprofundar seus conhecimentos, recomendo a leitura do capítulo 7 (da página 185 à 196) do livro Treinamento em Linguagem C – Curso Completo – Módulo 1, de Victorine Mizrahi e do capítulo 4 (da página 92 à 98) do livro C Completo e Total do autor Herbert Schildt.

Resumo

Nesta aula você aprendeu o conceito de vetor e como utilizar tal conceito com a linguagem C. Entre outras coisas, viu que a indexação de vetores nessa linguagem se inicia no índice 0 e que, como a linguagem C não conta com um tipo primitivo cadeia, vetores de caracteres são utilizados para armazenar palavras e frases. Para praticar, você implementou seus primeiros programas utilizando vetores.

Atividades de aprendizagem

1. Faça um programa que:

a) preencha dois vetores **a** e **b** de cinco posições, com números inteiros;

b) atribua a um vetor **res** à soma do vetor **a** com b (a primeira posição de a será somada à primeira posição de b e o resultado será atribuído à primeira posição do vetor res);

c) mostre os valores do vetor res.

2. Faça um programa que solicite a digitação e armazene 20 números reais em um vetor. Depois o programa deve ficar disponível para o usuário digitar o valor do índice para que seja exibido o número armazenado no índice solicitado. Para encerrar o programa, o usuário deve informar um

índice inválido (lembre-se de que para um vetor de 20 posições os índices válidos são de 0 a 19).

3. Faça um programa que solicite a digitação de 10 números inteiros e os armazene em um vetor. Depois o programa deve ler o vetor e imprimir na tela uma listagem dos múltiplos de 2, uma outra dos múltiplos de 3 e uma última listagem dos múltiplos de 5.

4. Faça um programa que solicite a digitação do nome do aluno e de 3 notas do mesmo e calcule a média das notas. Caso a média seja maior que 60, imprima uma mensagem contendo o nome do aluno e a palavra "Aprovado". Caso contrário, a mensagem deve conter o nome do aluno e a palavra "Reprovado".

5. Faça um programa que solicite a digitação de 10 números inteiros. Depois disso, o programa deve imprimir:

a) a lista dos dez números digitados;

b) uma mensagem informando qual foi o maior número digitado;

c) uma mensagem informando qual foi o menor número digitado;

d) uma mensagem informando a média dos números digitados;

e) uma mensagem informando a quantidade de números digitados maiores do que a média calculada no item anterior;

f) uma mensagem informando a quantidade de números digitados menores do que a média calculada no item anterior.

Referências

FARRER, Harry. **Algoritmos Estruturados.** Rio de Janeiro: LTC, 1999.

FORBELLONE, A. L. V., EBERSPÄCHER, H. F., **Lógica de Programação – A construção de algoritmos e estrutura de dados.** São Paulo: Makron Books, 2005.

KERNIGHAN, Brian W. **C Linguagem de Programação Padrão ANSI**. Rio de Janeiro: Elsevier, 1989.

LAUREANO, M. **Programando em C.** Rio de Janeiro: Brasport, 2005.

MIZRAHI, Victorine V. **Treinamento em Linguagem C** – Curso Completo – Módulo 1. São Paulo: Mc Graw Hill, 1990.

MORAES, P. S. **Curso básico de lógica de programação.** 2000. Disponível em: <http://www.siban.com.br/destaque/21_carta.pdf>. Acesso em: 30 set. 2008.

SANT'ANNA, S. R. **Programação I.** Vitória: CEFETES, 2007.

SCHILDT, Herbert. **C Completo e Total**. São Paulo: Pearson, 2006.

TANENBAUM, Andrew S. **Sistemas Operacionais.** Porto Alegre: Bookman, 2000.

Currículo do professor-autor

Victorio Albani de Carvalho obteve a titulação de técnico em processamento de dados pela antiga Escola Técnica Federal do Espírito Santo em 1998. Em 2003 se formou bacharel em Ciência da Computação na Universidade Federal do Espírito Santo. Nessa mesma universidade obteve o título de mestre em Informática em 2006.

Atuou em várias empresas públicas e privadas, dentre as quais destacam-se Xerox, Unisys e Cesan, sempre exercendo funções relacionadas ao processo de desenvolvimento de software, como programador, analista/projetista de sistemas e líder de equipe de desenvolvimento.

Em 2007 teve sua primeira experiência como professor de cursos superiores na Faculdade Salesiana de Vitória, onde lecionou por um ano.

Em 2008 se tornou professor do departamento de informática do Instituto Federal do Espírito Santo – Campus Colatina, onde leciona até a presente data, trabalhando com os cursos Técnicos em Informática, na modalidade presencial e a distância, e Superior de Tecnologia em Redes de Computadores.

e-Tec Brasil
Escola Técnica Aberta do Brasil

ISBN: