PROFª. SOLIMARA RAVANI DE SANT'ANNA

PROGRAMAÇÃO I

VITÓRIA

2009

Governo Federal
Ministro de Educação
Fernando Haddad

Ifes – Instituto Federal do Espírito Santo
Reitor
Dênio Rebello Arantes

Pró-Reitora de Ensino
Cristiane Tenan Schlittler dos Santos

Coordenadora do CEAD – Centro de Educação a Distância
Yvina Pavan Baldo

Coordenadoras da UAB – Universidade Aberta do Brasil
Yvina Pavan Baldo
Maria das Graças Zamborlini

Curso de Tecnologia em Análise e Desenvolvimento de Sistemas
Coordenação de Curso
Isaura Nobre

Designer Instrucional
Danielli Veiga Carneiro

Professor Especialista/Autor
Solimara Ravani Sant'Anna

Catalogação da fonte: Rogéria Gomes Belchior - CRB 12/417

S231 Sant'Anna, Solimara Ravani

Programação I. / Solimara Ravani Sant'Anna. – Vitória: CEFETES, 2007.
139 p. : il.
1. Algoritmos. 2. C (linguagem de programação de computadores).
3. Programação de computadores. I. Centro Federal de Educação Tecnológica do Espírito Santo. II. Título.

CDD 005.133
005.1

Ifes – Instituto Federal do Espírito Santo
Av. Vitória – Jucutuquara – Vitória – ES - CEP - (27) 3331.2139

Créditos de autoria da editoração
Capa: Juliana Cristina da Silva
Projeto gráfico: Juliana Cristina da Silva / Nelson Torres
Iconografia: Nelson Torres
Editoração eletrônica: Duo Translations

Revisão Técnica:
Henrique Monteiro Cristovão
Revisão de texto:
Ilioni Augusta da Costa
Maria Madalena Covre da Silva

Olá, Aluno(a)!

É um prazer tê-lo(a) conosco.

O Ifes oferece a você, em parceria com as Prefeituras e com o Governo Federal, o Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, na modalidade a distância. Apesar de este curso ser ofertado a distância, esperamos que haja proximidade entre nós, pois, hoje, graças aos recursos da tecnologia da informação (e-mails, chat, videoconferência, etc.), podemos manter uma comunicação efetiva.

É importante que você conheça toda a equipe envolvida neste curso: coordenadores, professores especialistas, tutores a distância e tutores presenciais, porque, quando precisar de algum tipo de ajuda, saberá a quem recorrer.

Na EaD - Educação a Distância, você é o grande responsável pelo sucesso da aprendizagem. Por isso, é necessário que você se organize para os estudos e para a realização de todas as atividades, nos prazos estabelecidos, conforme orientação dos Professores Especialistas e Tutores.

Fique atento às orientações de estudo que se encontram no Manual do Aluno.

A EaD, pela sua característica de amplitude e pelo uso de tecnologias modernas, representa uma nova forma de aprender, respeitando, sempre, o seu tempo.

Desejamos-lhe sucesso e dedicação!

Equipe do Ifes

ICONOGRAFIA

Veja, abaixo, alguns símbolos utilizados neste material para guiá-lo em seus estudos.

Fala Professor 

Fala do Professor

Conceitos 
Conceitos importantes. Fique atento!

Atividades 
Atividades que devem ser elaboradas por você, após a leitura dos textos.

Indicações 

Indicação de leituras complemtares, referentes ao conteúdo estudado.

Atenção 
Destaque de algo importante, referente ao conteúdo apresentado. Atenção!

Reflexão 
Reflexão/questionamento sobre algo importante referente ao conteúdo apresentado.

Anotações 
Espaço reservado para as anotações que você julgar necessárias.

Sumário

PROGRAMAÇÃO I

APRESENTAÇÃO

Olá,

Nessa disciplina vamos conhecer o que é Algoritmo e aprender a resolver problemas de forma que o computador possa entender e executar.

Vamos aprender os conceitos básicos da linguagem de programação C, que será utilizada para desenvolvimento dos programas.

O material impresso que você utilizará nessa disciplina foi preparado valorizando os detalhes, no intuito de amenizar as dificuldades que possam aparecer em nosso percurso, na sala de aula.

É importante que você o tenha sempre ao alcance, aproveitando o tempo disponível para leitura, revisão ou execução das atividades.

Todas as atividades aqui propostas, apesar de não serem avaliativas, deverão ser realizadas individualmente, pois só dessa forma você adquirirá auto confiança, e prática na solução de problemas que envolvem computadores.

Para valorizar o estudo of-line, serão encontradas ao longo do material, Ilustrações demostrando digitação, compilação e execução (no DEV C++), dos exemplos apresentados, já que alguns alunos não dispõem de computador com facilidade.

Apesar de a equipe EaD ser composta por profissionais compromissados com o seu aprendizado, é importante que você faça a sua parte com o mesmo comprometimento; só assim todos, você e a equipe EaD, poderão desfrutar do sucesso ao final dessa caminhada.

Bom estudo!!!!

CAPÍTULO 1

ALGORITMOS

Caro aluno,

Vamos iniciar o primeiro capítulo, em que você estudará o conceito de Algoritmo. Nessa fase, trabalharemos com exemplos da nossa vida diária no intuito de um entendimento correto desse conceito.

É importante que ao final deste Capítulo, você tenha clareza a respeito do significado de Algoritmo, a fim de iniciarmos nossa trajetória no mundo computacional.

Bom estudo!

Fala Professor

Quando criamos um Algoritmo, o que fazemos, é apontar uma dentre muitas soluções possíveis a um problema qualquer.

Para que a solução por nós apontada possa solucionar corretamente o problema, ela deverá ser pensada, planejada, executada e, por fim, testada. Somente após passarmos por essas etapas, teremos garantia mínima da solução do problema. Todas as etapas serão utilizadas por você no decorrer do nosso curso.

1.1 Entendendo Algoritmos

Exemplo: Suponha o seguinte problema: como proceder para **NÃO** obter sucesso nos cursos a distância.

Receita: Como **NÃO** obter sucesso nos Cursos EaD

1. Não administre seu tempo de forma a estudar pelo menos 2 horas por dia.
2. Só participe das discussões nos fóruns quando o assunto lhe interessar.
3. Nunca tenha o material impresso ao alcance de forma a aproveitar qualquer tempo para leitura.
4. Avance para o Capítulo seguinte na certeza de que entenderá melhor o anterior.
5. Nunca tente resolver os exercícios sozinho, resolva-os sempre com a ajuda do tutor.

Conceitos

Certamente que ***NÃO*** *seguiremos esses passos, mas fica claro que basta qualquer pessoa seguí-los para obter o resultado.*

Exemplo: Suponha outro problema: como fazer um bolo.

Receita: Bolo Festa

1. Separe os seguintes ingredientes: ovos, trigo, sal, leite, açúcar e fermento.
2. Bata em uma tigela a manteiga com o açúcar.
3. Acrescente as gemas, o leite, o trigo e o sal.
4. Misture bem.
5. Acrescente o fermento.
6. Coloque a massa na assadeira.
7. Asse por 30 minutos.

Os dois exemplos acima são Algoritmos, pois enumeram passos que levam à solução de problemas específicos.

1.2 Construindo Algoritmos

Vamos construir Algoritmos para resolver problemas propostos em alguns sites.

Atividades

1. Tente conduzir a vaquinha até o troféu. Acesse o link: http://www.q-lympics.de/iq-marathon/

Clique em 'Starten' e depois em 'Ok'. Agora, arraste as setas para determinar o caminho da vaquinha e em seguida aperte 'Go'. Observe o código que aparece em cada mudança de fase. Ele permite que você depois continue daquele ponto em diante.

Obs.: as setas azuis, diferentemente das vermelhas, não se apagam quando a vaquinha passa.

Copie aqui o código da última fase que você conseguiu chegar. Ao todo são 43 fases. _____

_____

_____

_____

_____

Atividades

2. Acesse o link: http://rachacuca.com.br/calculadora-quebrada/

Nesse link encontraremos uma calculadora quebrada e problemas a resolver.

Clicando no botão <**PRÓXIMO**>, teremos acesso ao primeiro problema.

Lembre-se de descrever a solução passo a passo, como fizemos nos exemplos acima.

3. Vamos acessar o link: http://rachacuca.com.br/jarros/

Nesse link encontraremos jarros e problemas a resolver.

Clicando no botão <**JOGAR**>, teremos acesso ao primeiro problema.

Resolva-o.

Fala Professor

Olá,

Com a conclusão dos exercícios, podemos observar que cada um aponta uma solução, para os problemas apresentados. Cada solução criada é um Algoritmo.

Vamos em frente!!!

Anotações

Fala Professor

Nosso objetivo, no entanto, é criar Algoritmos compostos de passos que o computador seja capaz de entender e executar e não Algoritmos com passos que as pessoas sejam capazes de entender e executar. Então, precisaremos somar mais conhecimentos aos vistos até aqui, uma vez que o computador não poderia executar os passos descritos em nossos exemplos acima.

Algoritmo: É uma sequência não ambígua e finita de instruções, cuja a execução, em tempo finito, resolve um problema computacional. (SEBESTA, 2003).

Conceitos

1.3 Construindo Algoritmos Para Computadores

Construir Algoritmos para computadores requer, em primeiro lugar, a transformação do nosso passo a passo em uma estrutura que possa ser entendida e executada pelo computador. Para isso, utilizaremos inicialmente uma linguagem conhecida como Portugol e mais tarde, assim que os principais conceitos tenham sido assimilados, a Linguagem C.

Para entender essa estrutura, vamos estudar o que é memória, variáveis e constantes.

1.3.1 Reservando Memória

É na memória do computador que o nosso programa será executado.

Quando vamos usar a memória em um Algoritmo para armazenar dados inerentes à solução de um problema como o resultado de uma operação ou mesmo um número ou uma palavra, é necessário informar primeiro ao computador que precisaremos dessa memória. Essa informação é passada ao computador por meio de comando, que veremos posteriormente.

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

Conceitos

1.3.2 Variáveis

Sintaxe: ***tipo identificador <ou Lista-de-identificadores>;***

Além de reservar a memória, temos que informar ao computador como vamos nos referir a essa reserva, ou seja, como denominaremos, o espaço de memória reservado.

O nome dado a esse espaço é seu endereço.

Dizemos que os endereços nomeados de memória são as variáveis do programa.

O conteúdo de uma variável pode mudar durante a execução do programa.

Por que precisamos declarar variáveis?

Em nosso Algoritmo precisaremos manipular diversos valores de forma a encontrar a solução do problema.

Esses valores deverão estar armazenados de forma que, quando necessários, possam ser identificados corretamente.

Uma correta declaração de variáveis é que disponibilizará uma identificação precisa do valor já que a variável é justamente o endereço do valor armazenado.

O que são valores?

Os dados necessários à solução do problema são os valores, que devem ser manipulados de maneira a fornecer resultados que solucionem o problema.

Os valores poderão ser inteiros (número da matrícula do funcionário, número de filhos), reais (valor do salário e o desconto de IRRF) e lógicos (funcionário sindicalizado ou não: assume falso ou verdadeiro).

Cada variável corresponde a uma posição de memória, cujo conteúdo pode variar durante a execução de um programa.

O que são tipos de variáveis?

Como dissemos, os valores diferem quanto ao seu tipo, isso significa que, para cada tipo de dado, teremos um tipo de variável específica, que passamos a ver:

- *Tipo inteiro:* Declararemos variáveis do tipo *numérico inteiro* para representar uma localização de memória do computador utilizada para armazenar os valores inteiros (positivos ou negativos) que constarem em nosso Algoritmo.
  Exemplo: 1 2000 -3

- *Tipo real:* Declararemos variáveis do tipo *numérico real* para armazenar os valores reais (números fracionários: aqueles com ponto decimal) que constarem em nosso Algoritmo.
  Exemplo: 1,0 2,000 -3,0

- *Tipo caractere*: Declararemos variáveis do tipo *literal caractere* para armazenar um único caractere, que pode ser uma letra ou um símbolo.

Exemplo: Para identificar o sexo do indivíduo, armazenaremos apenas o caractere 'F' ou 'M'.

- *Tipo cadeia*: Declararemos variáveis do *tipo literal* cadeia para armazenar uma seqüência de caracteres, ou seja, uma palavra, uma mensagem, um nome.

  Exemplo: Se decidirmos armazenar a palavra "Masculino" ou "Feminino", para identificar o sexo do indivíduo no lugar do caractere 'F' ou 'M', teremos que declarar o tipo da variável como cadeia.

- *Tipo lógica*: Declararemos variáveis do *tipo lógico* para armazenar valores lógicos, VERDADEIRO ou FALSO, ou ainda expressões lógicas, cujo resultado seja ou FALSO ou VERDADEIRO.

Como declarar variáveis?

Para criar o nome de uma variável, precisamos seguir algumas regras. São elas:

REGRA	EXEMPLO
Não inicie com número;	1NUM
Não utilize caracteres especiais.	1ºNum; Nome(M); N*B
Não coloque espaços em branco ou hífen entre nomes.	B Letra B-Letra
Utilize, se necessário, *underline* (ou *underscore*)	B_Letra
Crie suas variáveis com nomes sugestivos (não é uma regra, mas é bom seguir).	Se vai guardar nome de funcionários, crie a variável utilizando a palavra NOME.

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

Conceitos

Anotações

Fala Professor

Olá,

As atividades aqui propostas não são avaliativas. Sugiro, entretanto, que sejam realizadas sem consultas, dessa forma poderemos acompanhar o desenvolvimento.

Bom Estudo!!!

Atividades

3. Aprendemos que, ao declarar uma variável precisamos seguir algumas regras. Assinale, as variáveis que obedecem às regras estabelecidas:

a) ( ) NUM1
b) ( ) 1 NUM
c) ( ) 1ºNUM
d) ( ) SEXO
e) ( ) TELEFONE
f) ( ) Nome
g) ( ) R-2
h) ( ) Rua
i) ( ) P_Num
j) ( ) Num_RUA
k) ( ) Certo
l) ( ) MATR

Atividades

4. Para cada valor apresentado abaixo, foi definido um tipo de variável. Marque com X os tipos que foram definidos corretamente:

a)( ) valor= 1.0 tipo= real
b)( ) valor= 23 tipo= real
c)( ) valor= -3 tipo= real
d)( ) valor= -3.0 tipo= real
e)( ) valor= 1000 tipo = inteiro
f)( ) valor= 54 tipo = inteiro
g)( ) valor= -54 tipo = inteiro
h)( ) valor= 54,0 tipo = inteiro
i)( ) valor= 'F' tipo = cadeia
j)( ) valor= 'M' tipo = caract
k)( ) valor= "rua" tipo = cadeia
l)( ) valor= "*" tipo = caract

1.3.3 Constantes

Sintaxe: ***const identificador = valor;***

As constantes são criadas com base nas mesmas regras e tipos já vistos em variável. Diferem apenas no fato de armazenar um valor constante, ou seja, que não se modifica durante a execução de um programa.

Se você entendeu como declarar uma variável, ficará fácil entender como se cria uma constante.

Vamos tomar como exemplo a folha de pagamento dos funcionários de uma determinada empresa:

- Se assumirmos que a empresa concederá um aumento de 10% aos seus funcionários independentemente do valor do salário, a taxa de 10% será uma constante, durante a execução do programa que efetuará o cálculo.
  Nossa constante ficaria assim declarada: const TAXA= 0.10;

- O nome da empresa também será uma constante, ele será impresso em todos os contra-cheques, teremos mais uma constante.
  E a declararíamos assim: const NOME;
  NOME = "Empresa X";

Conceitos

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

Anotações

Atividades

5. Identifique o tipo de cada uma das constantes abaixo:

Exemplo: a) 35 [*constante tipo inteiro*]

b) 21000 [ ] f) 0,10 [ ]

c)“falso” [ ] g) “Ifes” [ ]

d) TRUE [ ] h) 2007 [ ]

e) FALSO [ ] i) “verdadeiro” [ ]

Fala Professor

O que aprendemos até aqui?

- *Algoritmos são os passos que levam à solução de um problema.*
- *As variáveis armazenarão valores numéricos reais e inteiros, valores literais caractere e cadeia e valores lógicos identificados no Algoritmo.*
- *O conteúdo de uma variável pode variar durante a execução do programa.*
- *A diferença entre constante e variável é que o conteúdo de uma constante não varia durante a execução do programa.*
- *Ao fazermos declarações de variáveis e de constantes temos que seguir regras.*

1.4 Operadores

Os operadores são símbolos que representam uma operação aritmética ou lógica.

1.4.1 Operadores Aritméticos

Os operadores aritméticos são empregados em expressão aritmética em que são utilizados constantes ou variáveis do tipo real ou inteira, como operandos. Vamos ver abaixo a ordem de precedência:

adição	+
subtração	-
multiplicação	*
divisão	/

1.4.2 Operadores Relacionais

Os operadores relacionais realizam comparações entre variáveis. São eles:

igual	=
maior	>
menor	<
maior ou igual	>=
menor ou igual	<=
não igual (ou diferente)	!=

1.4.3 Operadores Lógicos

Os operadores lógicos retornam Falso(F) ou Verdadeiro(V) , de acordo com seus operandos. São eles:

Operadores
E
OU
NÃO

Entendendo:

Vamos assumir que existam duas condições para o resultado favorável de uma prova:

1ª proposição: O aluno estudou.
2ª proposição: O aluno colou.

- Chamaremos a 1ª proposição de P.
- E chamaremos a 2ª proposição de Q.

Vamos verificar na tabela abaixo qual resposta teremos na 3ª coluna se o operador lógico utilizado entre as proposições for o **OU**:

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

Observe que o aluno só não terá resultado favorável se as duas proposições forem falsas, ou seja: Se o aluno não estudar **OU** se o aluno não colar.

Agora vamos verificar como se comporta a 3º coluna se o operador lógico utilizado for o **E**:

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

Nesse caso, para tirar uma nota favorável, o aluno, além de estudar, terá de colar.

Abaixo, a tabela para o operador lógico **NÃO**:

P	Não P
V	F
F	V

Ordem de precedência das operações:

Prioridade	Operador
1ª	aritmético
2ª	relacional
3ª	lógico - não
4ª	lógico - e
5ª	lógico -ou

1.5 Expressão Aritmética e Expressão Lógica

1.5.1 Expressão Aritmética

Expressão aritmética é a que utiliza operandos que sejam constantes ou variáveis, desde que sejam do tipo real ou inteiro e operadores aritméticos, os quis estudamos no item 1.4.

Exemplo: x + y

Ordem de precedência das operações:

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

Utilizando vários níveis de parênteses, quebraremos a prioridade e obteremos uma seqüência de cálculo diferente.

1.5.2 Expressão Lógica

Expressão lógica utiliza operandos que sejam constantes e/ou variáveis numéricas, literais ou lógicas e os operadores lógicos.

Exemplos: (x < y) e (y < z)
(y > t) ou verdade

Observe que nos exemplos acima o resultado obtido é sempre VERDADEIRO OU FALSO. Numa expressão lógica, sempre obteremos o resultado V ou F.

Exemplo: Dadas as variáveis e as seguintes atribuições:

var inteiro NUM1=10;
var inteiro NUM2= 5;
var inteiro NUM3=200;
var inteiro NUM4=200;

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

Vamos analisar todas as etapas necessárias:

1. NUM1+ NUM2 > NUM1 é o mesmo que (10+5>10), resposta **V**, já que 10+5 é maior que 10.

2. NUM1+ NUM3 > NUM4 é o mesmo que (10+200>200), resposta **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**

Anotações

Fala Professor

Olá,

É importante que as atividades abaixo sejam realizadas na ordem proposta, já que o grau de dificuldade vai crescendo na mesma ordem.

Bom estudo!!

Atividades

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

var inteiro NUM1=10;
var inteiro NUM2= 5;
var inteiro NUM3=200;
var inteiro NUM4=200;

6. Coloque F ou V nas expressões abaixo: **Exemplo**: (F) NUM4 > NUM3;

a) ( ) NUM1 > NUM2;

b) ( ) NUM1 < NUM3;

c) ( ) NUM1 < NUM4;

d) ( ) NUM3 = NUM4;

7. 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;

8. 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;

Atividades

9. 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;

10. 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;

Fala Professor

O que aprendemos sobre os operadores?

- *São símbolos que representam uma operação aritmética ou lógica.*
- *Os operadores podem ser: aritméticos, relacionais e lógicos.*
- *Os aritméticos são empregados em expressão aritmética .*
- *Os relacionais verificam as relações entre os operadores.*
- *Os lógicos retornam Falso (F) ou Verdadeiro (V).*
- *A ordem de precedência das operações.*

1.6 Comandos de Atribuição, de Entrada, de Saída e Comentário

1.6.1 Comando de Atribuição

Na construção de nossos Algoritmos, precisamos constantemente indicar que uma variável ou uma constante criada por nós armazenará um determinado valor ou expressão, durante a execução do programa.

A forma correta de fazer essa indicação é por meio do comando de atribuição, representado por uma seta(←).

```
Assim:  var int NUM;              /*criamos a variável inteira NUM*/
        NUM ← 10;                 /*atribuímos o valor 10*/
        var caractere  SEXO;      /*criamos a variável caractere SEXO*/
        SEXO ← 'F';               /*atribuímos o caractere 'F'*/

        var real  SALÁRIO;        /*criamos a variável real SALÁRIO*/
        SALÁRIO← 22.000,00        /*atribuímos o valor 22.000,00*/
```

1.6.2 Comando de Entrada

Sintaxe: ***leia(variável);***

Os dados de que precisaremos para execução do nosso Algoritmo serão geralmente informados por meio do teclado.

O comando de entrada será responsável pela leitura e armazenamento desses dados na variável que indicarmos.

1.6.3 Comando de Saída

Sintaxe: ***escreva("O nome é:" , variável);***

O comando de saída será responsável pela exibição de mensagens, de valores processados ou de valores lidos, conforme indicarmos.

Exemplo:

```
Algoritmo primeiro

var cadeia  NOME;

{

  NOME ← "Paulo Vitor";

  escreva("Olá, " , NOME);

}
```

Olá, Paulo Vitor

Veja o resultado desse Algoritmo no quadro ao lado.

Observe que se o comando de saída fosse:

escreva("Olá, , NOME");

Olá, ,NOME

a saída seria exatamente o conteúdo entre as aspas.

Portanto, coloque fora das aspas tudo aquilo que deve ser resolvido pelo computador.

1.6.4 Comentários

Utilizaremos comentário para explicar detalhes que julgarmos necessários em nosso programa. Desta forma, aumentamos a legibilidade do nosso algoritmo. Lembre-se que não devemos fazer programas apenas para que o computador execute, mas também para outras pessoas possam entendê-lo e assim participarem da sua construção e/ou manutenção.

Exemplo:

```
var int NUM; /*criamos a variável inteira NUM*/
NUM ← 10;    /*atribuímos o valor 10*/
```

Observe que existem caracteres especiais(/*) que foram utilizados no início e no fim do comentário.

Também é possível utilizar os caracteres // (duas barras). Neste caso, ele é finalizado com um salto de linha.

Assim:

```
var int NUM;   //criamos a variável inteira NUM
NUM ← 10;      //atribuímos o valor 10
```

O comentário sempre será feito com base em um dos modelos acima. Faça-o sempre que você sentir necessidade de explicar alguma linha do seu Algoritmo.

Anotações

Fala Professor

Olá,

Gostaria de chamar a atenção para a importância de as atividades serem realizadas individualmente.

Bom estudo !!!

Atividades

11. Assinale os comandos de atribuição realizados corretamente:

a) ( ) **var** cadeia SEXO ← 'F';

b) ( ) **var** inteiro ALTURA ← 1,80;

c) ( ) **var** real SALÁRIO ← 3.000,00;

d) ( ) **var** cadeia ← "NOME";

12. 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 ... {

linha 3 ... **var** int NUM1, NUM2, SOMA;

linha 4 ... **escreva**("Digite o primeiro número");

linha 5 ... **leia**(NUM1);

linha 6 ... **escreva**("Digite o segundo número");

linha 7 ... **leia**(NUM2);

linha 8 ... SOMA ← NUM1 + NUM2;

linha 9 ... **escreva**("A soma dos números digitados é:", SOMA);

linha 10 ... }

a) ( ) linha 5 → 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 8 → O resultado da soma dos valores digitados está sendo atribuído à variável SOMA;

d) ( ) linha 9 → No monitor serão exibidas a mensagem que está entre aspas e a soma dos números digitados;

Atividades

13. 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 média;

linha 2 ... {

linha 3 ... **var** _____NOTA1, NOTA2, MÉDIA;

linha 4 ... _____ (“Digite a primeira nota”);

linha 5 ... _____(NOTA1);

linha 6 ... **escreva**(“_____”);

linha 7 ... **leia**(_____);

linha 8 ... MÉDIA ← (_____ + _____)/2;

linha 9 ... **escreva**(“A média das notas é:” , _____);

linha 10 ... }

14. Faça o mesmo no Algoritmo abaixo, cuja finalidade é calcular 8% de aumento sobre um salário. linha 1 ... Algoritmo reajuste;

linha 2 ... {

linha 3 ... **var** _____ SALARIO, SALARIO_NOVO;

linha 4 ... _____ (“Digite o salário”);

linha 5 ... _____(SALARIO);

linha 6 ... SALARIO_NOVO ← _____* 1.08;

linha 7 ..._____(“O valor do novo salário é:” , _____);

linha 8 ... }

Fala Professor

O que aprendemos sobre os comandos de atribuição, de entrada e de saída?

- *O comando de atribuição é responsável por atribuir valores à variável.*
- *O comando de entrada faz a leitura do dado digitado no teclado e o armazena na variável.*
- *O comando de saída é responsável por exibir os dados.*
- *O comentário é utilizado para explicar detalhes que julgarmos necessários em nosso programa.*

- **atribuição**: Consiste em atribuir um valor a uma variável previamente declarada.(LAUREANO, 2005, p.27).
- **entrada**: Mecanismo de entrada consiste em ler caracteres da entrada-padrão, normalmente teclado. (SCHILDT, 1995, p.15).
- **saída**: Mecanismo de saída consiste em escrever caracteres da saída-padrão, normalmente monitor. (SCHILDT, 1995, p.15).
- **cometários**: Texto que não é interpretado pelo compilador. (LAUREANO, 2005, p. 6).

1.6.5 Como Construir Algoritmo

Vamos criar um Algoritmo para ler e multiplicar dois números inteiros e exibir o resultado.

É importante observar cada linha dessa seqüência.

```
linha 1   Algoritmo multiplicação;
linha 2   {
linha 3      var inteiro NUM1, NUM2, MULT;
linha 4      escreva ( "Digite o primeiro número");
linha 5      leia (NUM1);
linha 6      escreva ( "Digite o segundo número");
linha 7      leia (NUM2);
linha 8      MULT ← NUM1 * NUM2;
linha 9      escreva("O resultado da multiplicação é:",MULT);
linha 10  }
```

Vamos entender todas as linhas do nosso Algoritmo:

linha 1 ... Nome do programa.

linha 2 ... A chave indica o início do programa.

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

linha 4 ... O comando ***escreva*** exibirá a mensagem que solicita a digitação do primeiro número.

linha 5 ... O primeiro número digitado será lido e armazenado na variável NUM1.

linha 6 ... O comando ***escreva*** exibirá a mensagem que solicita a digitação do segundo número.

linha 7 ...O segundo número digitado será lido e armazenado na variável NUM2.

linha 8 ...A variável MULT receberá o resultado da multiplicação do primeiro pelo segundo número.

linha 9 ...O comando ***escreva*** exibirá uma mensagem com o resultado da multiplicação.

linha 10...A chave indica o fim do programa.

Atividades

15. Construa um Algoritmo que efetue o cálculo do salário bruto de um funcionário, considerando que SALARIO BRUTO = HORAS TRABALHADAS * VALOR HORA.

16. Faça um Algoritmo que leia o valor da cotação do dólar (dia), multiplique pelo valor (em dólar) de um determinado produto e imprima o valor convertido.

Atividades

17. O custo ao consumidor de um carro novo é a soma do custo de fábrica com a porcentagem do distribuidor e dos impostos (aplicados ao custo de fábrica). Supondo que a porcentagem do distribuidor seja 12% e a dos impostos 45%, prepare um Algoritmo para ler o custo de fábrica do carro e imprimir o custo do carro para o consumidor.

18. Uma empresa tem fichas que contém o nome, o número de horas trabalhadas e o número de dependentes de cada um de seus funcionários.

Considerando que:

- a empresa paga 12 reais por hora trabalhada e 40 reais por dependente;
- desconta do salário 8,5% para o INSS e 5% para IRRF.

Atividades

Faça um Algoritmo para ler o nome, o número de horas trabalhadas e o número de dependentes de um funcionário.

No final, o programa deverá exibir o nome, o salário bruto, os valores descontados para cada tipo de imposto e o salário líquido do funcionário.

Atividades

19. Dadas as seguintes informações de um funcionário: nome, idade, cargo e seu salário bruto, considere que:

- o salário bruto teve um reajuste de 38%.
- o funcionário receberá uma gratificação de 20% do salário bruto.
- 15% serão descontados do salário total.

Faça um Algoritmo para imprimir nome, idade, cargo, salário bruto e salário líquido.

Anotações

Indicações

Leituras complementares:

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

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

CAPÍTULO 2

LINGUAGEM C

Caro aluno,

Fala Professor

Vamos iniciar o segundo Capítulo da nossa disciplina. Nessa etapa, é importante que as atividades sejam feitas no computador. Desenvolveremos programas que nos ajudarão a melhorar nosso raciocínio lógico e a nossa agilidade na solução de problemas de forma que o computador seja capaz de interpretar.

Se você ainda não fez o download do ambiente Bloodshed Dev-C++, este é o momento.

Vamos em frente!!!!

A partir desta aula, vamos construir Algoritmos de forma que o computador possa entender e executar. Para isso, vamos conhecer a linguagem de programação C.

Conceitos

- **linguagem de programação:** Uma Linguagem de Programação é um método padronizado para expressar instruções para um computador. (LAUREANO, 2005, p. 4).
- **programas:** Um programa de computador é uma coleção de instruções que descrevem uma tarefa a ser realizada por um computador. (LAUREANO, 2005, p. 4).
- **código fonte:** Uma linguagem de programação é um conjunto de ferramentas, regras de sintaxe e símbolos ou códigos que nos permitem escrever programas de computador, destinados a instruir o computador para a realização de suas tarefas. (LAUREANO, 2005, p. 5).

Anotações

Para compilar nossos programas, usaremos o ambiente Bloodshed Dev-C++, disponível gratuitamente no link http://superdownloads.uol.com.br/download/199/bloodshed-dev-c/, ou no ambiente Moodle.

2.1 As Telas do Bloodshed Dev-C++

1º passo – janela 1

Assim que entrarmos no ambiente Dev C++, a tela abaixo(Ilustração 1) será a primeira a que teremos acesso.

Clique o botão <Fechar> da janela "Dica do dia".

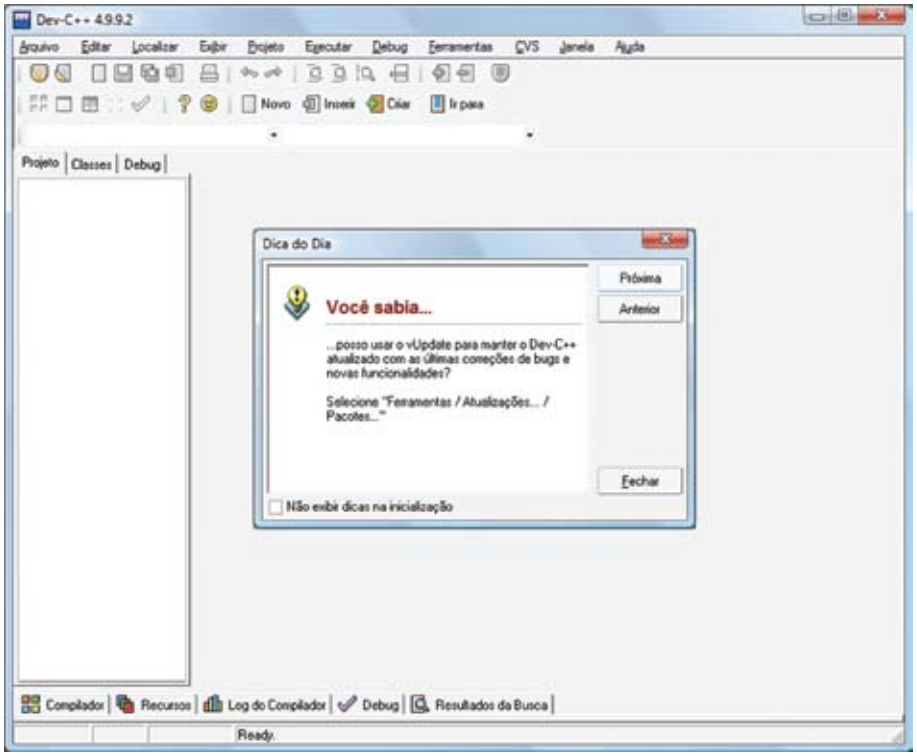


Ilustração 1: Apresentação da primeira tela

Anotações

2º passo – janela 2

Clique em meu Arquivo>Novo>Projeto, como apresentado na Ilustração 2:

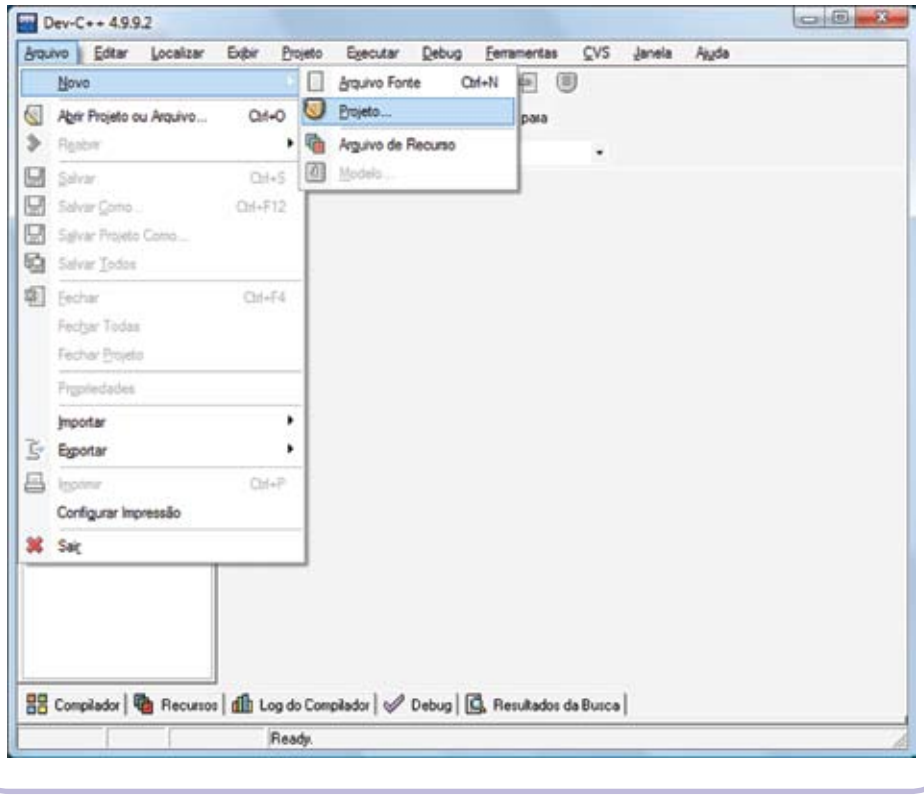


Ilustração 2: Apresentação da segunda tela

Anotações

3º passo – janela 3

Clique no ícone <Console Aplication>, no botão <Projeto C> e finalmente no <OK>. Observe a Ilustração 3, ela o ajudará a executar esses passos:

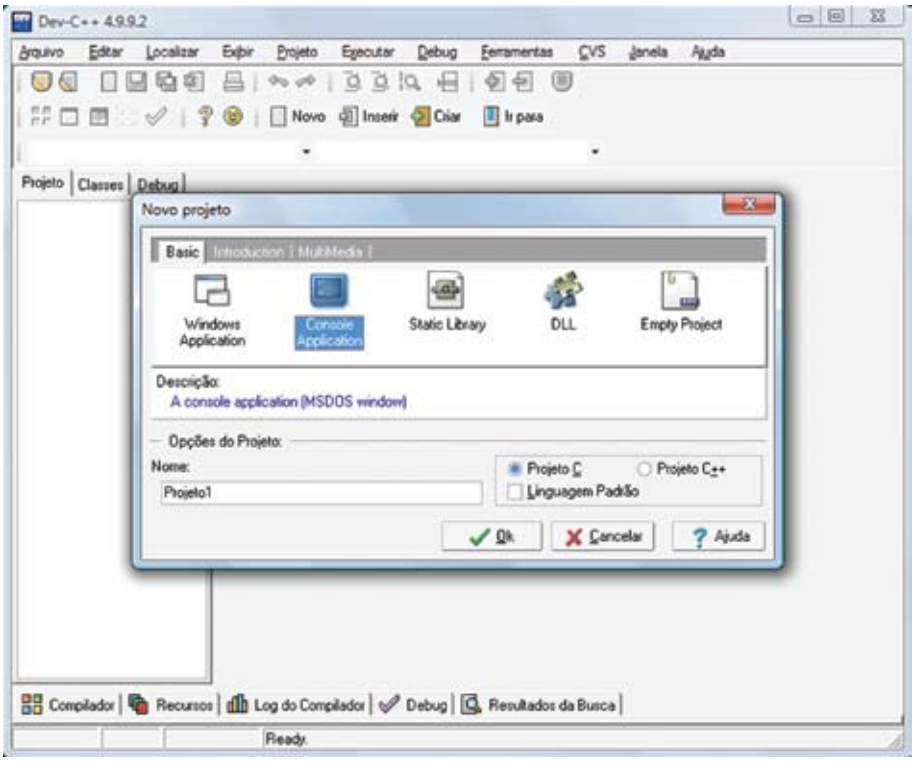


Ilustração 3: Apresentação terceira tela

Anotações

4º passo – janela 4

A janela Create new project se abrirá para iniciarmos nosso projeto, conforme Ilustração 4.

No campo <nome do arquivo>, digite um nome para seu arquivo e clique em <Salvar>.

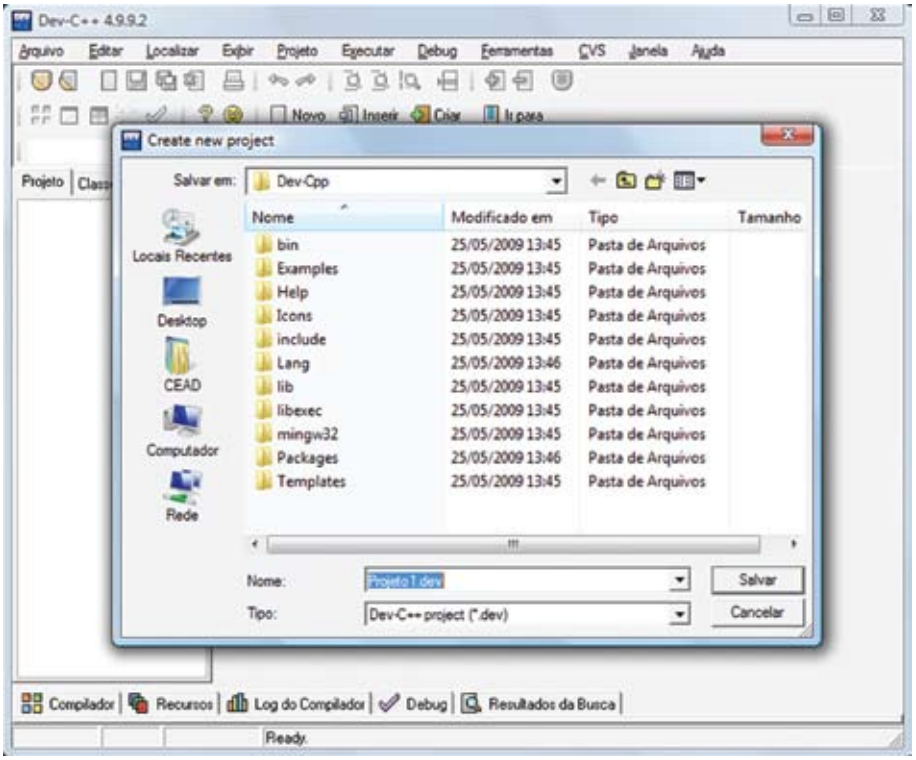


Ilustração 4: Apresentação da quarta tela

Anotações

5º passo – janela 5

É nessa janela que vamos digitar nosso código, conforme indica a Ilustração 5.

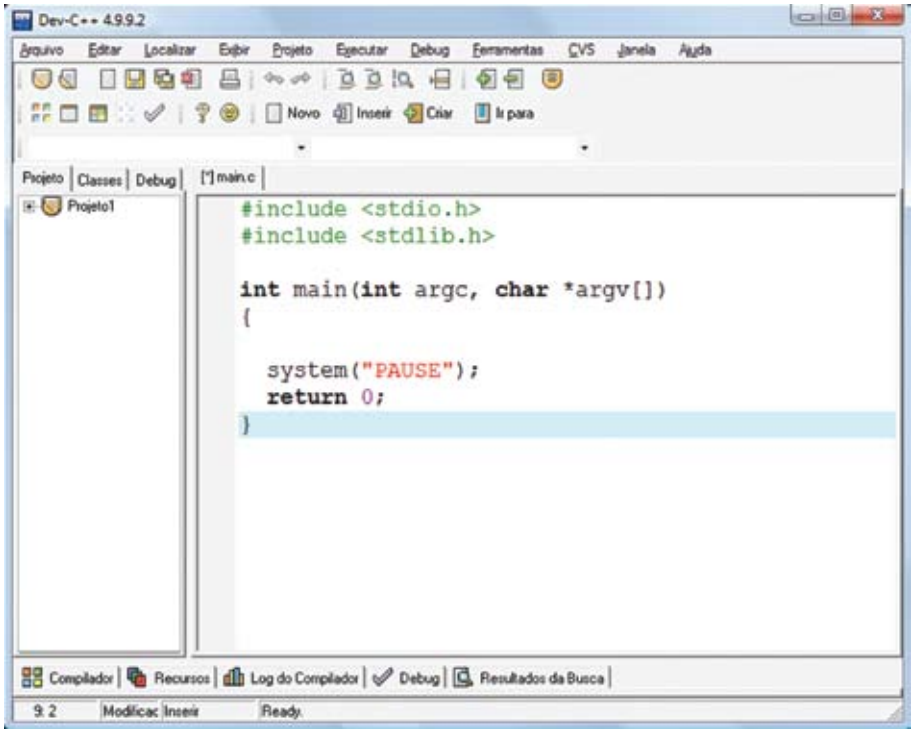


Ilustração 5: Apresentação da quinta tela

Neste primeiro momento, não exploraremos todos os menus disponíveis no ambiente Bloodshed Dev-C+. A exposição dos seus recursos se dará à medida que avançarmos nos nossos estudos.

Vamos em frente!!!

Fala Professor

2.2 Visão Geral da Linguagem C

Para termos uma visão geral da linguagem que usaremos no desenvolvimento dos programas, vamos analisar como ficaria nosso Algoritmo Soma, na Linguagem C:

Exemplo1:

as mudanças de estado do sistema.

Linguagem – Portugol	**Linguagem – C**
linha 1 Algoritmo soma;	linha 1 #include <stdio.h>
linha 2 {	linha 2 #include <stdlib.h>
linha 3 var int NUM1, NUM2, SOMA;	linha 3 int main( )
linha 4 escreva (“Digite o primeiro número”);	linha 4 {
linha 5 leia (NUM1);	linha 5 int num1, num2, soma;
linha 6 escreva (“Digite o segundo número”);	linha 6 printf(“Digite o primeiro numero: “);
linha 7 leia (NUM2);	linha 7 scanf (“%d”, &num1);
linha 8 SOMA ← NUM1 + NUM2;	linha 8 printf(“Digite o segundo numero: “);
linha 9 escreva (“A soma é:”, SOMA);	linha 9 scanf (“%d”, &num2);
linha 10 }	linha 10 soma = num1 + num2;
	linha 11 printf(“A soma e: %d \n”, soma);
	linha 12 system(“PAUSE”);
	linha 13 return(0);
	linha 14 }

2.3 Comandos da Linguagem de Programação C

Vamos entender cada uma das 14 linhas que compõem o programa Soma, que foi criado na linguagem de programação C. É importante atentarmos para o fato de que no código do programa Soma não utilizamos nenhum tipo de acentuação.

Ao salvar seu programa, siga algumas regras: não utilize acentos e nem deixe espaços entre palavras.

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

As linhas indicam a inclusão de biblioteca que possui as funções de entrada e saída de dados, necessárias à execução do nosso programa Soma. Veremos mais adiante que outras bibliotecas serão necessárias; quando isso acontecer, vamos incorporá-las.

- **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( ), em alguma parte do programa.

- **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 as variáveis necessárias à execução do programa. Iniciamos a declaração informando que as variáveis seriam do tipo inteiro(int). Observe a existência de um ponto-e-vírgula; seu emprego indica o final do comando.

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

A função printf( ) é uma função de entrada e saída. Permiti que a mensagem entre aspas seja exibida no monitor.

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

A função scanf( ) é responsável por ler os dados que forem informados pelo teclado. Nessa linha a função lerá o primeiro número que for digitado e o armazena no endereço da variável num1, conforme indicado (“%d”, &num1).

- **A décima linha** - soma = num1 + num2;

O comando de atribuição (=) atribui o resultado da soma dos valores contidos no endereço de num1 e num2.

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

Já vimos que a função printf( ) permite a exibição da mensagem 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 o %d.
O código especial \n é responsável por fazer saltar uma linha.

- **A décima segunda** - system(“PAUSE”);

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

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

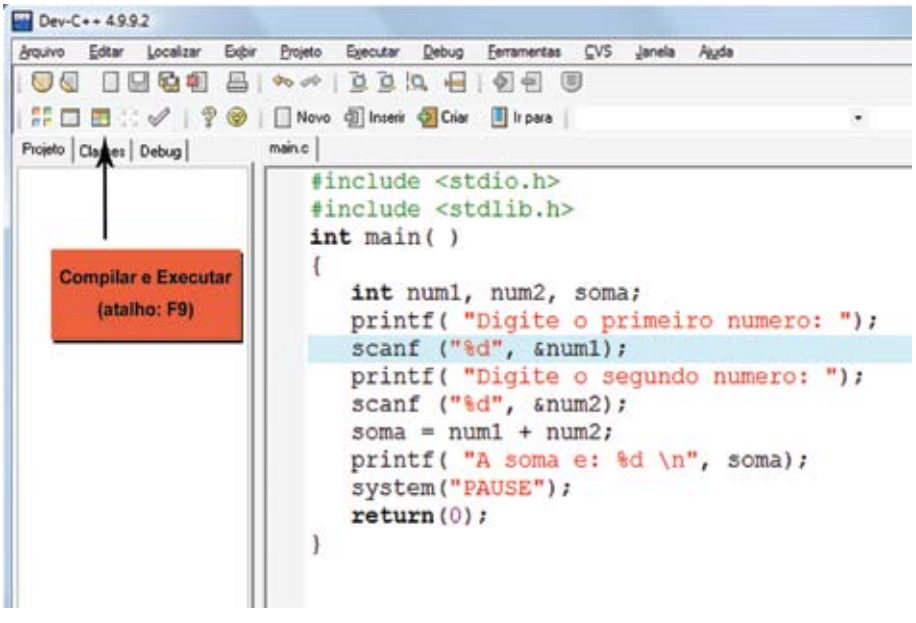
Indica o número inteiro que está sendo retornado pela função, em nosso caso, o número zero. O comando return (0) será detalhado adiante.

- **A décima quarta linha** - }

Indica o fim do programa. O fim de main( ).

EXECUÇÃO DO PRIMEIRO PROGRAMA APRESENTADO

Antes de executar o programa é necessário traduzí-lo para uma linguagem em que o computador possa entender, isto é, fazer a sua compilação. Use a tecla F9 ou o botão indicado na ilustração 6.



lustração 6: Compilação e Execução

Na primeira vez é solicitado o salvamento do arquivo: informe um nome qualquer para o arquivo, desde que tenha a extensão ".c"

A compilação também é o momento em que os erros de sintaxe (grafia) são apresentados.

Veja na tabela abaixo alguns exemplos de erro de digitação e suas respectivas mensagens de alerta.

Faça estas simulações de erro para entender melhor o processo e se familiarizar com as mensagens do ambiente.

Experimente também compilar com outros erros.

Depois desta vivência você ficará mais preparado para enfrentar situações reais de erro.

Linha original	Como foi digitado	Mensagem de erro	Explicação da mensagem de erro
#include <stdio.h>	#include <stdioh>	Linha 1: stdioh: No such file or directory.	Sem ponto o nome do arquivo não foi encontrado.
int num1, num2, soma;	int num1, num2, soma,	Linha 7: syntax error before string constant	Sem o ponto-e-vírgula o compilador deduz que o comando ainda não terminou e por isso vai acusar erro na linha posterior, quando o código já não faz mais sentido. Outros erros em linhas mais abaixo são indicados em função deste primeiro erro.
printf("Digite o primeiro numero: ");	printe("Digite o primeiro numero: ");	undefined reference to `printe'	Não foi encontrada a função 'printe' nas bibliotecas indicadas
scanf ("%d", &num2);	scanf ("%d, &num2);	Linha 10: missing terminating " character	Falta o caractere aspas como terminador.
return (0);	return 0);	Linha 14: syntax error before ')' token	Acusou erro antes do token ')'. Token é como o compilador chama elementos individuais do programa.

Para acertar o erro faça a edição diretamente no ponto desejado e re-compile: CTRL F9

Quando não houver mais erros será mostrada a tela da ilustração 7.

Ilustração 7: Compilação sem erros

Depois de compilado o programa e execução é feita logo em seguida.

Observe o resultado na ilustração 8.

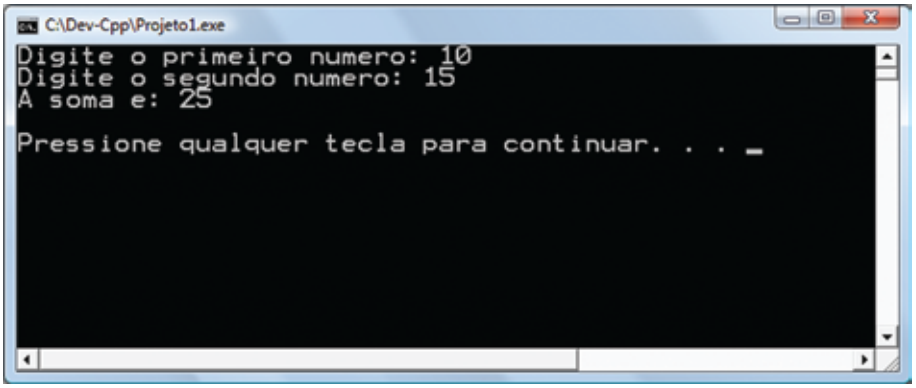


Ilustração 8: Compilação e execução do código fonte de Soma

Fala Professor

Olá,

As explicações dadas na sequência do conteúdo serão acompanhadas de exemplos que você deverá digitar, compilar e executar no programa 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.*

Não avance se as dúvidas permanecerem.

Bom estudo !!

2.4 Constantes e Variáveis na Linguagem de Programação C

Já aprendemos que uma constante tem o valor fixo e que uma variável que pode conter, a cada tempo, valores diferentes. Vejamos um exemplo de declaração de variável em C.

Exemplo 2: Neste exemplo temos uma variável do tipo inteira, que armazenará o valor digitado pelo usuário e exibirá em seguida esse valor.

```
#include <stdio.h>
#include <stdlib.h>
int main( )
{
     int num;
     printf ("Digite um numero: ");
     scanf("%d", &num);
     printf ("O numero digitado foi: %d \n",num);
     system("pause");
     return(0);
}
```

A Ilustração 9 vai nos mostrar o resultado da compilação e da execução desse programa.

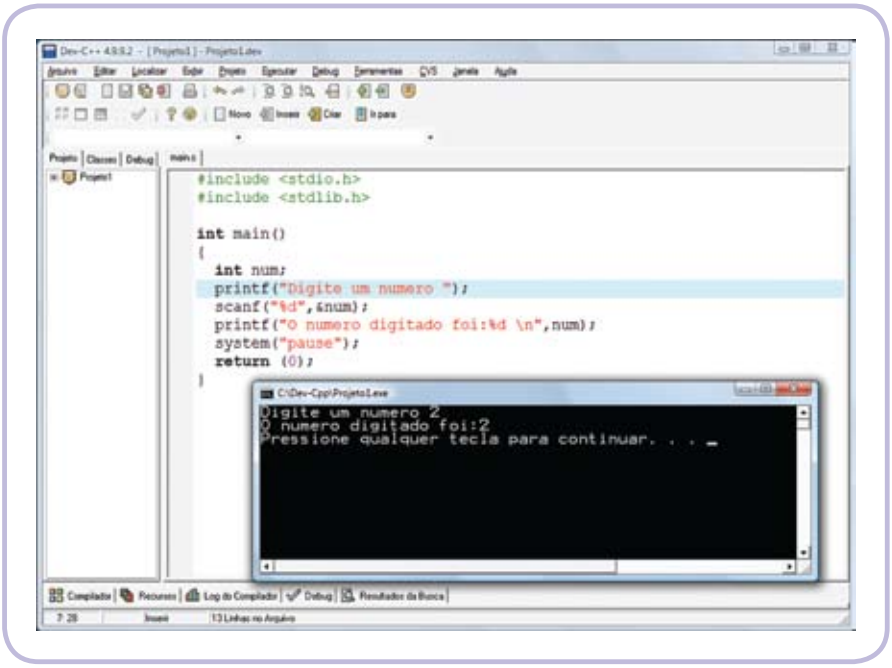


Ilustração 9: Compilação e execução do código fonte de Exemplo 2

2.4.1 Tipos de Variáveis na Linguagem de Programação C

O tipo de variável tem o objetivo de informar a quantidade de memória, que ocupará em bytes. Inicialmente, veremos 3 tipos de variáveis:

TIPO	BYTES
char	1
int	2
float	4

Exemplo 3:

```
#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 média = %f \n\n", media_final);
    system("pause");
    return(0);
}
```

A Ilustração 10 vai nos mostrar o resultado da compilação e da execução desse programa.

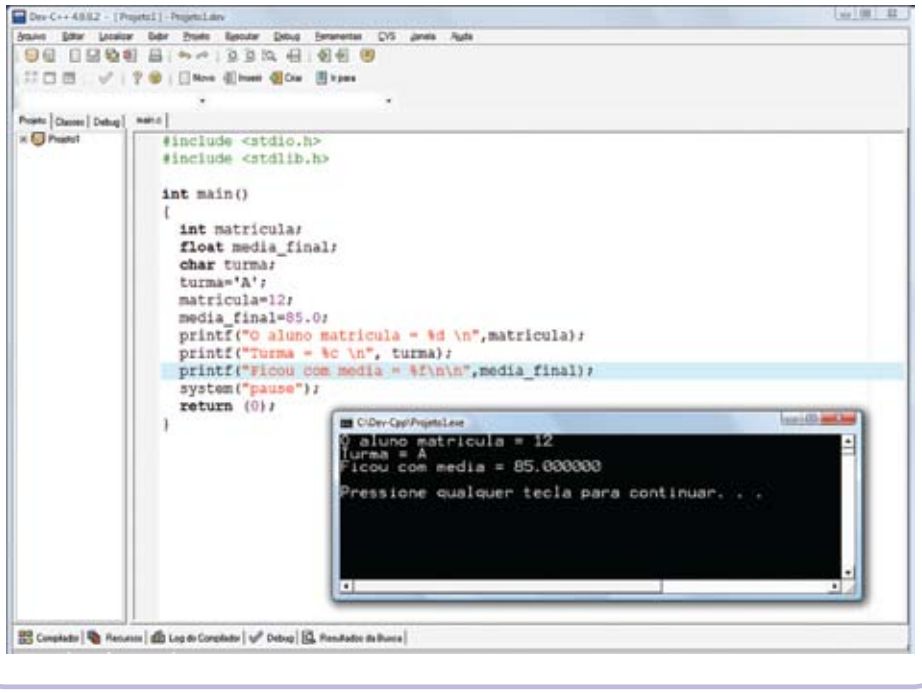


Ilustração 10: compilação e execução do código fonte do Exemplo 3.

2.5 Alguns dos Códigos para Impressão Formatada de Printf( )

Como já vimos, a função printf( ) usa o caractere de percentual (%) seguido de uma letra para identificar o formato de impressão.

No exemplo acima, utilizamos 3 tipos de caracteres na função printf( ). Na tabela abaixo estão relacionados os caractere com tipo de variável a ser impresso.

CÓDIGO	SIGNIFICADO
%c	usado quando a função for exibir apenas um caractere (tipo char). Exemplo: 'M'
%f	usado quando a função for exibir número com ponto flutuante, ou seja, um número real com possibilidade de casas decimais (tipo float). Exemplo: 1,80
%s	usado quando a função for exibir uma cadeia de caracteres, ou seja, uma ou várias palavras (tipo char[ ]). Exemplo: "azul"
%d	usado quando a função for exibir um número inteiro na base decimal (tipo int). Exemplo: 25

2.6 Códigos Utilizados Pela Função Scanf( )

CÓDIGO	FUNÇÃO
%c	usado quando a função for armazenar um caractere (tipo char).
%d	usado quando a função for armazenar um número inteiro (tipo int).
%f	usado quando a função for armazenar um número real, com possibilidades de casas decimais (tipo float).
%s	usado quando a função for armazenar uma cadeia de caracteres, ou seja, uma ou várias palavras (tipo char[ ]).

Anotações

Agora, volte ao código fonte do Exemplo 3, revise a função printf( ) e a função scanf( ), confrontando os caracteres utilizados no exemplo com os apresentados nas tabelas.

Não siga adiante sem concluir essa verificação, pois as funções serão empregadas constantemente em nossos exercícios. Compreendê-las é de suma importância.

Bom estudo !!

2.7 Como fazer Comentários

O comentário deve ser feito exatamente como aprendemos em Algoritmos. Veja como ele foi colocado no programa abaixo:

Exemplo 4:

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

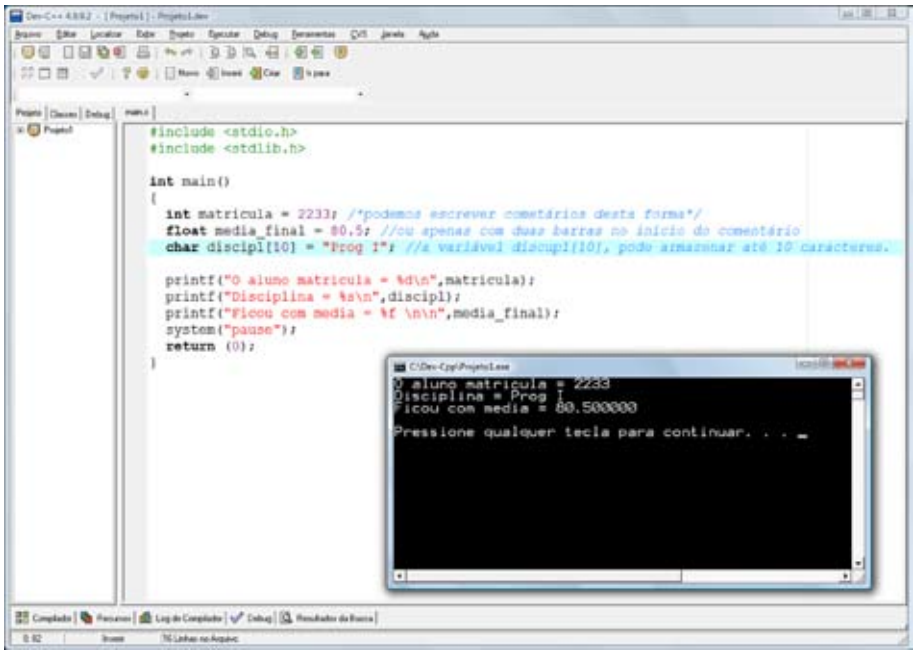


Ilustração 11: Compilação e execução do código fonte de Exemplo 4

Observe que o comentário só apareceu no código fonte.

A tela preta mostra a execução do programa sem o comentário.

Fala Professor

Vamos ver como ficam os operadores lógicos na linguagem C^?

Portugol	C
E	&&
OU	||
NÃO	!

Fala Professor

Anotações

Atividades

20. 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) multiplique os valores e guarde o resultado numa variável chamada **MULT;**

f) exiba o resultado.

Obs.: Caso encontre dificuldades para fazer esse programa, consulte o exemplo ***Soma*** (primeiro exemplo deste capítulo).

Atividades

21. Faça um programa que:

a) peça ao usuário para digitar um número inteiro;

b) armazene esse número numa variável chamada **A**;

c) peça ao usuário para digitar outro número inteiro;

d) armazene esse número numa variável chamada **C**;

e) exiba os valores digitados.

Obs.: Caso encontre dificuldades para fazer este programa, consulte o **Exemplo 2**, visto no item 2.4.

Fala Professor

O que aprendemos até aqui?

Transformar Algoritmo escrito na linguagem natural para a Linguagem C.

- *Declarar variáveis.*
- *Fazer cometários em nosso códigos.*
- *Compilar e executar exemplos apresentados.*

Além disso, conhecemos as particularidades das funções printf e scanf e obtivemos uma visão geral da Linguagem C.

2.8 Comandos de Seleção

No desenvolvimento de um programa, podemos nos deparar com várias condições que acarretaram processamentos diferenciados, ou seja, muitas vezes uma parte do nosso programa só é executada se a condição para essa execução for verdadeira.

Para testar essa condição, temos os comandos **if e switch.**

2.8.1 Comando If

Sintaxe: **if (condição) declaração**

O comando **if** será utilizado quando o programa ou parte dele necessitar de uma condição simples para sua execução.

Vamos ver um exemplo em que o resultado da soma de dois números só seja exibido se for maior que 2.

Exemplo 5:

```
#include <stdio.h>
#include <stdlib.h>
main( )
{
    int n1,n2,s;
    printf( "Digite primeiro numero ");
    scanf("%d",&n1);
    printf( "Digite segundo numero ");
    scanf("%d",&n2);
    s=n1 + n2;
    if(s>2)        // O comando if verifica se a soma dos números digitados é maior que dois.
     printf("\O resultado da soma dos valores digitados e maior que dois: %d \n",s);
    system("pause");
}
```

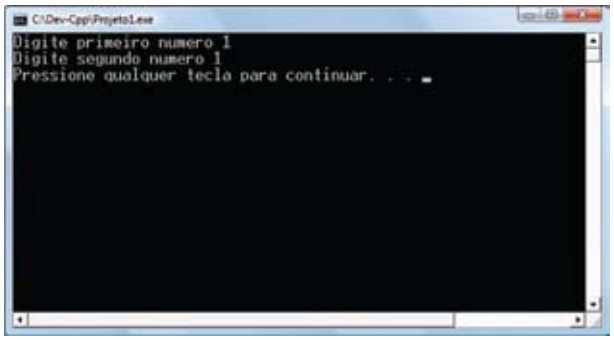


Ilustração 12: Demonstração do resultado se a soma for menor que 2.

Ilustração 13: Demostração do resultado se a soma for maior que 2.

Observe a Ilustração 13. O programa não mostrará nada se a soma dos dois números for menor que dois.

Apenas será exibida a mensagem default " pressione qualquer tecla para continuar...".

Fala Professor

Anotações

Exemplo 6:

Este programa indica se o usuário tem o voto obrigatório.

Para isso ele solicita a sua idade, em seguida verifica se esta idade é maior ou igual a 18 e ao mesmo tempo é menor ou igual a 70. Se sim, então exibe a mensagem de voto obrigatório.

```
#include <stdio.h>
#include <stdlib.h>
int main( )
{
    int idade;
    printf( "Entre com a sua idade: ");
    scanf ("%d", &idade);
    if(idade >= 18 && idade <= 70)
       printf( "O seu voto nas eleicoes e obrigatorio \n");
    system("pause");
    return (0);
}
```

Observe na ilustração 14 a saída na console para uma entrada com idade igual a 18.

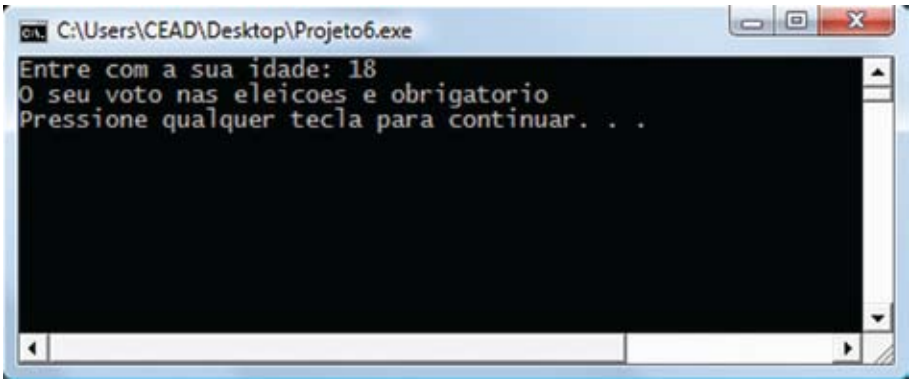


Ilustração 14: Demostração do resultado do exemplo6.

Anotações

Exemplo 7:

Este exemplo exibe o preço de um produto com o acréscimo de uma taxa de venda. Nem todos os produtos possuem esta taxa.

Observe que a entrada da taxa de venda é condicionada a resposta do usuário. Mas, em ambos os casos o programa exibe o preço final do produto, com ou sem acréscimo.

```
#include <stdio.h>
#include <stdlib.h>
int main( )
{
    float preco, taxa;
    char possuiTaxa;  // vamos assumir 's' para sim e 'n' para não
    printf("Entre com o preco do produto: ");
    scanf ("%f", &preco);
    printf("O produto possui taxa de venda (s/n)? ");
    scanf (" %c", &possuiTaxa);   // use espaço em branco antes do "%c"
    if(possuiTaxa == 's') {
       printf("Qual e o valor da taxa de venda? ");
       scanf ("%f", &taxa);
       preco = preco + taxa;
    }
    printf( "O preco do produto e: %f \n", preco);
    system("PAUSE");
    return (0);
}
```

Por que colocar espaço em " %c" ?

A função scanf obtém apenas os caracteres selecionados pelo "%". Desta forma, quando o processamento passa pelo "%f" na entrada do preço, apenas o número digitado será capturado e não o ENTER. Assim o ENTER colocado à disposição no buffer é obtido pelo próximo scanf("%c",&possuiTaxa), se escrito sem espaço. Por isso temos a falsa impressão de que o scanf não foi executado.

Observe na ilustração 15 a saída no console para uma entrada com preço igual a 200,00 e taxa de venda igual a 15,00

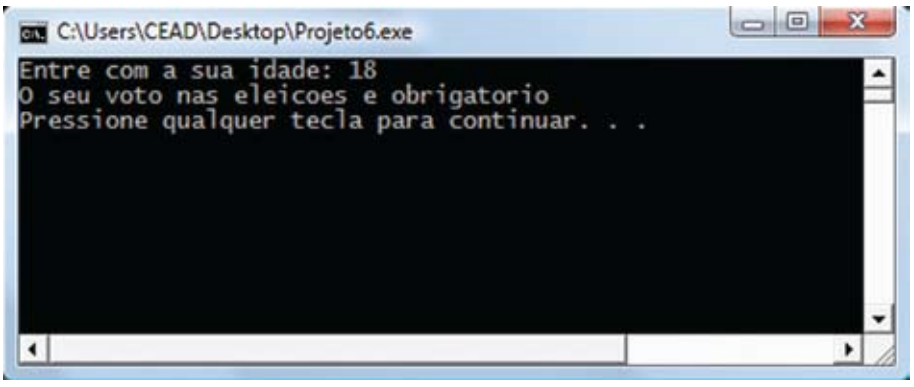


Ilustração 15: Demostração do resultado do exemplo6.

Anotações

Fala Professor

Olá,

Façamos as atividades propostas, a seguir, para verificar como estamos desenvolvendo o nosso aprendizado.

É importante que você digite os exemplos vistos, conforme orientação dada no início deste Capítulo.

Vamos em frente!!!

Atividades

22. 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".

Atividades

23. 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'.

24. Faça um programa que leia um número dado como entrada e apresente a mensagem " O número está na faixa correta", somente se o valor fornecido for entre 20 e 90.

Atividades

25. 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 10%. Em seguida exiba o salário, independente se ele sofreu reajuste ou não.

2.8.2 Comando If-Else

Sintaxe**: if (condição) declaração_1**
else declaração_2

O comando **if-else** permite maior agilidade ao programa, quando este for composto por duas ou mais condições, as quais deve ser avaliadas antes da execução das partes (bloco de comandos) que o compõem.

No exemplo visto no **comando if**, o programa só exibiria a mensagem quando o resultado fosse maior que 2. Para resultados menores que 2, nós não definimos nenhum bloco de comando; logo, o programa simplesmente não executaria nenhum comando.

Agora, definiremos um bloco de comandos para resultados menores que 2:

Exemplo 8:

```
#include <stdio.h>
#include <stdlib.h>
main( )
{
    int n1,n2,s;
    printf( "Digite primeiro numero ");
    scanf("%d",&n1);
    printf( "Digite segundo numero ");
    scanf("%d",&n2);
    s=n1 + n2;
    if(s>2)       // O comando if verifica se a soma dos números digitados é maior que dois.
     printf("\O resultado da soma dos valores digitados e maior que dois: %d \n",s);
    else
     printf("\O resultado da soma dos valores digitados e menor que dois: %d \n",s);
    system("pause");
}
```

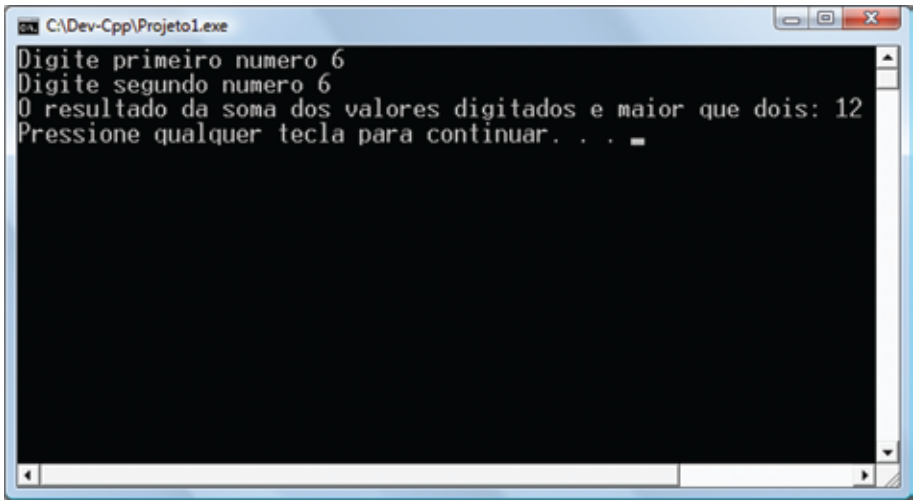


Ilustração 16: Demonstração do resultado se a soma for maior que 2.

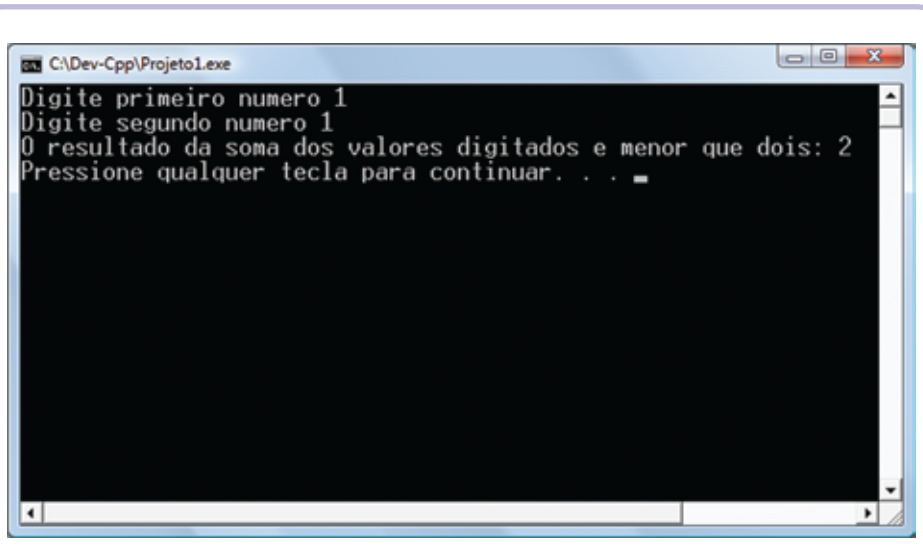


Ilustração 17: Demonstração do resultado se a soma for menor que 2.

Observe que agora definimos comandos específicos para resultados maiores que 2 e para resultados menores que 2, conforme demonstrado nas Ilustrações 16 e 17.

Exemplo 9:

Este exemplo complementa o exemplo 6 exibindo se o usário tem voto facultativo ou proibido.

Observe no primeiro 'if' que se a idade não estiver no intervalo de 18 a 70 o algoritmo verifica se a idade é menor do que 16 (pois se o processamento chegou até neste ponto é porque a idade é menor do que 18 ou maior do que 70) . Quando a idade for menor que 16 o voto é proibido. Caso contrário será facultativo, ou seja, 16, 17 ou maior do que 70.

```
#include <stdio.h>
#include <stdlib.h>
int main( )
{
    int idade;
    printf( "Entre com a sua idade: ");
    scanf("%d", &idade);
    if(idade >= 18 && idade <=70)        // O comando if verifica se a soma dos números digitados é maior que dois.
       printf( "O seu voto nas eleicoes e obrigatorio \n");
    else
       if(idade < 16)
          printf( "O seu voto nas eleicoes e proibido \n");
       else
          printf( "O seu voto nas eleicoes e facultativo \n");
    system("PAUSE");
    return (0);
}
```

Observe na ilustração 18 a saída no console para uma entrada com idade igual a 17.

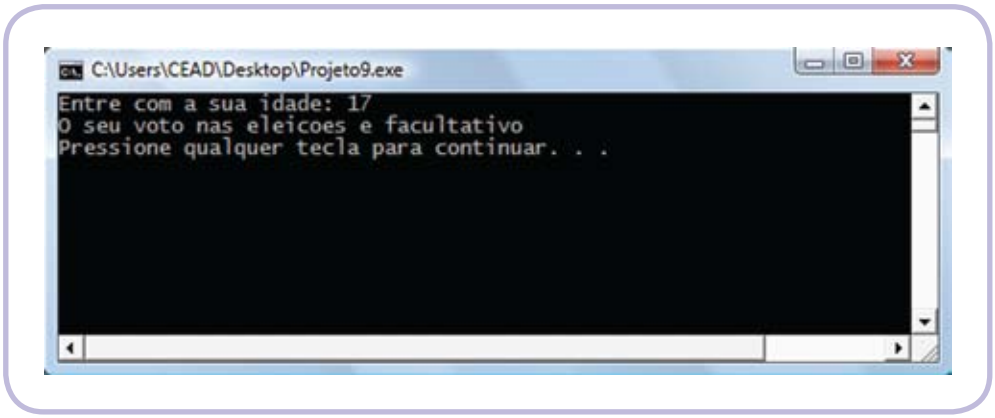


Ilustração 18: Demostração do resultado do exemplo 9.

Observe ainda que é possível escrever o comando else em conjunto com o próximo if de uma forma mais compacta. Veja o mesmo exemplo escrito desta forma, com else if ...

```c
#include <stdio.h>
#include <stdlib.h>
int main( )
{
    int idade;
    printf( "Entre com a sua idade: ");
    scanf("%d", &idade);
    if(idade >= 18 && idade <=70)
       printf( "O seu voto nas eleicoes e obrigatorio \n");
    else if(idade < 16)
       printf( "O seu voto nas eleicoes e proibido \n");
    else
       printf( "O seu voto nas eleicoes e facultativo \n");
    system("PAUSE");
    return (0);
}
```

Anotações

Exemplo 10:

Neste exemplo, o usuário responde o valor do salário e em seguida recebe uma classificação por nível , variando de 1 a 6. A primeira comparação verifica se o salário é negativo. Neste caso, o programa acusa erro.

```
#include <stdio.h>
#include <stdlib.h>
int main( )
{
    float salario;
    printf( "Qual e o valor do seu salario? ");
    scanf("%f", &salario);

    if(salario < 0)
       printf( "valor invalido! \n");
    else if(salario < 900)
       printf( "Salario nivel 1 \n");
    else if(salario < 1500)
       printf( "Salario nivel 2 \n");
    else if(salario < 3500)
       printf( "Salario nivel 3 \n");
    else if(salario < 6000)
       printf( "Salario nivel 4 \n");
    else if(salario < 10000)
       printf( "Salario nivel 5 \n");
    else // neste caso o salário é maior ou igual a 10 mil
       printf( "Salario nivel 6 \n");

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

Observe na ilustração 19 a execução do programa para um salário igual a 12 mil

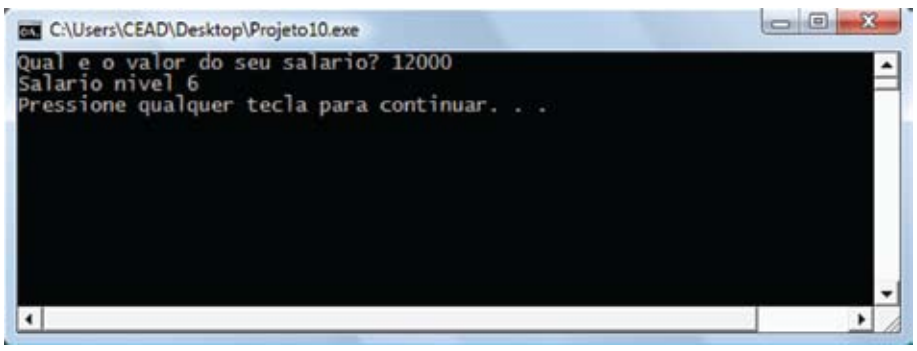


Ilustração 19: Demostração de uma execução do exemplo 10.

Anotações

Fala Professor

Vamos aproveitar os exercícios de 22 a 25 da seção anterior para realizar os exercícios desta seção.

Lembre-se de que as atividades devem ser feitas individualmente.

Bom estudo!!!

Atividades

26. Como complemento do exercício 22, o programa deverá exibir também a mensagem "Aluno reprovado", quando a nota final do aluno for menor que 60.

Atividades

27. Como complemento do exercício 23, o programa deverá exibir também a mensagem "Sexo inválido", se o caractere digitado for diferente de 'M' ou 'F'.

28. Como complemento do exercício 24, o programa deverá exibir também a mensagem " O número está na faixa incorreta", caso o valor fornecido não esteja entre 20 e 90.

29. Como complemento do exercício 25, o programa deverá aplicar também um aumento de 5%, se o salário for maior do que R$ 500,00.

2.8.3 Comando Switch

sintaxe:

```
switch (variável)
{
case constante 1 : declaração 1; break;
case constante 2 : declaração 2; break;
default:
declaração_default;
}
```

O comando ***switch*** é similar ao ***if-else***, mas não poderá ser usado quando a condição a ser testada for uma expressão.

Só é aceitável uma variável no ***switch***. Esta variável deve ser de algum tipo numérico inteiro, como por exemplo, int e char (char armazena o código ASCII do caracter, isto é, um número inteiro).

Exemplo 11:

Neste programa o usuário, além de digitar 2 números inteiros, poderá escolher a operação a ser realizada.

Este exemplo não condiz com o código!!!

```
#include <stdio.h>
#include <stdlib.h>
int main( )
{
    int n1,n2,opcao;
    printf( "Digite primeiro numero ");
    scanf("%d",&n1);
    printf( "Digite segundo numero ");
    scanf("%d",&n2);
    printf( "\ .....Escolha......\n");
    printf( "1=Somar \n");
    printf( "2=Subtrair \n");
    printf( "3=Multiplicar \n");
    printf( "4=Dividir \n");
    scanf("%d",&opcao);
    switch(opcao)
    {
      case 1: printf("O resultado e: %d \n", n1 + n2);break;
      case 2: printf("O resultado e: %d \n", n1 - n2);break;
      case 3: printf("O resultado e: %d \n", n1 * n2);break;
      case 4: printf("O resultado e: %d \n", n1 / n2);break;
      default: printf("Opcao invalida, tente novamente\n");break;
    }
    system("pause");
    return (0);
}
```

Ilustração 20: Demonstração compilação e execução do Exemplo 11.

Anotações

Exemplo 12:

O usuário informa o número do mês e o programa responde com o nome deste mês. Se o valor não for de 1 a 12 é exibida uma mensagem de alerta.

```
#include <stdio.h>
#include <stdlib.h>
int main( )
{
    int mes;
    printf("Entre com o numero do mes: ");
    scanf("%d",&mes);
    switch(mes)
    {
      case 1: printf("Janeiro \n");    break;
      case 2: printf("Fevereiro \n");  break;
      case 3: printf("Marco \n");      break;
      case 4: printf("Abril \n");      break;
      case 5: printf("Maio \n");       break;
      case 6: printf("Junho \n");      break;
      case 7: printf("Julho \n");      break;
      case 8: printf("Agosto \n");     break;
      case 9: printf("Setembro \n");   break;
      case 10: printf("Outubro \n");   break;
      case 11: printf("Novembro \n");  break;
      case 12: printf("Dezembro \n");  break;
      default: printf("o numero deve ser de 1 a 12 \n");
    }
    system("PAUSE");
    return 0;
}
```

Observe na ilustração 21 a execução do programa para o mês igual a 8

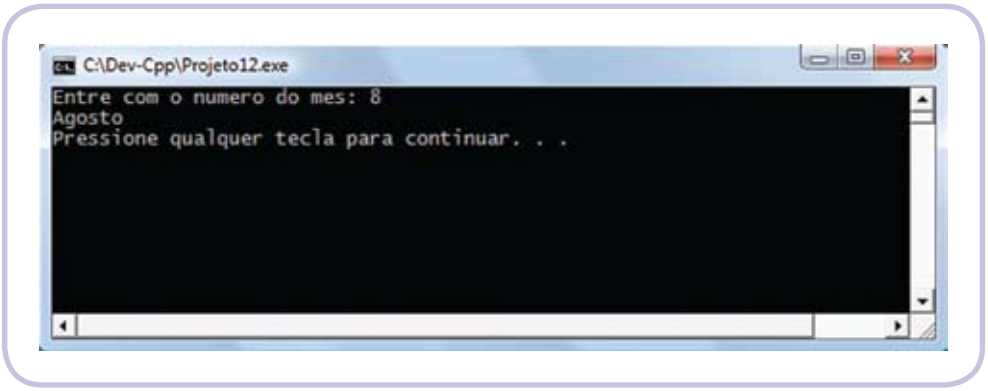


Ilustração 21: Demostração de uma execução do exemplo 12.

Anotações

Exemplo 13:

Neste exemplo o usuário informa o número do mês e o programa exibe a quantidade máxima de dias possíveis para o referido mês. Observe que os comandos 'case' podem ser agrupados para disparar uma única ação. Por exemplo, os meses abril, junho, setembro e novembro (números 4, 6, 9 e 11) possuem um máximo de 30 dias e assim fazem uma única atribuição de 30 para a variável maximoDias.

```
#include <stdio.h>
#include <stdlib.h>
int main( )
{
    int mes, maximoDias;
    printf("Entre com o numero do mes: ");
    scanf("%d",&mes);
    switch(mes)
    {
      case 1:
      case 3:
      case 5:
      case 7:
      case 8:
      case 10:
      case 12: maximoDias = 31; break;

      case 4:
      case 6:
      case 9:
      case 11: maximoDias = 30; break;

      case 2: maximoDias = 29; break;

      default: printf("o numero deve ser de 1 a 12 \n");
               system("PAUSE");
               return 0;
    }
    printf("Este mes tem no maximo %d dias \n", maximoDias);
    system("PAUSE");
    return 0;
}
```

Observe na ilustração 22 a execução do programa para o mês de junho (número 6).

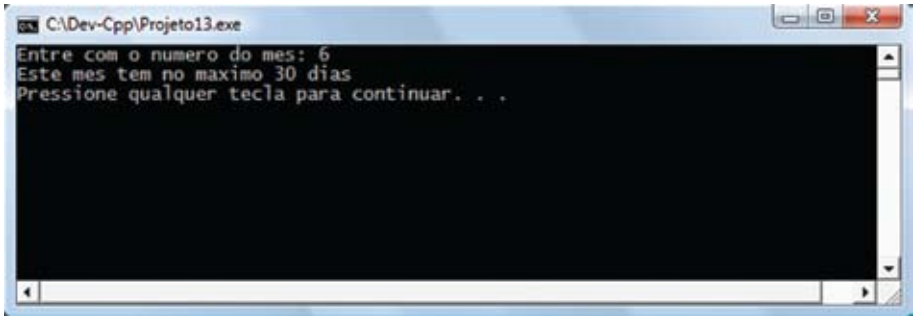


Ilustração 22: Demostração de uma execução do exemplo 13.

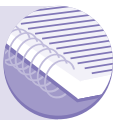
Anotações

Conceitos

- **if:** Estrutura condicional "if" assume uma as duas possíveis ações com base no resultado do teste. (SCHILDT, 1995, p. 89)
- **if-else:** Estrutura condicional "if" assume uma as duas possíveis ações com base no resultado do teste. (SCHILDT, 1995, p. 91)
- **swith:** Testa uma variável em relação a diversos valores preestabelecidos.

  (SCHILDT, 1995, p. 98)

Atividades



30. Faça um programa que leia três valores distintos a serem digitados pelo usuário e, utilizando o comando ***if-else*** , determine e exiba o menor deles.

31. Uma empresa dará aumento aos seus funcionários, de acordo com sua função:

a) função 1 (vendedor) = 0,10(10%) de aumento;

b) função 2 (gerente) = 0,15(15%) de aumento;

c) função 3 (diretor) = 0,20(20%) de aumento.

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

Atividades

32. Usando o ***comando if***, desenvolva um programa que peça ao usuário um valor e o armazene na variável X. Tal programa deverá efetuar o cálculo de C ← (A+B) * X se o valor informado for maior que 5, e efetuar o cálculo C ← (A-B)*X se o valor for menor que 5. Se o valor for igual a 5 não faça nada.

Fala Professor

O que aprendemos até aqui?

- *Que há três comandos de seleção, os quais nos ajudam a desenvolver programas com mais clareza.*
- *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 entre os variados blocos de comandos existentes.*
- *Que o comando **switch** é utilizado quado temos condições que não sejam expressões.*

2.9 Comandos de Repetição

Até agora, só conseguíamos executar o programa uma única vez. Se houvesse a necessidade de um novo teste, o que fazíamos era executar o programa novamente.

A partir de agora, conheceremos os comandos de repetição, os quais nos ajudarão em casos em que a repetição de parte do programa se fizer necessária.

Os comandos são: **for, while e do while**

2.9.1 O Comando For

Sintaxe: ***for (inicialização ; condição ; incremento) declaração;***

Usaremos o comando ***for*** quando conhecermos antecipadamente o número de vezes que uma determinada parte do programa (loop) se repetirá.

O comando ***for*** avalia primeiramente a expressão ***inicialização.*** Depois avalia a expressão ***condição*** que, se verdadeira, passa a executar o corpo de comandos (***declaração***). Em seguida avalia a expressão ***incremento*** e recomeça tudo novamente na avaliação da expressão ***condição***. Se a expressão condição não for verdadeira então o processamento vai para o próximo comando após o ***for.***

Exemplo 14:

Uma utilização do comando ***for*** bem simples: exibir no console a frase "Linguagem de Programação C" 5 vezes.

```
#include <stdio.h>
#include <stdlib.h>
int main( )
{
    int n;
    for (n = 1; n <= 5; n++)
        printf("Linguagem de Programacao C \n");

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

Vamos entender o que fazem as duas linhas:

```
for(n = 1; n <= 5; n++)
        printf("Linguagem de Programacao C \n");
```

A linha 1 inicialmente faz a variável n receber 1. Em segundo lugar avalia se n ainda é menor ou igual a 5. Em terceiro lugar, a linha 2 exibe a frase "Linguagem de Programação C". Em quarto lugar , o comando for

avalia n++ (isto é o mesmo que fazer n receber n+1, ou seja, é uma expressão para incrementar em uma unidade o valor de n). Nesta hora, com n valendo 2, volta para avaliar se n ainda é menor ou igual a 5 e continua.

Quando n receber o valor 6 então o processamento sai do comando for, pois a condição n<=5 não será mais verdadeira.

A ilustração 23 mostra a saída do exemplo 14.

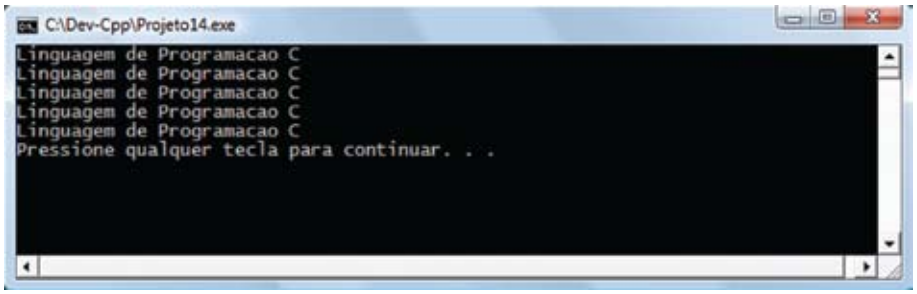


Ilustração 23: Demostração de uma execução do exemplo 14.

Exemplo 15:

Uma outra utilização simples: exibir os números pares de 10 a 500.

```
#include <stdio.h>
#include <stdlib.h>
int main( )
{
    int par;
    for (par = 10; par <= 500; par += 2)
        printf("%d ", par);

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

Observe que a expressão:

```
par += 2
equivale a:
par = par + 2
```

O comando ***for***, desta vez, determina o início da variável par em 10 e o término quando a variável par passar de 500.

A ilustração 24 mostra a saída do exemplo 15.

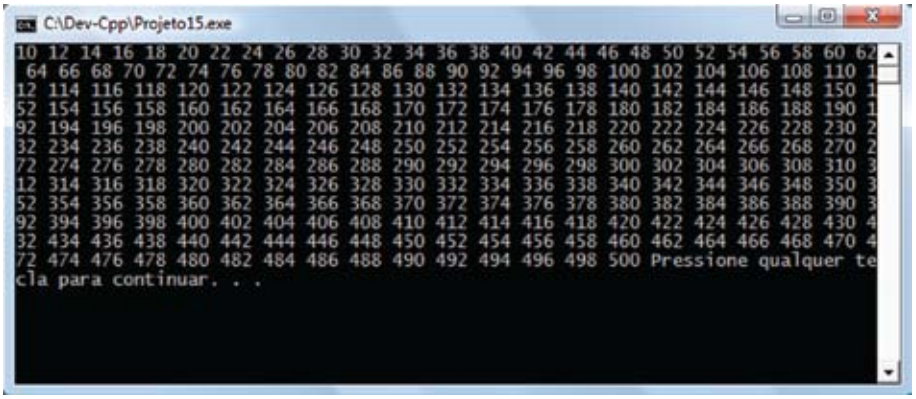


Ilustração 24: Demostração de uma execução do exemplo 15.

Exemplo 16: 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=0, conta;
linha 6      for(conta=0; conta<=9; conta++)
linha 7      {
linha 8         printf( "Digite a nota ");
linha 9         scanf("%f",&nota);
linha 10        soma=soma+nota;
linha 11     }//esta chave encerra o comando de repetição for.
Linha 12     media= soma/conta;
linha 13     printf( "A media da turma e %f \n",media);
linha 14     system("PAUSE");
linha 15     return 0;
linha 16  }
```

Vamos entender melhor algumas linhas do código acima.

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

Houve a necessidade de iniciarmos essas variáveis com zero, por tratar-se de variáveis que terão valores cumulativos.
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 os valores sejam acumulados corretamente.

linha 6... for(conta=0;conta<=;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

Observe que o comando **for** possui um par de chaves envolvendo os comandos Isto é necessário quando há mais de um comando para ser repetido. Compare com os exemplos 14 e 15 que não possuem estas chaves. Isto é o mesmo que acontecia com a estrutura ***if-else***.

linha 10... soma=soma+nota;
Nessa linha acumula-se a soma das notas da turma.

linha 12... 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.

linha 13... printf("A media da turma e %f \n ", media);
Para melhorarmos a exibição dessa mensagem, basta trocar %f por %.2f , será exibido apenas 2 casas depois da vírgula.

Vejamos na Ilustração 15 o resultado do nosso programa:

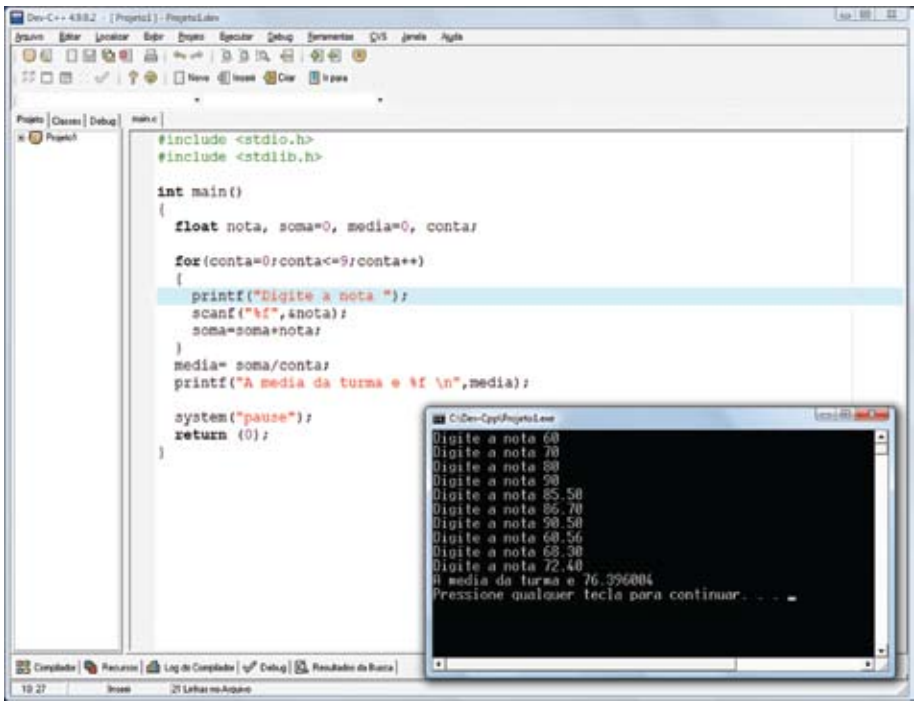


Ilustração 25: compilação e execução do código fonte do Exemplo 16

Fala Professor

Olá,

Para fazer as atividades seguintes, lembre-se de consultar todo material já disponível e ainda outros materiais, se julgar necessário. Tente sempre fazer suas atividades individualmente, pois isso, lhe dará autoconfiança e você crescerá em conhecimento e agilidade para resolver problemas.

Bom estudo !!!

Atividades

33. Desenvolva um programa para exibir os números ímpares de 3 a 49.

Atividades

34. Faça um programa que leia 5 valores reais e escreva o seu somatório.

35. Elabore um programa para exibir a tabuada de um número fornecido pelo usuário.

Por exemplo se o número informado for 5, então será exibido:

5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
5 x 6 = 30
5 x 7 = 35
5 x 8 = 40
5 x 9 = 45
5 x 10 = 50

Atividades

36. Apresente uma tabela de conversão de reais em dólares. Ela deve ser totalmente configurável, ou seja o usuário pode informar o valor inicial e final, o valor de incremento e o valor de 1 dólar. Apresente os números no formato monetário com duas casas decimais. Dica: use o formato "%.2f " na função printf para obter exatamente duas casas decimais.

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

2.9.2. O Comando While

sitaxe: **while** (condição) declaração;

O comando **while** repete o bloco de comandos (***declaração***) enquanto a condição for verdadeira.

Usaremos o comando **while** quando não soubermos antecipadamente o número de vezes que o programa será executado. Quem definirá o fim da execução do programa será o usuário.

Exemplo 17: Vamos utilizar o mesmo exemplo do comando ***for,*** porém o que faremos é ler a primeira nota as demais só serão lidas se o usuário desejar.

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=0;
linha 6    int resp=1, contador=0;
linha 7    while(resp==1)
linha 8    {  // esta chave inicia o comando 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. Perceba que ela já contém a primeira resposta. Isso se faz necessário para que a primeira vez possa ser executada.

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

linha 14... contador++
A variável contador está acumulando as entradas.

Vamos ver na Ilustração 16 o resultado da compilação e da execução do nosso programa:

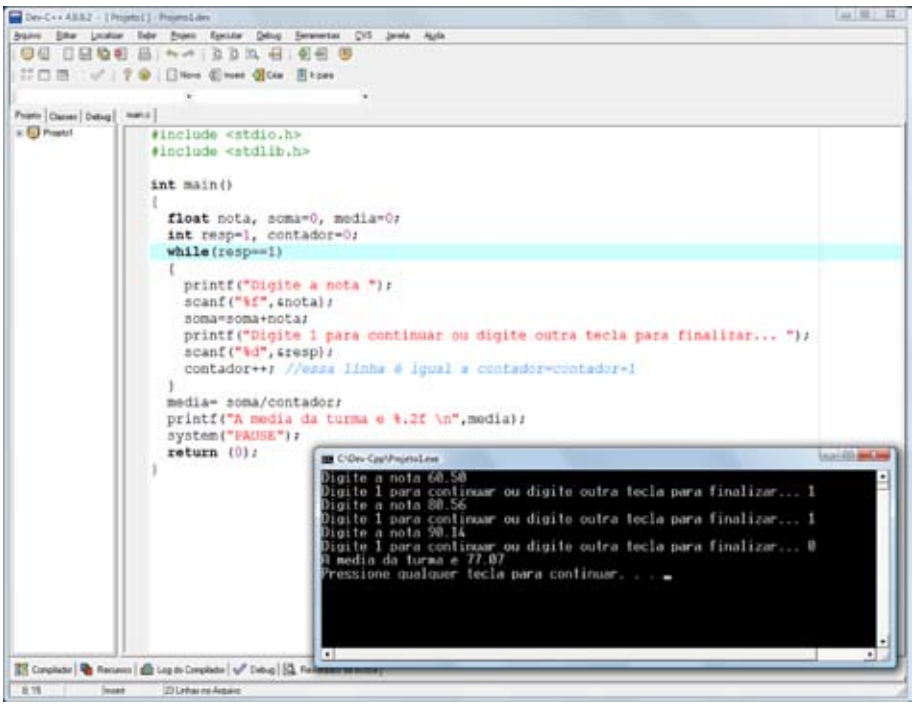


Ilustração 26: compilação e execução do código fonte do Exemplo 17

Exemplo 18:

Vamos refazer o exemplo 15 (exibe os números pares de 10 a 500), agora com o comando ***while.***

```c
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int par = 10;
    while(par <= 500) {
        printf("%d ", par);
        par += 2;
    }

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

Comparando com o exemplo 15, é como se houvesse um desmembramento das três expressões do comando ***for***. A primera fica logo no início pois é avaliada apenas uma vez. A segunda permanece no comando pois é a condição para manter a repetição, e a terceira fica no final do bloco de comandos pois é executada somente depois.

A saída no console do exemplo 18 é igual a do exemplo 15.

Exemplo 19:

O usuário irá fornecer uma relação de notas dos alunos de uma turma. Para terminar a entrada ele terá que digitar uma nota negativa. Em seguida será exibida a maior de todas as notas.

```c
#include <stdio.h>
#include <stdlib.h>
int main()
{
    float nota = 0, notaMaior = 0;
    while(nota >= 0) {
        printf("Entre com a nota: ");
        scanf("%f", &nota);
        if(nota > notaMaior)
           notaMaior = nota;
    }
    printf("\nA maior nota e: %.2f \n\n", notaMaior);

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

Observe a importância da variável notaMaior. É ela que armazena a maior nota, a cada nova nota que é digitada.

A ilustração 27 mostra a saída do exemplo 19.

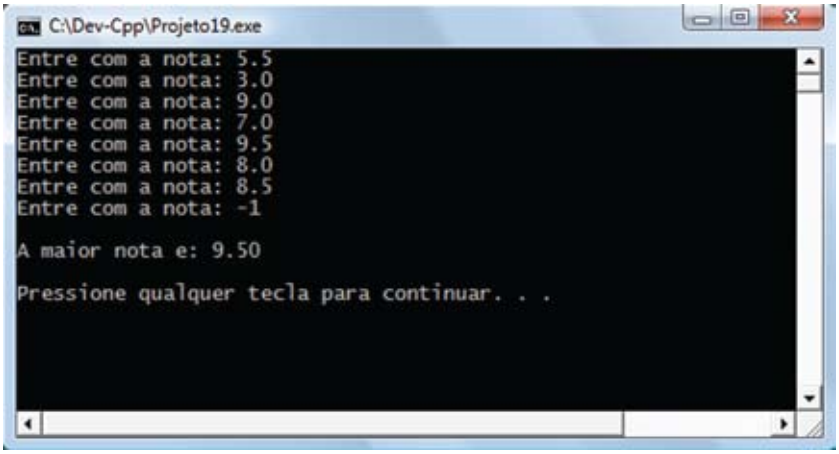


Ilustração 27: Demostração de uma execução do exemplo 19.

Fala Professor

Olá,

Lembre-se de que as atividades solicitadas neste fascículo, não são avaliativas. A finalidade delas é dar base ao aluno para que ele possa aproveitar melhor o tempo disponível quando estiver no ambiente virtual. Os exercícios preparam você para realizar com sucesso as atividades propostas na sala de aula virtual.

Por isso, aproveite, faça sempre todas as atividades propostas no seu material impresso.

Bom estudo !!!

Atividades

38. Usando o comando ***while*** faça um programa para exibir os números múltiplos de 5 a partir de 10. A quantidade de números será determinada pelo usuário.

39. Usando o comando ***while*** faça um programa para receber as idades de um grupo de pessoas. Não se sabe a quantidade de pessoas do grupo, permita que o usuário determine o término da entrada quando ele digitar uma idade negativa. Em seguida exiba a menor de todas as idades.

Atividades

40. Faça um programa que leia a data de validade (apenas o número do mês) e o preço das mercadorias de um supermercado. Considere que:
a) A mercadoria que vencer no mês 7 terá um desconto de 15%.
b) A mercadoria que vencer nos meses 8 e 9 terá um desconto de 10%.
c) As mercadorias com vencimento nos demais meses não terão desconto.
Exiba o preço da mercadoria já com o desconto.
Não se sabe a *priori* a quantidade de mercadorias, por isso o programa será executado enquanto o usuário quiser.

2.9.3. O Comando *do while*

Sintaxe: ***do***

```
{
   comandos;
} while(condição)
```

Com esse comando, asseguramos que o programa seja executado pelo menos uma vez. Depois de executar a primeira, o loop continuará a ser executado, enquanto a condição for verdadeira.

Exemplo 20: Vamos utilizar o mesmo exemplo do comando ***while.***

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=0;
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 algumas linhas do código acima.

linha 7... do
Início do comando do-while

linha 15... } while(resp= =1);
Enquanto essa condição for verdadeira, o programa continuara em execução.

Vamos ver na Ilustração 28 o resultado da compilação e da execução do nosso programa:

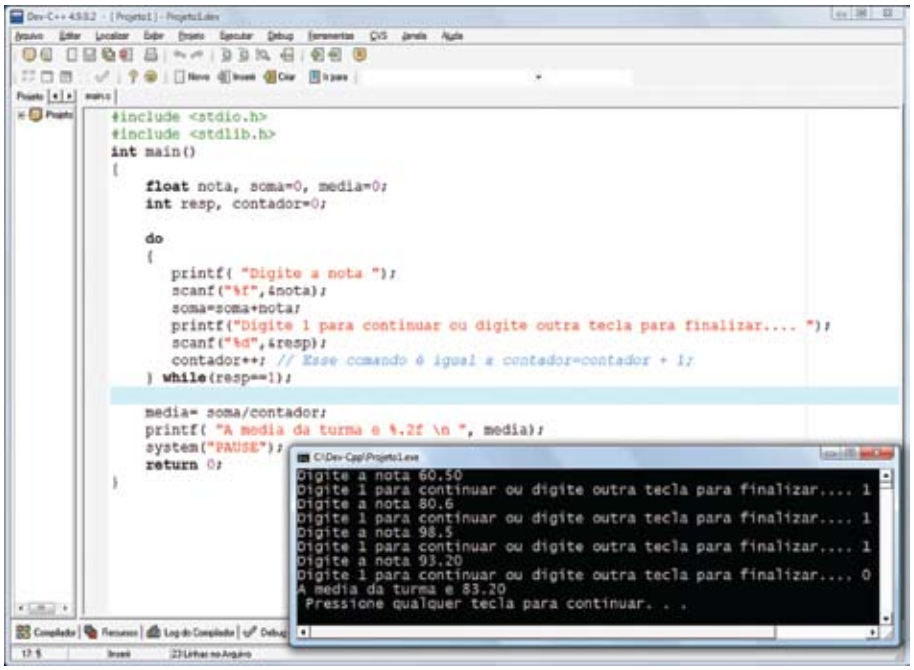


Ilustração 28: compilação e execução do código fonte do Exemplo 20

Anotações

Exemplo 21:

Vamos refazer o exemplo 15 (exibe os números pares de 10 a 500), agora com o comando ***do-while.***

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int par = 10;
    do
    {
        printf("%d ", par);
        par += 2;
    } while(par <= 500);

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

Comparando com o exemplo 18, observamos que a avaliação da condicional par<=500 vai ser feita por último. Neste caso a saída será a mesma dos exemplos 18 e 15, mas existem casos em que esta mudança traz diferenças bem visíveis pois a verificação só é realizada depois que o processamento já executou pelo menos uma vez o código. Veja mais detalhes nos três próximos exemplos.

Exemplos 22, 23: O objetivo é obter a letra inicial do nome do usuário e imprimí-la uma determinada quantidade de vezes.

O exemplo 22, a seguir, mostra uma versão feita com ***while***

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

int main( )
{
  char letraInicial;
  int qtdeVezes, contador;
  printf("Qual e a letra inicial do seu nome? ");
  scanf(" %c", &letraInicial);
  printf("Quantas vezes devo exibi-la? ");
  scanf("%d", &qtdeVezes);
  contador = 1;
  while(contador <= qtdeVezes) {
     printf("%c", letraInicial);
     contador++;
  }

  printf("\n\n"); // salta duas linhas em branco
  system("PAUSE");
  return 0;
}
```

Exemplo 22 – versão com while.

O exemplo 23 mostra uma versão feita com ***do-while***, mas, como o programador apenas substituiu o ***while*** pelo ***do-while*** sem fazer os ajustes necessários, ele não funciona para a quantidade igual a zero, pois a função printf é executada primeiramente para depois acontecer a avaliação da expressão.

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    // Não funciona para qtdeVezes igual a zero
    char letraInicial;
    int qtdeVezes, contador;
    printf("Qual e a letra inicial do seu nome? ");
    scanf(" %c", &letraInicial);
    printf("Quantas vezes devo exibi-la? ");
    scanf("%d", &qtdeVezes);
    contador = 1;
    do {
       printf("%c", letraInicial);
       contador++;
    }
    while(contador <= qtdeVezes);

    printf("\n\n"); // salta duas linhas em branco
    system("PAUSE");
    return 0;
}
```

Exemplo 23 – versão com do-while

Para resolver este problema pode-se usar um comando ***if-else*** para verificar se a quantidade é maior do que zero. Melhor mesm é usar o comando apropriado para cada situação, ou seja, neste caso, o comando ***while*** (como no exemplo 22) ou o comando ***for*** que ainda é mais adequado para situações onde já se tem a quantidade de repetições, como neste caso.

Atividades

41. Construa um programa capaz de ler uma série de números até obter um número inferior a 100.

42. Construa um programa capaz de ler uma série de números até que apareça um número entre 1 e 5.

43. Refaça o exercício 39, agora com o comando ***do-while***.

Fala Professor

O que aprendemos até aqui?

- *Os comandos **for, while e do while** são responsáveis pela repetição do programa.*
- *Os três comandos podem ser usados para resolver o mesmo problema, cabe ao programador decidir qual deles melhor responderá às necessidades para a solução do problema.*

Conceitos

- **for:** comando de controle de fluxo que executa repetidamente um conjunto de comandos um numero determinado de vezes. (LAUREANO, 2005, p. 39)
- **while:** comando de controle de fluxo que executa repetidamente um conjunto de comandos baseados em uma condição avaliada antes do comando ser executado. (LAUREANO, 2005, p. 40)
- **do while:** comando de controle de fluxo que executa repetidamente um conjunto de comandos baseados em uma condição avaliada depois do comando ser executado.(LAUREANO, 2005, p. 41)

Indicações

Leituras complementares:

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

ROBERT, Sebesta **Conceitos de Linguagem de Programação**, BookMan 2003.

Anotações

CAPÍTULO 3

VETORES E MATRIZES

Olá,

Neste Capítulo será abordado o conceito de vetor e matriz. Reserve um tempo maior para estudo e realização de atividades.

Vamos em frente!!!!

Fala Professor

3.1 Entendendo Vetor

Os comandos de repetição nos oportunizaram conhecer o comportamento de armazenagem de uma variável. Conseguimos constatar que cada vez que o loop se repetia, os dados eram "atualizados" nas variáveis e nesse processo, perdíamos os dados anteriores ao loop em execução.

Verificamos, então, a necessidade de conhecermos uma outra estrutura em que fosse possível armazenarmos os valores, sem que eles fossem constantemente "atualizados".

O vetor é uma estrutura em que poderemos armazenar vários dados, mas precisamos ficar atentos ao fato de que os dados armazenados deverão ser de um só tipo. Na declaração do vetor já definimos o tipo, ou seja, indicamos se os valores que serão armazenados nele, são do tipo **float** ou **inteiro**; por isso, conceitua-se o vetor como sendo uma estrutura de dados homogênea;

O tipo **char** será visto posteriormente.

•

3.1.1 Declarando Vetor

Sintaxe: ***tipo_da_variável nome_da_variável [tamanho];***

Vamos exemplificar o vetor para melhor entender a sua declaração.

A Ilustração 18 apresenta o vetor *NUMEROS* capaz de armazenar 5 valores inteiros:

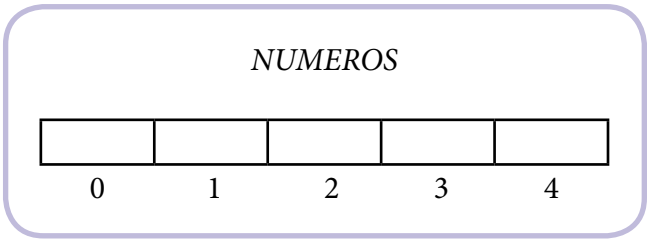


Ilustração 18: Demostração do Vetor NUMEROS

A declaração do nosso vetor ficará assim: **int** *NUMEROS*[*5*];

Na declaração, estamos informando que:

int - armazenará apenas valores do tipo inteiro;

NUMEROS[*5*] - poderá armazenar até 5 valores.

Atenção

O fato de termos declarado o vetor com 5 posições , Figura 18, não significa que estamos livre 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.

3.1.2 Atribuindo Valores ao Vetor (Inicialização)

Para atribuirmos um valor ao vetor, precisaremos indicar em qual posição o valor será armazenado. Para essa indicação, utilizaremos o índice do vetor.

Exemplo 24:

```
#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("%d foi armazenado no vetor. \n", NUMEROS[0]);
    printf("%d foi armazenado no vetor. \n", NUMEROS[1]);
    printf("%d foi armazenado no vetor. \n", NUMEROS[2]);
    printf("%d foi armazenado no vetor. \n", NUMEROS[3]);
    printf("%d foi armazenado no vetor. \n", NUMEROS[4]);
    system("PAUSE");
}
```

Vejamos, na Ilustração 18, como ficaria esse código, após execução:

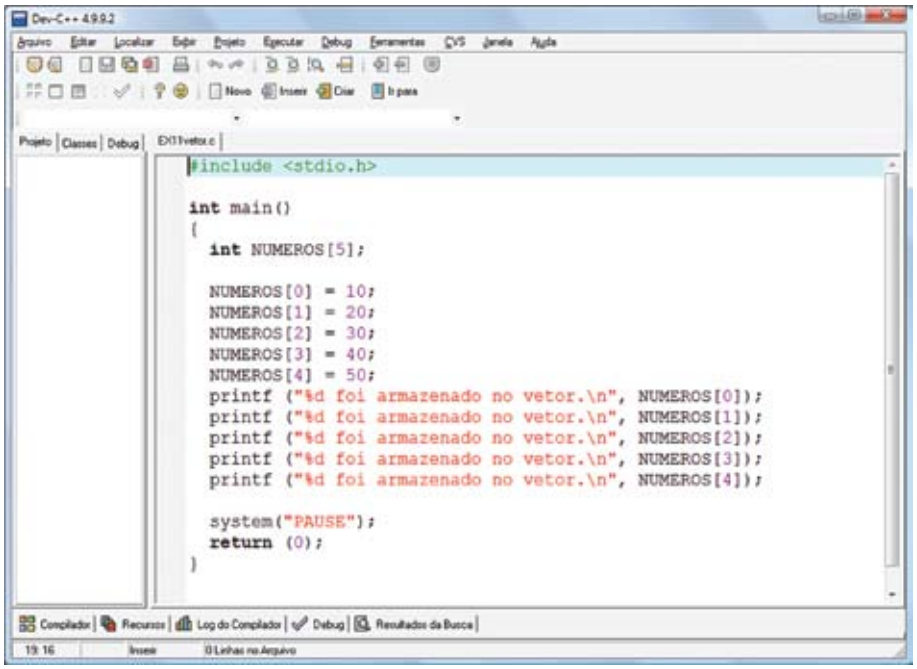


Ilustração 29: compilação e execução do código fonte do Exemplo 24

Se a atribuição que fizemos no Exemplo 24, fosse de 10 valores inteiros para um vetor com 10 posições, teríamos que escrever mais cinco atribuições. E se fosse um vetor com 500 posições? Imagine escrever essas atribuições uma a uma.

Vamos utilizar o comando de repetição **for**, para nos ajudar a fazer essas atribuições sem ter que escrevê-las.

Exemplo 25:

```
linha 1   #include <stdio.h>
linha 2   #include <stdlib.h>
linha 3   int main ()
linha 4   {
linha 5      int NUMEROS[10], conta;
linha 6      for(conta=0; conta<10; conta++)
linha 7      {
linha 8         NUMEROS[conta]=conta;
linha 9      }
linha 10     for(conta=0; conta<10; conta++)
linha 11     {
linha 12        printf ("%d foi armazenado no vetor.\n", NUMEROS[conta]);
```

```
linha 13     }
linha 14     system("pause");
linha 15 }
```

Vamos entender algumas linhas do código:

linha 5... int NUMEROS[5], conta;
O vetor NUMEROS foi declarado.
A variável conta foi declarada.

linha 6... for(conta=1;conta<10;conta++)
O comando for será utilizado para repetir a atribuição enquanto a variável conta for menor que 10.

linha 8... NUMERO[conta]=conta;
Dentro do bloco do comando for o valor da variável conta será atribuída ao vetor.

linha 10 ... for(conta=1;conta<10;conta++)
O comando for irá exibir um a um os valores do vetor NUMEROS[ ].

linha 12 ...printf ("%d foi armazenado no vetor.\n", NUMEROS[conta]);
Os valores que foram armazenados no vetor serão exibidos um a um.

Na Ilustração 19 podemos conferir como se comportou o nosso código:

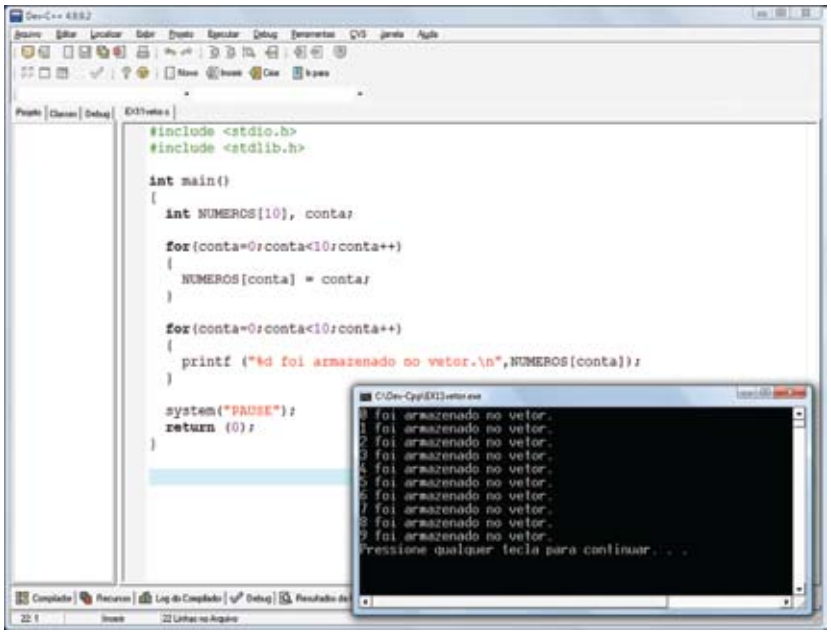


Ilustração 30: compilação e execução do código fonte do Exemplo 25

Anotações

Exemplo 26: Vamos observar as duas utilizações de comando for em nosso programa.

- o primeiro **for** irá preencher o vetor com valores inteiros digitados pelo usuário.
- o segundo **for** irá exibir os valores digitados.

O código ficara assim:

```c
#include <stdio.h>
#include <stdlib.h>
int main ()
{
    int NUMEROS[5],conta;
    printf("  PREENCHENDO O VETOR: \n\n");
    for(conta=0;conta<5;conta++)
    {
        printf("Digite o valor: ");
        scanf("%d", &NUMEROS[conta]);
    }
    printf("\n EXIBINDO VALORES DO VETOR: \n\n");
    for(conta=0;conta<5;conta++)
        printf("%d foi armazenado no vetor. \n", NUMEROS[conta]);

    system("PAUSE");
}
```

A Ilustração 31 mostrará a compilação e execução do programa.

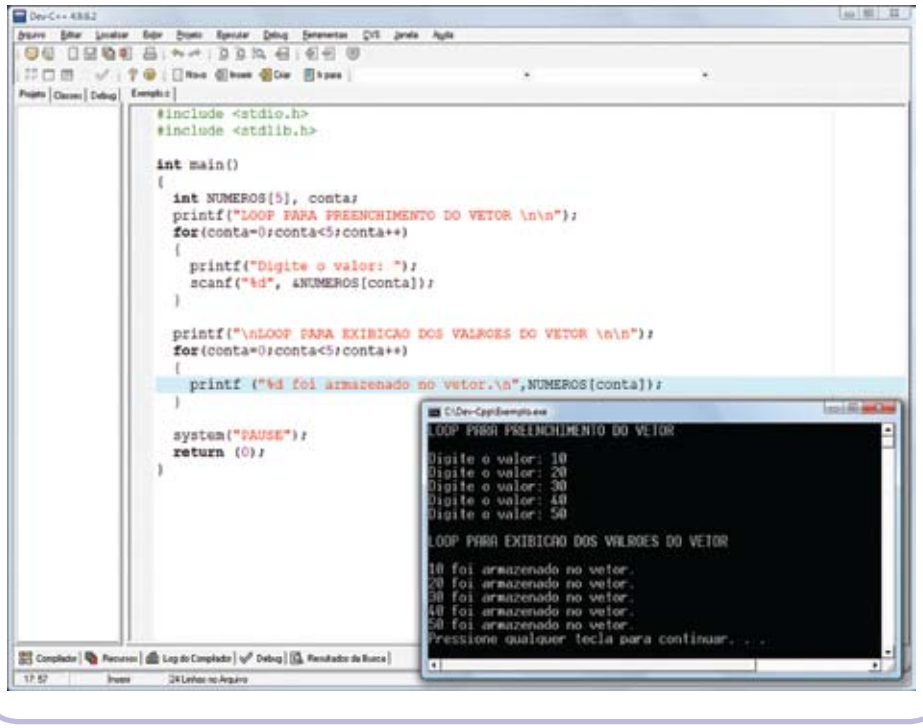


Ilustração 31: Compilação e execução do código fonte do Exemplo 26

Anotações

Fala Professor

Detalharemos mais as operações sobre vetores no ambiente da sala de aula virtual.

Vamos em frente!!!!

Atividades

44. Faça um programa que:

a) preencha dois vetores A e B de 5 posições, com números inteiros;

b) atribua a um vetor RES a soma do vetor A com B (a primeira posição de A será somada à primeira posição de B e o resultado será atribuído a primeira posição do Vetor RES);

c) mostre os valores do vetor RES. _____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

3.1.3. Vetor de String

Sintaxe: ***char nome_da_strig [tamanho];***

Na Linguagem C, ao declararmos uma string, estaremos, na realidade declarando um vetor que irá armazenar vários caracteres, terminando com um caractere nulo com valor inteiro igual a zero.

Vejamos na Ilustração 32 uma demostração de um vetor de 9 posições, para armazenar a string "Educação".

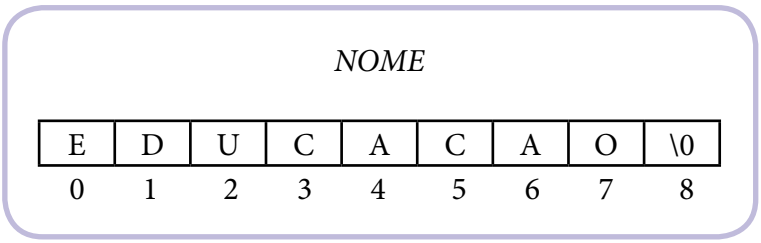


Ilustração 32: Demostração do vetor NOME

A declaração do nosso vetor ficará assim: **char** NOME[9];

Observe que declaramos uma posição a mais, garantindo uma posição para o caractere nulo.

3.1.4 Leitura de Vetor de String

Para armazenar uma string num vetor, a leitura será feita por meio da função **gets**( ). Essa função colocará o terminador nulo na string, assim que a tecla enter for pressionada.

Vejamos um exemplo de leitura de vetor do tipo string.

Exemplo 27:

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

Vejamos na Ilustração 33 como ficará nosso programa, após ser compilado e executado:

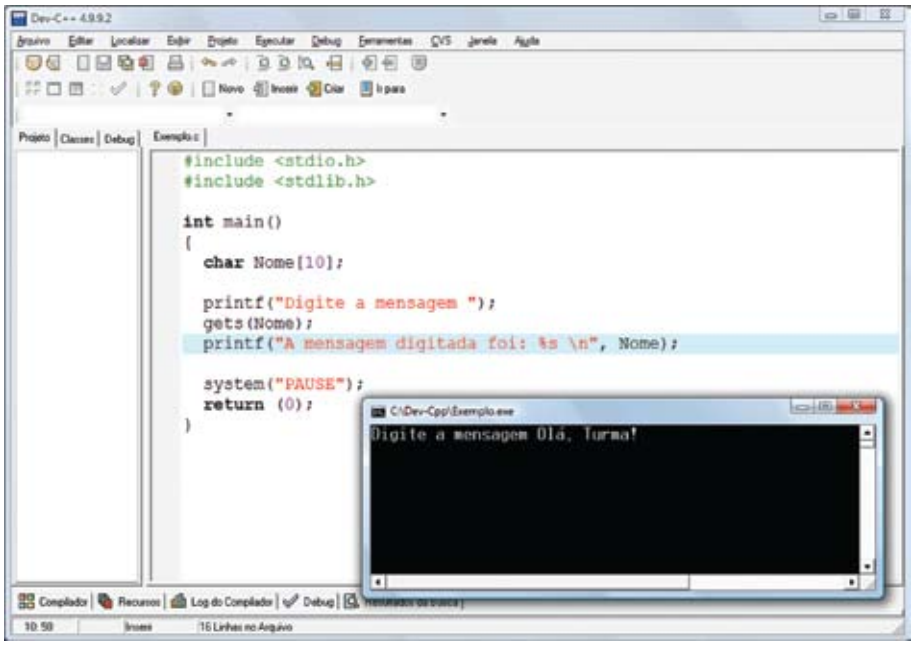


Ilustração 33: Compilação e execução do código fonte do Exemplo 27.

Vamos entender algumas linhas do código:

linha 5... char Nome[10];
Declaramos um vetor com 10 posições. Isso significa que esse vetor poderá armazenar até 9 caracteres.

linha 7... gets(Nome);
Observe que para leitura não utilizamos a função scanf( ); a função utilizada para leitura foi a gets( ).

linha 8... printf("A mensagem digitada foi: %s \n",Nome);
O que nos chama a atenção nessa linha é o caractere de impressão %s. Esse caractere, como já vimos, imprimi uma cadeia de caracteres.

O que aprendemos até aqui?

Fala Professor

- *Vetor é uma estrutura indexada que armazenará dados de um mesmo tipo(homogêneos).*
- *string na realidade, é um vetor de caractere, que deverá ser lida por meio da função gets( ).*

3.2 Matriz

Sintaxe: ***tipo_da_variável nome_da_variável [altura][largura];***

Exemplificamos, abaixo, uma matriz para entendermos a sua declaração.

A Ilustração 34 irá representar a matriz DADOS, capaz de armazenar as duas notas de 3 alunos.

DADOS

cada linha irá conter as duas notas, logo:

aluno	nota1	nota2
0	80,50	75,90
1	70,50	75,30
2	40,30	60,21

0	80,50	75,90
1	70,50	75,30
2	40,30	60,21
	0	1

cada coluna representa uma nota, logo:
0 = primeira nota
1 = segunda nota

Ilustração 34: Demonstração Matriz

A declaração da nossa matriz ficará assim: float DADOS[3][2];

Na declaração informamos que:

- **float** - armazenará apenas valores do tipo float;
- DADOS [3][2] - o índice da esquerda [3] – indexa as linhas;
  o índice da direita[2] – indexa as colunas;

Atenção

Vale a pena lembrar :

- Na linguagem C, os índices iniciam de zero;
- Nós é que mantemos os índices na faixa declarada, a Linguagem C não fará o controle automaticamente.

Façamos o código do programa da matriz, representada na Ilustração 35:

Exemplo 28:

```
linha 1  #include <stdio.h>
linha 2  #include <stdlib.h>
linha 3  int main ()
```

```
linha 4   {
linha 5     float dados[3][2];
linha 6     int lin,col;
linha 7     printf("INICIANDO O LOOP DE LEITURA \n \n");
linha 8     for(lin=0;lin<3;lin++)
linha 9        for(col=0;col<2;col++)
linha 10       {
linha 11          printf("Digite a nota: ");
linha 12          scanf("%f", &dados[lin][col]);
linha 13       }
linha 14    printf("\nINICIANDO O LOOP DE EXIBICAO \n\n");
linha 15    for(lin=0;lin<3;lin++)
linha 16       for(col=0;col<2;col++)
linha 17          printf("nota = %.2f  \n",dados[lin][col]);
linha 18    system("pause");
linha 19 }
```

A execução do código poderá ser analisada, por meio da Ilustração 24, abaixo:

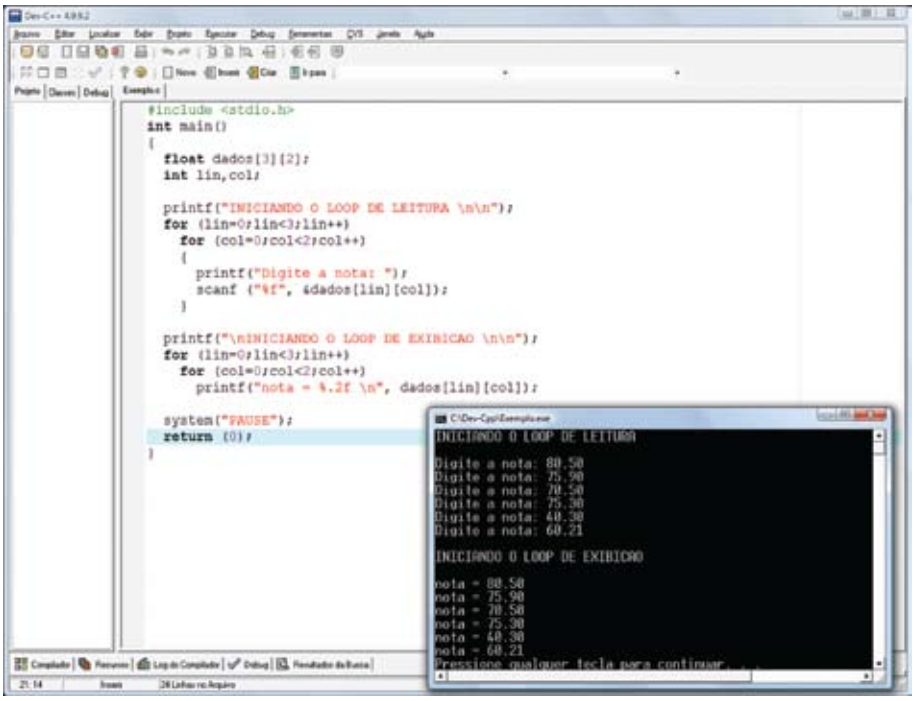


Ilustração 35: Compilação e execução do código fonte do Exemplo 28

Vamos entender algumas linhas do código:

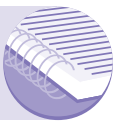
```
linha 8     for(lin=0;lin<3;lin++)

linha 9         for(col=0;col<2;col++)
```

O primeiro ***for*** controlará a linha, e o segundo controlará a coluna.

Anotações

Atividades



45.Carregue uma matriz com as 3 notas de 4 alunos. Calcule a média das notas e mostre:

a) quantas notas ficaram abaixo da média;

Atividades

46. Carregue uma matriz 3 x 3 com os valores das vendas de uma loja supondo 3 meses e 3 vendedores. Calcule e mostre: em cada mês qual foi o vendedor que vendeu mais.

3.2.1 Matriz de String

Pelo que aprendemos até aqui, podemos deduzir que uma matriz de string armazena vários nomes. Precisamos ficar atentos ao fato de que uma é a forma de se declarar a matriz e outra é a forma de se fazer referências a ela.

Ao declararmos a matriz, precisamos informar os dois índices (linha e coluna), porém, a referência será feita apenas por um índice.

Exemplo 29: Façamos uma matriz para armazenar o nome de três pessoas, sendo que cada nome poderá ter, no máximo, 30 caracteres.

```
linha 1  #include <stdio.h>
linha 2  #include <stdlib.h>
linha 3  int main ()
linha 4  {
linha 5     char nomes[3][30];
linha 6     int lin;
linha 7     printf("INICIANDO O LOOP DE LEITURA\n\n");
linha 8     for(lin=0;lin<3;lin++)
linha 9     {
linha 10       printf("Digite o nome: ");
linha 11       gets(nomes[lin]);
linha 12    }
linha 13    printf("\nINICIANDO O LOOP DE EXIBICAO\n\n");
linha 14    for(lin=0;lin<3;lin++)
linha 15        printf("nome = %s  \n",nomes[lin]);
linha 16    system("pause");
linha 17 }
```

Vamos entender algumas linhas do código:

linha 5... char nomes[3][30];
Declaramos a matriz para receber 3 nomes de, no máximo, 30 caracteres.

linha 6...int lin;
Declaramos a variável lin.

linha 7...printf("INICIANDO O LOOP DE LEITURA \n \n");
O printf imprimirá uma mensagem avisando o início da leitura.

linha 8... for(lin=0;lin<3;lin++)
Como a matriz é de string, só precisamos de um for para controlar o índice linha.

linha 11... gets(nomes[lin]);
Utilizamos a função gets( ) para ler os nomes (os caracteres).

linha 15... for(lin=0;lin<3;lin++)
Exibindo os nomes digitados.

Acompanhe a Ilustração 26: digitação, compilação e execução do Exemplo 29:

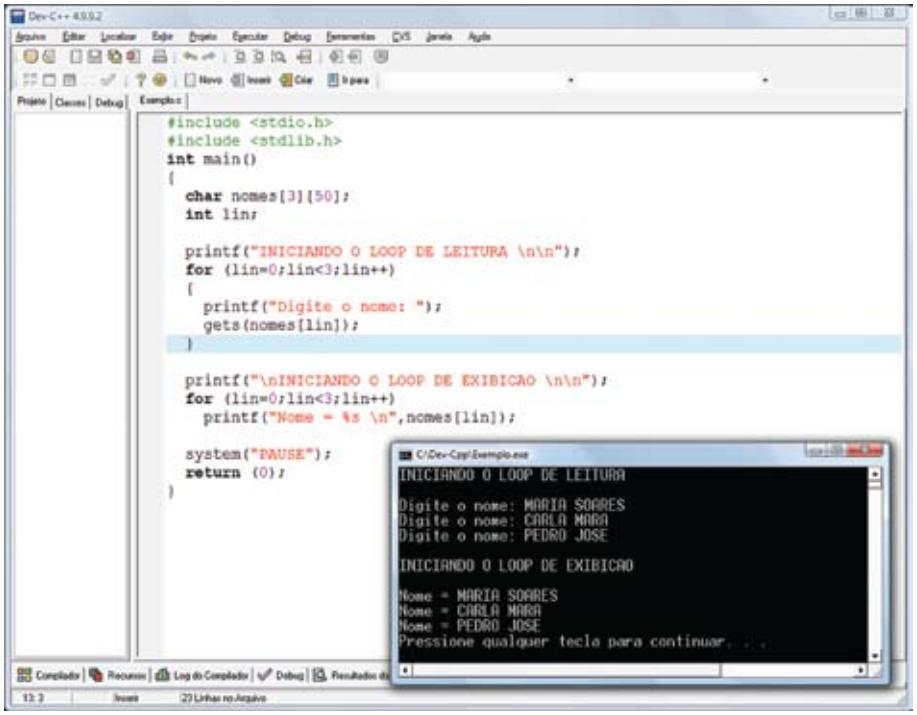


Ilustração 36: Compilação e execução do código fonte do Exemplo 29

Exemplo 30:

Neste programa vamos permitir o cadastro de vários nomes com as suas respectivas idades. Ao término da entrada será calculada a média das idades e em seguida será exibida a relação dos nomes cuja idade é maior do que a média.

```c
#include <stdio.h>
#include <stdlib.h>
void SOMA(int *x, int *y, int z);

int main ()
{
    int num1, num2, num3;
    num1 = 10;
    num2 = 20;
    num3 = 30;
    SOMA(&num1,&num2,num3); // colocar o & da var que será passada por referência
    printf("EXIBICAO DO VALOR DE num1 num2 num3:zn");
    printf(" num1 = %d \n", num1);
    printf(" num2 = %d \n", num2);
    printf(" num3 = %d \n\n", num3);
    system("PAUSE");
    return 0;
}

void SOMA(int *x, int *y, int z)
{   //x e y terao seus valores alterados no progr. principal
     int resultado;
     resultado = *x + *y + z;
     *x = 0;
     *y = 1;
     z = 2;
     printf("num1+num2+num3= %d \n\n", resultado);
}
```

Algumas observações:

- Foi declararada uma matriz de char para que houvesse um limite de 50 nomes com no máximo29 caracteres, pois o último já é reservado ao \0.
- O vetor das idades armazena de forma correspondente as posições dos nomes, ou seja, a quinta idade (idades[4]) se refere ao quinto nome (nomes[4]).
- O acesso ao nome é feito usando-se apenas a primeira dimensão da matriz nome[n].

A ilustração 37 mostra uma possível execução do exemplo 30.

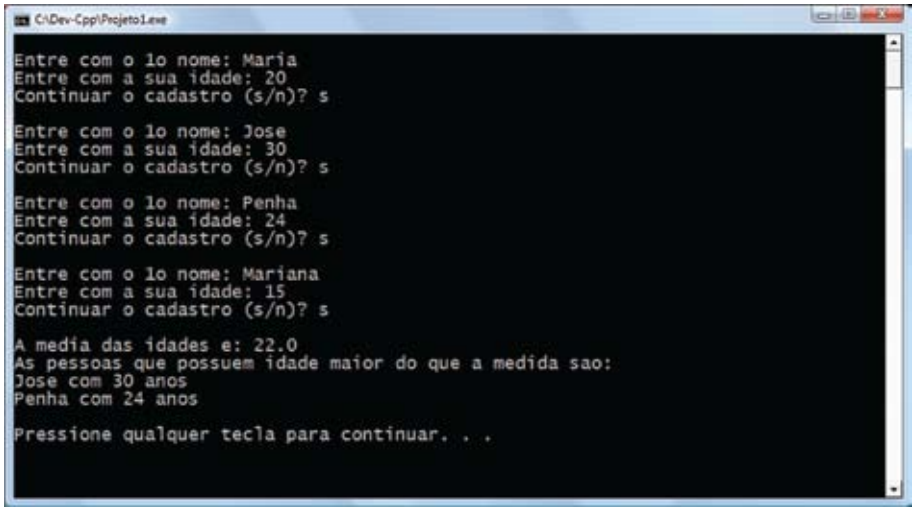


Ilustração 37: Demostração de uma execução do exemplo 30.

Anotações

Atividades

47. Elabore um programa para receber os alunos de uma turma com as suas respectivas notas finais. Após a entrada exiba os nomes cuja notas sejam maiores do que a média da turma e que ao mesmo tempo sejam maiores do que a média de aprovação que é igual a 7.0

Atividades

Fala Professor

O que aprendemos até aqui?

• Na matriz podemos armazenar vários dados, desde que eles sejam do mesmo tipo.

• Uma matriz de string armazenará, por exemplo, vários nomes.

• Para fazer referência a uma matriz de string basta indicar apenas um índice.

Conceitos

•vetor: Matrizes unidimensionais. (LAUREANO, 2005, p.69).

• matriz: Agregado homogêneo de elementos de dados cujo elemento individual é identificado por sua posição no agregado em relação ao primeiro. (LAUREANO, 2005, p.70).

Indicações

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

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

CAPÍTULO 4

ESTRUTURAS(STRUCT)

Olá,

Estamos chegando na reta final, é importante continuarmos com o mesmo interesse e com a mesma dedicação.

Este capítulo exigirá de você muita dedicação e cumprimento de todas as tarefas.

Vamos em frente!!!!

Fala Professor

Vimos que na Matriz podemos armazenar dados de um mesmo tipo. Na Estrutura estes dados podem ser de vários tipos.

4.1 Declarando uma Estrutura

Sintaxe: ***struct nome_do_tipo_da_estrutura***

```
{
   tipo_1  nome_1;
   tipo_2  nome_2;
   ...
   tipo_n nome_n;
} variáveis_estrutura;
```

- Vamos declarar uma estrutura chamada pauta, que armazenará a matrícula e a média de um aluno:

```
struct pauta
 {
    int matr;
    float media;
 } aluno;
```

Temos que: pauta é uma estrutura com dois campos (matr e media).

aluno é uma variável do tipo pauta.

- Abaixo foi declarada uma estrutura endereço, que guardará os dados referentes ao endereço de uma única pessoa:

```
struct tipo_endereco
{
   char rua[50];
   int numero;
   char bairro[20];
   char cidade[30];
   char sigla_estado[3];
   int CEP;
}end;
```

- Criemos, agora uma estrutura chamada ficha_pessoal, com os dados pessoais de uma pessoa:

```
struct ficha_pessoal
{
  char nome[50];
  int telefone;
  struct tipo_endereco end;
}agenda;
```

Vemos, pelo exemplo acima, que uma estrutura pode fazer parte de outra (a struct tipo_endereco é usada pela struct ficha_pessoal).

Fala Professor

Olá,

Vamos fazer algumas declarações de estruturas para testar nosso aprendizado até aqui. As declarações solicitadas nessa atividade, devem ser realizadas individualmente.

Consulte a declaração da estrutura pauta, acima. Ela poderá auxiliá-lo.

Vamos em frente!!!!

Atividades

48. Declare as estruturas, conforme solicitado em cada um dos enunciados:

a) Para armazenar a média final e a matrícula de um aluno: _____

b) Para armazenar a matrícula, os número de dependentes e o salário de um funcionário: _____

c) Para armazenar o número do telefone, o número do celular, o valor da compra e o código de um cliente: _____

4.2 Utilizando Estrutura no Programa

Neste exemplo será declarada uma estrutura para armazenar e exibir a matrícula e a média de um aluno:

Exemplo 31:

```
linha 1   #include <stdio.h>
linha 2   #include <stdlib.h>
linha 3   int main() {
linha 4       struct pauta
linha 5       {
linha 6          int matr;
linha 7          float media;
linha 8       }aluno;
linha 9       printf("...SOLICITANDO OS DADOS...\n \n");
linha 10      printf("Digite a matricula: ");
linha 11      scanf("%d", &aluno.matr);
linha 12      printf("Digita a media: ");
linha 13      scanf("%f", &aluno.media);
linha 14      printf(" \n...MOSTRANDO OS DADOS...\n \n");
linha 15      printf("Matricula = %d\n",aluno.matr);
linha 16      printf("Media = %.2f \n", aluno.media);
linha 17      system("pause");
linha 18      return (0); }
```

Vejamos na Ilustração 39, como ficou o programa.

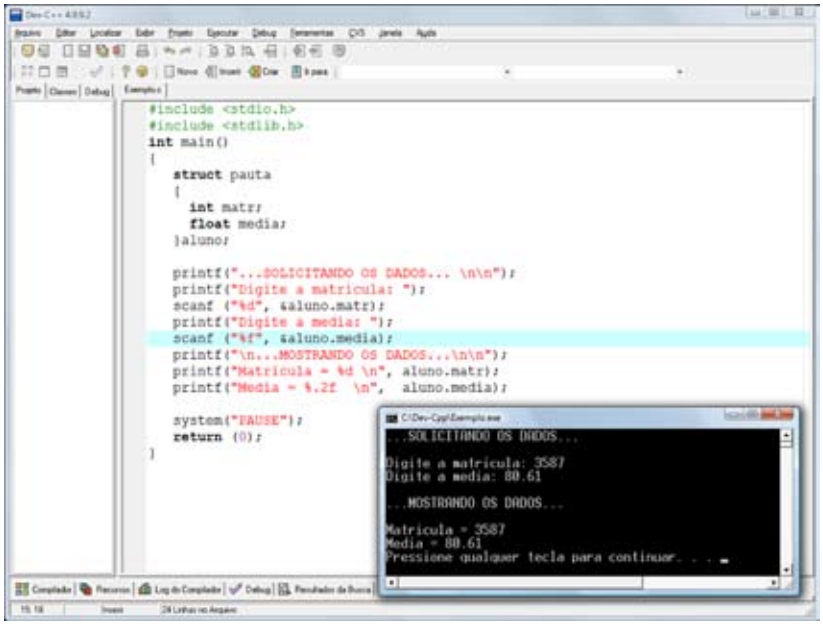


Ilustração 39: Demostração de uma execução do exemplo 31.

Vamos analisar algumas linhas do programa:

linha 11... scanf("%d", &aluno.matr);
Para a leitura correta do dado, precisamos informar em qual campo da variável aluno o dado lido será armazenado: Na linha 12, informamos que o dado será armazenado no campo matr da variável aluno (aluno.matr).

linha 16... printf("Media = %.2f \n", aluno.media);
Da mesma forma precisamos informar qual o campo será impresso. Na linha 17, informamos que o dado a ser impresso é a média (aluno.media).

Atividades

49. Aproveitando as declarações feitas na atividade 41, vamos agora desenvolver o programa:

a) Faça um programa que leia e exiba a média final e na matrícula de um aluno: _____

b) Faça um programa que leia e exiba a matrícula, os números de dependentes e o salário de um funcionário:

Atividades

c) Faça um programa que leia e exiba o número do telefone, o número do celular, o valor da compra e o código de um cliente

Fala Professor

Olá,

Até aqui fizemos várias atividades de estruturas, armazenando dados de uma única pessoa. Vamos, agora, fazer estruturas que armazenem os dados de várias pessoas e alterar os exercícios propostos na atividade 42 !!

Vamos em frente!!!!

4.3 Vetores de Estrutura (Struct)

Um vetor de estrutura é um vetor que armazenará os dados de várias pessoas. A estrutura pauta, que declaramos no início do capítulo, e que armazenava os dados de um único aluno, aqui irá armazenar os dados de uma turma inteira. Para isso temos que fazer uma pequena alteração, assim:

```
struct pauta
{
   int matr;
   float media;
} aluno[4];
```

Pronto, declaramos que a variável aluno tem 4 posições, verifique a representação da estrutura na Ilustração 40:

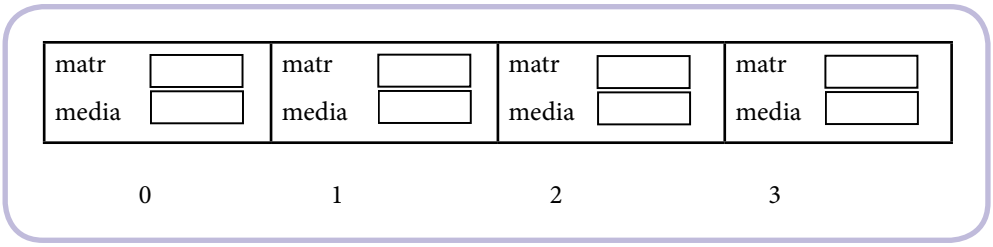


Ilustração 40: Representação da estrutura pauta

Vamos desenvolver um programa utilizando essa declaração:

Exemplo 32:

```
linha 1   #include <stdio.h>
linha 2   #include <stdlib.h>
linha 3   int main()
linha 4   {
linha 5      int lin;
linha 6      struct pauta
linha 7      {
linha 8         int matr;
linha 9         float media;
linha 10     }aluno[4];
linha 11     printf("...LOOP - SOLICITANDO OS DADOS...\n \n");
linha 12     for(lin=0;lin<4;lin++)
linha 13     {
linha 14       printf("Digite a matricula: ");
linha 15       scanf("%d", &aluno[lin].matr);
linha 16       printf("Digite a media: ");
linha 17       scanf("%f", &aluno[lin].media);
linha 18     }
linha 19     printf(" \n...LOOP - MOSTRANDO OS DADOS...\n \n");
linha 20     for(lin=0;lin<4;lin++)
linha 21     {
linha 22        printf("Matricula = %d\n",aluno[lin].matr);
linha 23        printf("Media = %.2f \n", aluno[lin].media);
linha 24     }
linha 25     system("pause");
linha 26     return (0);
linha 27  }
```

Vejamos, na Ilustração 41, como ficou o programa:

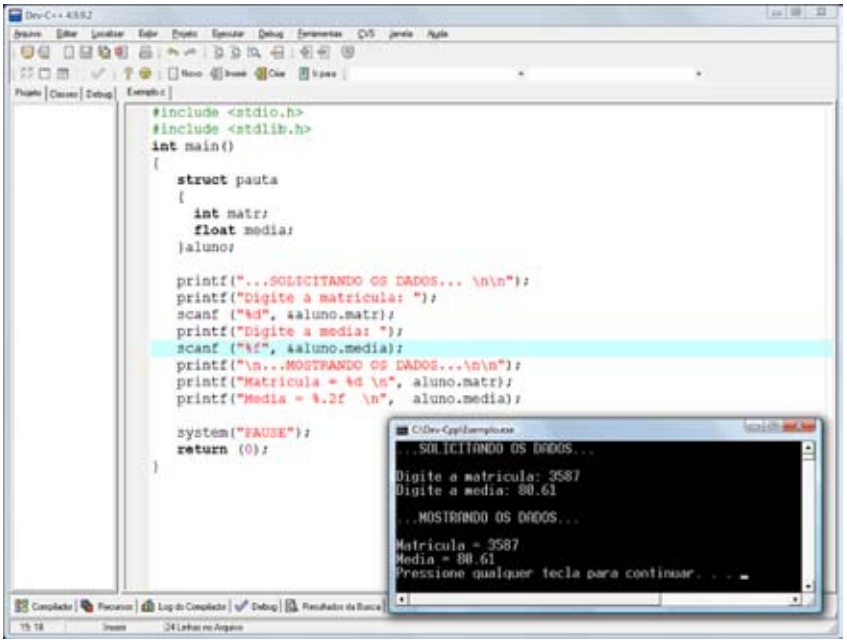


Ilustração 41: Compilação e execução do código fonte do Exemplo 32

Vamos analisar algumas linhas do programa:

linha 10... }aluno[4];
Declaramos um vetor do tipo struct com 4 posições.

linha 12... for(lin=0;lin<4;lin++)
Comose trata de um vetor, precisamos do comando for, para controlar as posições de armazenamento e exibição.

linha 15... scanf("%d", &aluno[lin].matr);
Informamos que a matricula que está sendo lida naquele momento, ou seja, na posição indicada por lin, será armazenada nessa posição.

linha 17... scanf("%f", &aluno[lin].media);
Da mesma forma, informamos a impressão.

Atividades

50. Aproveitando os mesmos exercícios realizados na atividades 42, vamos agora desenvolver o programa para armazenar os dados de um número maior de pessoas:

a) Faça um programa que leia e exiba o número do telefone, o número do celular, o valor da compra e o código de cinco clientes.

b) Faça um programa que leia e exiba a matrícula, os números de dependentes e o salário de cinco funcionários.

Atividades

c) Faça um programa que leia e exiba a media final e a matrícula de quatro alunos

Conceitos

struct: Agrupa um conjunto de dados não-similares, formando um novo tipo de dado. (LAUREANO, 2005, p.120)

Indicações

Leituras complementares:

LAUREANO, Marcos **Programando em C**. Rio de Janeiro 2005.

SCHILDT, Herbert. **C Completo e Total.** São Paulo: Makron Books , 1996.

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

Anotações

Fala Professor

O que aprendemos até aqui?

- *Os dados armazenados podem ser de vários tipos.*
- *Um vetor de estrutura é um vetor que armazenará os dados de várias pessoas.*

CAPÍTULO 5

PROCEDIMENTOS E FUNÇÕES

Olá,

O Capitulo 5 exigirá mais leitura do que os outros, mas os conceitos são necessários para o entendimento e para a aprendizagem do conteúdo.

Esta é a reta final.

Vamos em frente!!!!

Fala Professor

5.1 Modularização

É uma técnica de programação que utilizaremos para dividir um programa maior em programas menores. Os programas menores serão feitos individualmente, até que todo o programa tenha sido desenvolvido.

Para entendermos melhor, vamos imaginar um programa para calcular a folha de pagamento de uma empresa qualquer.

Podemos dividir esse programa em vários programas menores. Vamos identificar quais poderiam ser os programas menores da folha de pagamento:

- CALCULAR INSS
- CALCULAR DESCONTO DE FGTS
- CALCULAR IRRF
- CALCULAR SALÁRIO FAMÍLIA
- CALCULAR FÉRIAS
- CALCULAR SALÁRIO LÍQUIDO
- IMPRIMIR CONTRA CHEQUE.

Observe que agora o programa folha de pagamento será composto de 7 programas menores.

Utilizando a técnica de modularização, deixaremos o código mais claro e fácil de ser testado, já que cada programa menor poderá ser desenvol-

vido e testado individualmente, antes de fazer parte do programa maior (folha de pagamento).

Aos programas menores denominamos funções ou procedimentos.

Conceitos

modularização: programação modular, técnica de modularização, denota a construção de programas pela composição de partes pequenas para formar partes maiores. As partes são chamadas módulos. (SEBESTA, 2003, p. 123).

5.2 Funções

Sintaxe: ***tipo_de_retorno nome_da_função (declaração_de_parâmetros)***

{

corpo_da_função

}

O que difere um procedimento de uma função é o retorno da execução do bloco. Uma função irá retornar um valor para o local onde foi chamada.

5.2.1 Entendendo Funções

Vamos trabalhar com o exemplo da folha de pagamento para entendermos melhor como funciona esse bloco de programa.

Exemplo 33: Nesse exemplo vamos fazer o programa apenas do cálculo do INSS.

```
linha 1  #include <stdio.h>
linha 2  #include <stdlib.h>
linha 3  /*DECLARANDO A FUNCAO*/
linha 4  float CALCULAR_INSS (float salario)
linha 5  {    float sal_liq;
linha 6       sal_liq= salario - (salario*0.08);
linha 7       return(sal_liq); }
linha 9  //INICIANDO O PROGRAMA PRINCIPAL
linha 10 int main(){
linha 11     float salario_bruto, salario_novo;
linha 12     printf ("Digite o salario do fucionario:");
linha 13     scanf ("%f",&salario_bruto);
linha 14     salario_novo = CALCULAR_INSS(salario_bruto);
```

```
linha 15      printf("Salario liquido = %f\n\n",salario_novo);
linha 16      system("PAUSE");
linha 17      return 0; }
```

Vejamos na Ilustração 42, como ficou o programa:

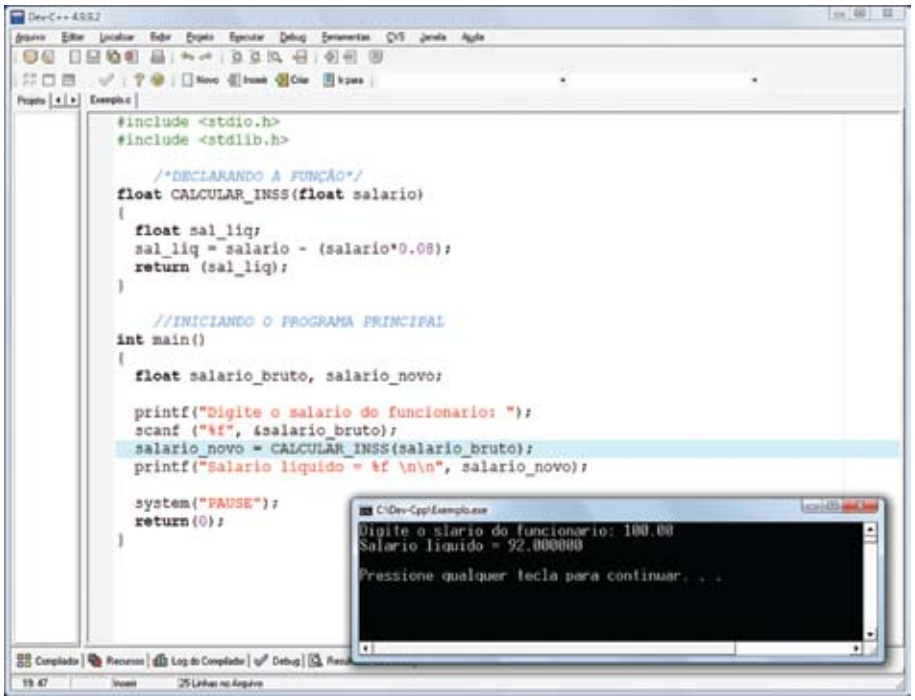


Ilustração 42: Compilação e execução do código fonte do Exemplo 33

Vamos entender algumas linhas:

linha 3... /*DECLARANDO A FUNCAO*/
Comentário apenas indicando que estamos iniciando a função.

Linha 4.. float CALCULAR_INSS (float salario)

- A função retornará um valor do tipo float.
- A função deverá ser chamada por: CALCULAR_INSS ( ).

linha 5..{float sal_liq;
Declaramos uma variável chamada sal_liq que utilizaremos dentro da função.

linha 6.. sal_liq= salario - (salario*0.08);
Efetuando o cálculo do desconto.

linha 7.. return(sal_liq);
O comando return( ), retornará o valor da variável sal_liq, para o programa principal, na linha em que chamamos a função.

linha 9..//INICIANDO O PROGRAMA PRINCIPAL
Comentário apenas indicando que estamos iniciando a função main( ), ou seja, o programa principal.

linha 11.. float salario_bruto, salario_novo;
Declarando as variáveis que serão utilizadas no programa principal.

linha 14.. salario_novo = CALCULAR_INSS(salario_bruto);

- Estamos atribuindo à variável salario_novo, o valor que será calculado pela função CALCULAR_INSS();
- A chamada à função CALCULAR_ INSS( ) é feita justamente nesse momento, ou seja, o seu nome é a chamada dela ao programa..

5.3 Procedimentos

Como a função, o procedimento é um bloco de programas, porém ele não retornará nenhum valor.

Na Linguagem C, toda função que retorna void (vazio) é um procedimento.

5.3.1 Entendendo Procedimentos

Para melhor compreensão, continuaremos utilizando o programa folha de pagamento. Identificaremos quais dos programas menores que o compõem, não retornarão nenhum valor.

Observe: o programa IMPRIMIR CONTRA CHEQUE terá void como retorno, ou seja, não retornará nenhum valor para o programa principal, ou para outra função.

Exemplo 34: Vamos acrescentar ao exemplo 33 uma função que imprimirá o valor do salário liquido.

```
linha 1  #include <stdio.h>
linha 2  /*DECLARANDO A FUNCAO CALCULAR_INSS*/
linha 3  float CALCULAR_INSS (float salario)
linha 4  {    float sal_liq;
linha 5       sal_liq= salario - (salario*0.08);
```

```
linha 6        return(sal_liq);

linha 7   }



linha 8   /*DECLARANDO A FUNCAO MOSTRAR*/

linha 9   void MOSTRAR(float salario)

linha 10 {     float salario_novo;

linha 11       salario_novo = CALCULAR_INSS(salario);

linha 12       printf("Salario liquido =%f\n\n",salario_novo);

linha 13 }



linha 14 //INICIANDO O PROGRAMA PRINCIPAL

linha 15 int main() {

linha 16      float salario_bruto, salario_novo;

linha 17      printf("Digite o salario do fucionario:");

linha 18      scanf ("%f",&salario_bruto);

linha 19      MOSTRAR(salario_bruto);

linha 20      system("PAUSE");

linha 21      return 0;
linha 22 }
```

Vamos entender algumas linhas:

linha 8...void MOSTRAR(float salario)
Informamos que a função MOSTRAR( ) retornará void (retorno vazio).

linha 10... salario_novo = CALCULAR_INSS(salario);
Agora a chamada à função CALCULAR_INSS( ) será feita dentro da função MOSTRAR( ), porque é nesse momento que precisaremos do valor de retorno que ela fornece.

linha 17... MOSTRAR(salario_bruto);
O procedimento é chamado apenas pelo nome.

Vejamos na Ilustração 43, como ficou o programa:

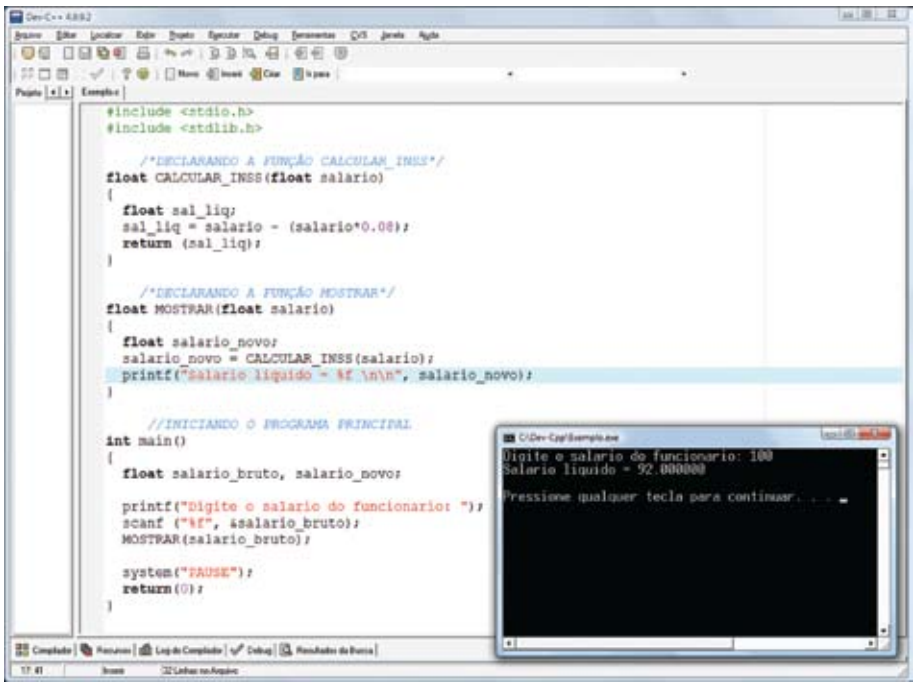


Ilustração 43: Compilação e execução do código fonte do Exemplo 34

Conceitos

função e/ou procedimento: Trecho de código que será utilizado muitas vezes no programa e/ou agrupamento de códigos correlatos. (LAUREANO, 1995, p.47).

Anotações

5.4 Escopo de Variáveis

Nos exemplos apresentados pôde-se observar a utilização de variáveis que foram declaradas em diferentes locais, (dentro de parênteses, fora deles), sem que houvesse preocupação, de nossa parte, de que tipo de comportamento elas estariam assumindo, quando o bloco de programa fosse acoplado ao programa maior (folha de pagamento).

Para entendermos esse comportamento, precisamos entender a definição de escopo de variáveis, e a passagem de parâmetros:

As variáveis podem assumir dois tipos de comportamentos:

- Locais: permanecem na memória durante a execução da função.
- Globais: permanecem na memória durante toda execução do programa principal e são declaradas fora das funções.

No programa folha de pagamento, a variável salário bruto será global, já que as funções utilizarão o seu valor para efetuarem os cálculos a que se propõem.

As variáveis que foram criadas dentro da função serão locais, só sendo visíveis àquela função.

As variáveis globais devem ser evitadas, pois ocupam memória o tempo todo, tornando o programa mais difícil de ser entendido. O código fica muito dependente, aumentando o acoplamento da função e tornando-a difícil de ser usada por outro programa.

E se for definida uma variável local com o mesmo nome de uma variável global?

escopo de variáveis: O escopo é o conjunto de regras que determinam o uso e a validade de variáveis nas diversas partes do programa. (BRIAN, 1978, p.35).

5.1.4 Passagem de Parâmetro

Parâmetros são valores que informamos à função, no momento em que a chamamos.

Chamamos essa informação de passagem de parâmetros, que pode ser feita de duas formas:

- **por valor:** mesmo que a variável sofra alteração, dentro da função, para onde foi passada, o valor dela não é alterado no programa principal.

Exemplo 35:

```
#include <stdio.h>
#include <stdlib.h>
void SOMA(int x, int y, int z)
{
     int resultado;
     resultado = x + y + z;
     x = 0; y = 1; z = 2;
     printf("num1+num2+num3= %d \n\n", resultado);
}

int main ()
{
    int num1, num2, num3;
    num1 = 10;     num2 = 20;     num3 = 30;
    SOMA(num1,num2,num3);
    printf("EXIBICAO DO VALOR DE num1 num2 num3:zn");
    printf(" num1 = %d \n", num1);
    printf(" num2 = %d \n", num2);
    printf(" num3 = %d \n\n", num3);
    system("PAUSE");
    return 0;
}
```

Pela Ilustração 44 poderemos verificar que, mesmo que os valores tenham sido alterados dentro da função SOMA, no programa principal eles ficaram inalterados.

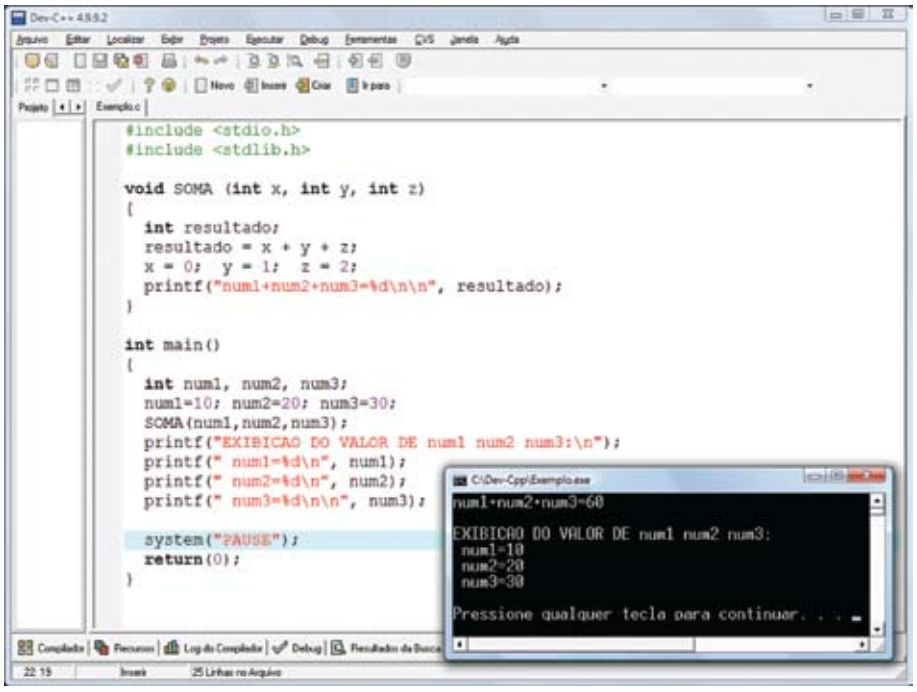


Ilustração 44: : Compilação e execução do código fonte do Exemplo 35

- **Por referência:** Na Linguagem C, utilizamos um recurso para simular a passagem de parâmetro por referência, declarando seus parâmetros formais como ponteiros. Os ponteiros são a "referência" de que precisamos para alterar a variável no programa principal.

  O único inconveniente é que, quando chamarmos a função (no programa principal, por exemplo), devemos colocar um & na frente das variáveis que estivermos passando para a função.

Exemplo 36: Usaremos o o Exemplo 34, mas agora passaremos as variáveis num1 e num2, por referência. Para isso, colocaremos um *(asterisco), transformado-as em um ponteiro.

```
#include <stdio.h>
#include <stdlib.h>
void SOMA(int *x, int *y, int z)
{   //x e y terao seus valores alterados no progr. principal
     int resultado;
     resultado = *x + *y + z;
     *x = 0;
     *y = 1;
      z = 2;
      printf("num1+num2+num3= %d \n\n", resultado);
}

int main ()
{
    int num1, num2, num3;
    num1 = 10;     num2 = 20;     num3 = 30;
    SOMA(&num1,&num2,num3);//colocar o & antes da var que será passada por referência
    printf("EXIBICAO DO VALOR DE num1 num2 num3:zn");
    printf(" num1 = %d \n", num1);
    printf(" num2 = %d \n", num2);
    printf(" num3 = %d \n\n", num3);
    system("PAUSE");
    return 0;
}
```

Na Ilustração 45 poderemos verificar que somente os valores das variáveis num1 e num2 foram alterados fora da função SOMA( ), isso porque a passagem dessas duas variáveis foram feitas por referência.

A variável num3, como foi passada por valor, só teve o seu valor alterado dentro da função SOMA( ).

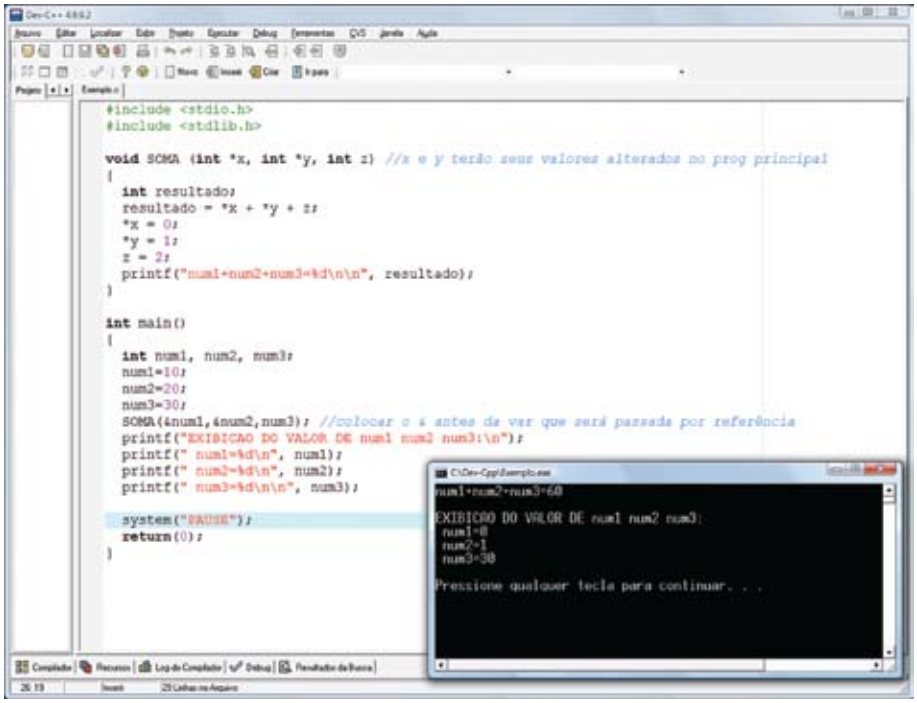


Ilustração 45: execução do exemplo 36.

Conceitos

parâmetro: o termo parâmetro muitas vezes é utilizado como sinônimo de argumento, mas geralmente utiliza-se "parâmetros" quando se faz referência às variáveis situadas na assinatura de um método ou função e "argumentos" aos valores atribuídos a esses parâmetros. (BRIAN, 1978, p.25).

passagem de parâmetro por referência: a mudança do valor de um parâmetro dentro de uma função afeta o valor da variável original. (BRIAN, 1978, p.26).

passagem de parâmetro por valor: Alterações no parâmetro não afetam a variável externa. (BRIAN, 1978, p.26).

Atividades

50. Crie um programa que usando uma sub_rotina de função retorne o valor da divisão de dois números inteiros para o programa principal e mostre o resultado.

FUNÇÃO MAIN – pedirá os dois números inteiro, chamará a função DIVISAO e exibirá o resultado.

FUNÇÃO DIVISAO– dividirá os números e retornará o valor da divisão para main( ). _____

_____

Atividades

51. Criar um programa que receba um número que corresponda a um mês do 1º trimestre e escreva o mês correspondente; caso o usuário digite o numero fora do intervalo deverá aparecer uma mensagem informando que o número é inválido. O programa deverá ser executado enquanto o usuário quiser.

52. Criar um programa que apresente um menu de seleções no programa principal. Esse menu deverá dar ao usuário a possibilidade de escolher uma entre quatro operações aritmética. Escolhida a opção desejada, deverá ser solicitada a entrada de dois números. Processada a operação o resultado deverá ser exibido. O programa deverá ser executado enquanto o usuário quiser.

Atividades

FUNÇÃO MAIN – Mostrará um menu de opções (“1–dividir”), (“2-somar”) , (“3–multip)”e (“4-subtrair”).

Pedirá dois números inteiros.

Utilizará o comando ***switch*** para chamar a função que corresponda a opção do usuário.

FUNÇÃO DIVIDIR – dividirá os valores lidos e mostrará o resultado.

FUNÇÃO SOMAR– somará os valores lidos e mostrará o resultado.

FUNÇÃO MULTIPLICAR– multiplicará os valores lidos e mostrará o resultado.

FUNÇÃO SUBTRAIR – subtrairá os valores lidos e mostrará o resultado.

_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____

5.1.5 Funções Recursivas

Podemos chamar uma função de dentro da própria função. Essa técnica é chamada de recursividade. Se uma função F chama a própria função F, dizemos que ocorre uma recursão. A implementação de alguns Algoritmos fica muito mais fácil, usando-se a recursividade. Quando uma função é chamada recursivamente, cria-se um ambiente local para cada chamada. As variáveis locais de chamadas recursivas são independentes entre si, como se estivéssemos chamando funções diferentes.

Exemplo 37: A série de Fibonacci é muito conhecida na matemática. A série é composta assim, o primeiro e o segundo termos são 1. A partir do terceiro termo, todos os termos são a soma dos dois últimos.

Série de Fibonacci: 1, 1, 2, 3, 5, 8, 13, 21, 34...

Façamos um programa para encontrar o enésimo termo da seqüência de Fibonacci.

```c
#include <stdio.h>
#include <stdlib.h>
int fib(int n)
{
    if (n>2)
        return ( fib(n-1) + fib(n-2) );
    else
        return 1;
}

main ()
{
    int n;
    printf("Digite um numero: ");
    scanf("%d", &n);
    printf("O termo %d da serie de Fibonacci e: %d\n", n, fib(n));
    system("PAUSE");
    return 0;
}
```

Na Ilustração 46, poderemos verificar a compilação e execução do Exemplo 37:

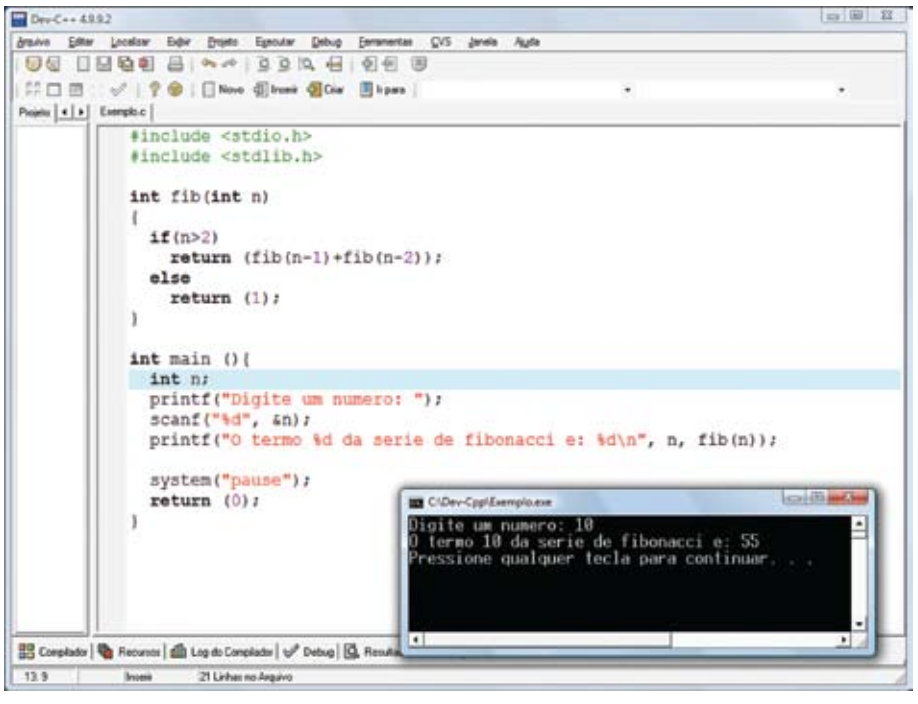


Ilustração 46: execução do exemplo 37

Conceitos

função recursiva: recursão é um método de programação no qual uma função pode chamar a si mesma. (LAUREANO, 2005, p. 239).

5.1.6 Protótipo de Função

Todas as funções que fizemos até aqui foram colocadas antes do programa principal (main( )), dessa forma, quando o compilador chegasse a ele, já saberia os formatos das funções.

Usando protótipo de função, poderemos colocar as funções após o programa principal. O protótipo é uma declaração de função, ou seja, declaramos a função que utilizaremos no programa. Dessa forma, o compilador tomará conhecimento do seu formato, antes da compilação, assim:

Exemplo 38: Vamos utilizar o Exemplo 36 e alterar seu código, incluindo o protótipo da função SOMA():

```
#include <stdio.h>
#include <stdlib.h>
void SOMA(int *x, int *y, int z);

int main ()
{
    int num1, num2, num3;
    num1 = 10;
    num2 = 20;
    num3 = 30;
    SOMA(&num1,&num2,num3); // colocar o & da var que será passada por referência
    printf("EXIBICAO DO VALOR DE num1 num2 num3:zn");
    printf(" num1 = %d \n", num1);
    printf(" num2 = %d \n", num2);
    printf(" num3 = %d \n\n", num3);
    system("PAUSE");
    return 0;
}

void SOMA(int *x, int *y, int z)
{   //x e y terao seus valores alterados no progr. principal
     int resultado;
     resultado = *x + *y + z;
     *x = 0;
     *y = 1;
     z = 2;
     printf("num1+num2+num3= %d \n\n", resultado);
}
```

Na Ilustração 47, poderemos verificar a compilação e execução do Exemplo 38. Observe que a declaração da função (**void SOMA(int *x, int *y, int z);**) deu-nos condições para alterarmos a posição do programa principal, colocando-o no início do código.

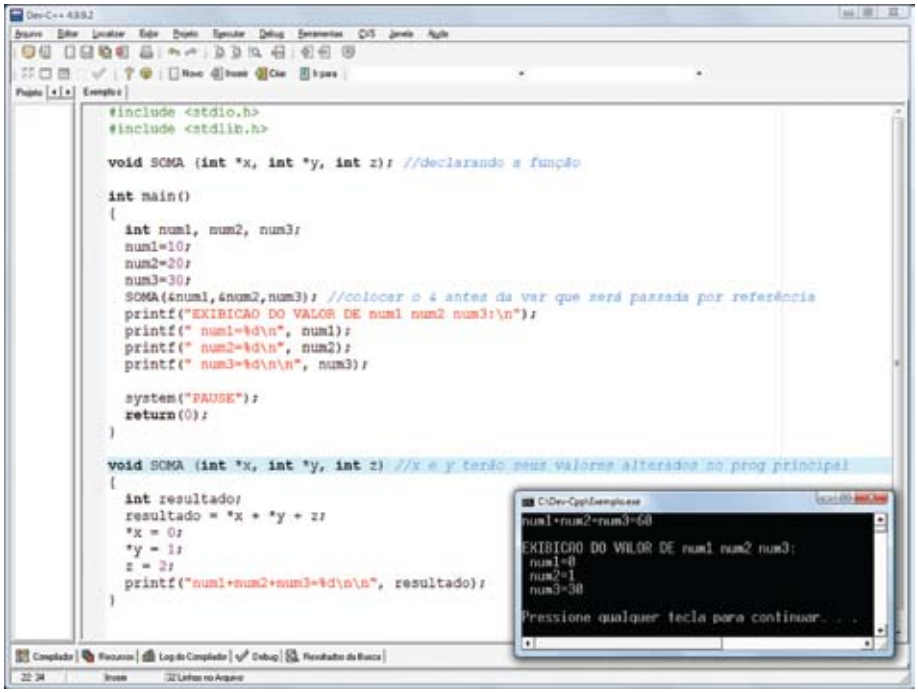


Ilustração 47: : Compilação e execução do código fonte do Exemplo 38

protótipo de função: assinatura da função ou definição da, indica ao compilador seu nome e parâmetros. (LAUREANO, 2005, p. 47).

Fala Professor

O que aprendemos até aqui?

- *Modularização é uma técnica de programação que utilizaremos para dividir um programa maior em programas menores. Aos programas menores denominamos funções ou procedimentos.*
- *As variáveis podem assumir dois tipos de comportamentos, locais e globais.*
- *A passagem de parâmetros, pode ser feita de duas formas: por valor e por referência.*
- *Os ponteiros são a "referência" de que precisamos para alterar a variável no programa principal.*
- *Podemos chamar uma função de dentro da própria função. Essa técnica é chamada de recursividade.*
- *As variáveis locais de chamadas recursivas são independentes entre si, como se estivéssemos chamando funções diferentes.*
- *O protótipo é uma declaração de função.*

Fala Professor

Atividades

53. Refaça os exercícios 50, 51 e 52, usando protótipo de função.

Referências Bibliográficas

FARRER, HARRY, **Algoritmos Estruturados**, LTC, 1999

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

DEITEL, H.M. **Como Programar em C**. Rio de Janeiro, 1999

SEBESTA, Robert W. **Conceitos de linguagem de programacao**. Sao Paulo: Bookman., 2003.

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

LAUREANO, Marcos **Programando em C**. Rio de Janeiro 2005,

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