Função main()

Essa função é responsável pelo início da execução do programa, e é dentro dela que colocamos os comandos que queremos que o programa execute.

O comando system(“pause”) serve para interromper a execução do programa (fazer uma pausa). Ela está definida dentro da biblioteca stdlib.h

A declaração de um comando quase sempre termina com ponto e vírgula (“;”)

As chaves definem o início (“{”) e o fim (“}”) de um bloco de comandos/instruções.

Os parênteses definem o início (“(”) e o fim (“)”) da lista de argumentos de uma função. Um argumento é a informação que será passada para a função agir

A indentação é o espaçamento (ou tabulação) colocado antes de começar a escrever o código na linha. Ela tem como objetivo indicar a hierarquia dos elementos

O código-fonte é normalmente escrito de uma forma que facilite a leitura pelos seres humanos, no caso o programador. Ele não tem nenhum significado para o computador, que somente entende códigos de máquina. Para fazer com que nosso código-fonte seja entendido como um programa pelo computador, é preciso traduzi-lo para esse código de máquina. A essa tradução damos o nome compilação e suas etapas são: pré processamento (expansão do código e interpretação); verificação sintatica(erros de sintaxe); compilacação (produção de instruções em linguagem de máquina); link edicao (arquivo executável, o programa).

Se o programador quiser comentar uma única linha do código, basta adicionar // na frente da linha. Tudo o que vier na linha depois do // será considerado comentário e ignorado pelo compilador. • Se o programador quiser comentar mais de uma linha do código, isto é, um bloco de linhas, basta adicionar /\* no começo da primeira linha de comentário e \*/ no final da última linha de comentário. Tudo o que vier depois do símbolo de /\* e antes do \*/ será considerado comentário e ignorado pelo compilador.

O comando #include é utilizado para declarar as bibliotecas que serão utilizadas pelo programa.

variável é uma posição de memória onde poderemos guardar determinado dado ou valor e modificá-lo ao longo da execução do programa.

tipo\_da\_variável nome\_da\_variável; O tipo\_da\_variável determina o conjunto de valores e de operações que uma variável aceita, ou seja, que ela pode executar. Já o nome\_da\_variável é como o programador identifica essa variável dentro do programa.

o operador de ponto e vírgula é utilizado para separar as instruções da linguagem C

Lembre-se: apenas quando declaramos uma variável é que o computador reserva um espaço de memória para guardarmos nossos dados. Assim, não é possível usar esse espaço antes de tê-lo reservado na memória.

O tipo char Comecemos pelo tipo char. Esse tipo de dados permite armazenar em um único byte (8 bits) um número inteiro muito pequeno ou o código de um caractere do conjunto de caracteres da tabela ASCII: char c = ‘a’; char n = 10; Caracteres sempre ficam entre aspas simples! Lembre-se: uma única letra pode ser o nome de uma variável. As aspas simples permitem que o compilador saiba que estamos inicializando nossa variável com um caractere e não com o conteúdo de outra variável.

O tipo int O segundo tipo de dado é o tipo inteiro: int. Esse tipo de dados permite armazenar um número inteiro (sem parte fracionária). Seu tamanho depende do processador em que o programa está rodando e é tipicamente 16 ou 32 bits: int n = 1459;

Os tipos float e double O terceiro e o quarto tipo de dados são os tipos reais: float e double. Esses tipos de dados permitem armazenar um valor real (com parte fracionária), também conhecido como ponto flutuante. A diferença entre eles é a sua precisão: • Tipo float: precisão simples. • Tipo double: dupla precisão. São úteis quando queremos trabalhar com intervalos de números reais realmente grandesA linguagem C usa o padrão numérico americano, ou seja, a parte decimal fica depois de um ponto. Veja os exemplos: float f = 5.25; double d = 15.673;

O tipo void Por fim, temos o tipo void. Esse tipo de dados permite declarar uma função que não retorna valor ou um ponteiro genérico, como será visto nas próximas seções.

A função printf() recebe dois parâmetros de entrada: • “tipos de saída”: conjunto de caracteres que especifica o formato dos dados a serem escritos e/ou o texto a ser escrito. • lista de variáveis: conjunto de nomes de variáveis, separados por vírgula, que serão escritos.

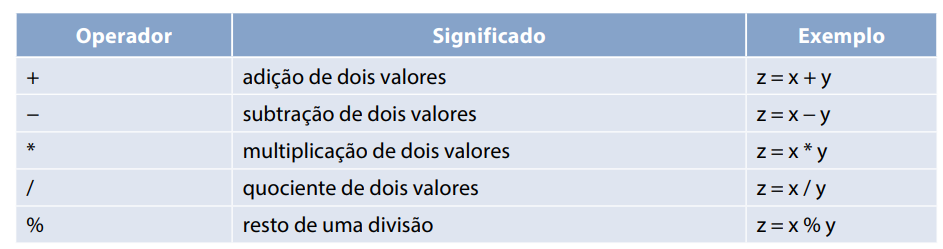
Diferentemente do comando scanf(), o comando printf() não exige o símbolo & na frente do nome de uma variável que será escrita na tela.

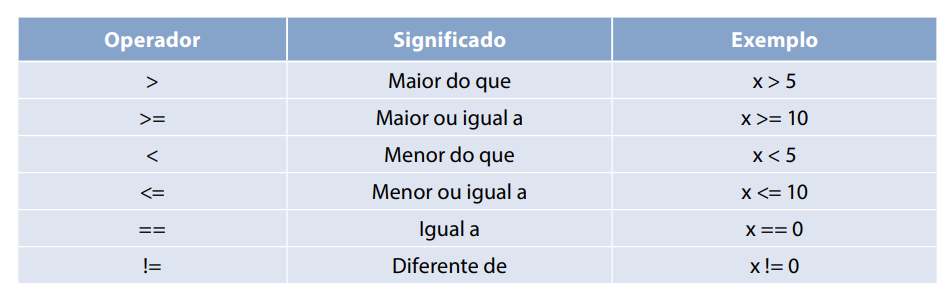
A função scanf() recebe dois parâmetros de entrada: • “tipos de entrada”: conjunto de caracteres que especifica o formato dos dados a serem lidos. • lista de variáveis: conjunto de nomes de variáveis que serão lidos e separados por vírgula, em que cada nome de variável é precedido pelo operador &.

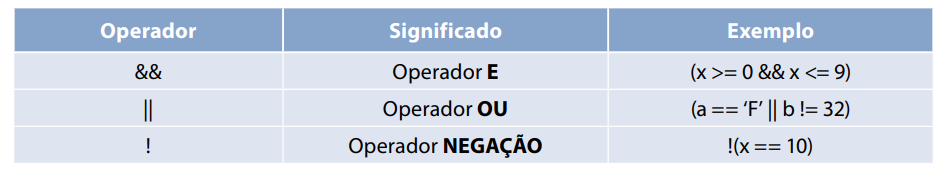
Uma variável declarada no escopo global, ou simplesmente variável global, é uma variável declarada fora de todas as funções do programa, ou seja, na área de declarações globais do programa (acima da cláusula main, juntamente com as bibliotecas do programa). Essas variáveis existem enquanto o programa estiver executando, ou seja, o tempo de vida de uma variável global é o tempo de execução do programa.

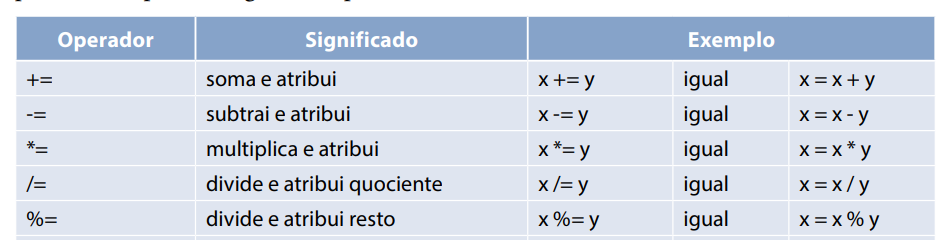
As variáveis globais devem ser evitadas porque qualquer parte do programa pode alterá-la. Isso prejudica a manutenção do programa, pois torna mais difícil saber onde a variável é inicializada, para que serve etc.

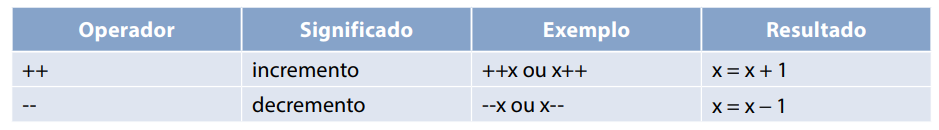
Uma variável declarada no escopo local, ou simplesmente variável local, é uma variável declarada dentro de um bloco de comandos delimitado pelo operador de chaves ({ }, escopo local).











Por exemplo, a expressão x = x + y − 10; pode ser reescrita usando o operador simplificado como x + = y − 10;

x \*= y − 10; equivale a x = x \* (y − 10); e não a x = x \* y − 10; x -= 5 + y; equivale a x = x − (5 + y); e não a x = x − 5 + y;

x = (y = 2, y + 3); Nesse caso, as expressões são executadas da esquerda para a direita: o valor 2 é atribuído a y, o valor 3 é somado a y, e o total (5) será atribuído à variável x.

Na linguagem C, o operador “,” é um separador de comandos, enquanto o operador “;” é um terminador de comandos.

CONDICIONAIS

uma expressão condicional é qualquer expressão que resulte em uma resposta do tipo verdadeiro ou falso. Ela pode ser construída utilizando operadores: • Matemáticos : +, −, \*, /, % • Relacionais: >, <, >=, <=, ==, != • Lógicos: &&, ||

Comando if Na linguagem C, o comando if é utilizado sempre que é necessário escolher entre dois caminhos dentro do programa ou quando se deseja executar um ou mais comandos que estejam sujeitos ao resultado de um teste. A forma geral de um comando if é: if(condição) { sequência de comandos; }

se o programador desejar que mais de uma instrução seja executada por aquele comando if, esse conjunto de instruções deve estar contido dentro de um bloco delimitado por chaves ({ }): if(condição) { comando 1; comando 2; ... comando n; }

O comando else é opcional, e sua sequência de comandos somente será executada se o valor da condição que está sendo testada pelo comando if for FALSA

if(condição) {

primeira sequência de comandos;

}

else{

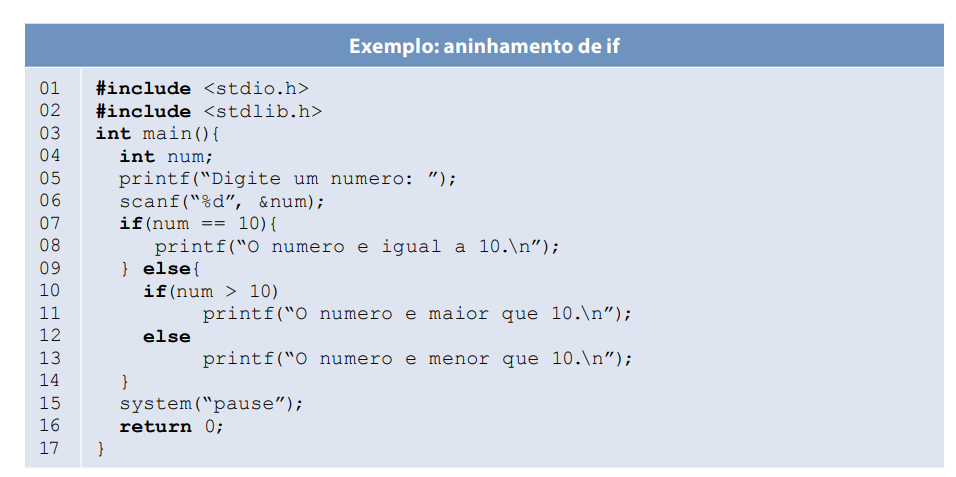
segunda sequência de comandos;

}

Se o comando if diz o que fazer quando a condição é verdadeira, o comando else trata da condição quando ela é falsa.

O comando else não tem condição. Ele é o caso contrário da condição de if.

A sequência de comandos de if é independente da sequência de comandos de else. Cada comando tem o seu próprio conjunto de chaves ({ }).



O comando else é o caso contrário da condição do comando if. Assim, para cada else deve existir um if anterior, porém nem todo if precisa ter um else.

O comando switch é indicado quando se deseja testar uma variável em relação a diversos valores preestabelecidos.

O comando default é opcional, e sua sequência de comandos somente será executada se o valor da variável que está sendo testada pelo comando switch não for igual a nenhum dos valores dos comandos case.

Note que a sequência de comandos a ser repetida está subordinada a uma condição. Por condição entende-se qualquer expressão relacional (ou seja, que use os operadores >, <, >=, <=, == ou !=) que resulte em uma resposta do tipo verdadeiro ou falso. A condição pode ainda ser uma expressão que utiliza operadores: • Matemáticos : +, −, \*, /, % • Relacionais: >, <, >=, <=, ==, != • Lógicos: &&, ||

Um laço infinito (ou loop infinito) é uma sequência de comandos em um programa de computador que sempre se repete, ou seja, infinitamente. Isso geralmente ocorre por algum erro de programação, quando: • Não definimos uma condição de parada. • A condição de parada existe, mas nunca é atingida.

O comando while equivale ao comando “enquanto” utilizado nos pseudocódigos apresentados até agora. A forma geral de um comando while é: while (condição){ sequência de comandos; }

O comando for é muito similar ao comando while, visto anteriormente. Basicamente, o comando for é usado para repetir um comando ou uma sequência de comandos diversas vezes. A forma geral de um comando for é: for (inicialização; condição; incremento) { sequência de comandos; }

Na execução do comando for, é realizada esta sequência de passos: • A cláusula inicialização é executada: nela as variáveis recebem um valor inicial para usar dentro do for. • A condição é testada: º Se a condição for considerada verdadeira (ou possuir valor diferente de zero), a sequência de comandos será executada. Ao final da sequência de comandos, o fluxo do programa é desviado para o incremento. º Se a condição for considerada falsa (ou possuir valor igual a zero), a sequência de comandos não será executada (fim do comando for). • Incremento: terminada a execução da sequência de comandos, ocorre a etapa de incremento das variáveis usadas no for. Ao final dessa etapa, o fluxo do programa é novamente desviado para a condição.

É muito útil quando se sabe de antemão quantas vezes a repetição deverá ser executada.

Este laço utiliza uma variável para controlar a contagem do loop, bem como seu incremento.

Trata-se de um comando bem enxuto, já que própria estrutura faz a inicialização, incremento e encerramento do laço.

O comando do-while é bastante semelhante ao comando while visto anteriormente. Sua principal diferença é com relação à avaliação da condição: enquanto o comando while avalia a condição para depois executar uma sequência de comandos, o comando do-while executa uma sequência de comandos para depois testar a condição.

A forma geral de um comando do-while é: do{ sequência de comandos; } while(condição);

O comando do-while é utilizado sempre que se desejar que a sequência de comandos seja executada pelo menos uma vez.

VETORES E MATRIZES

Um array ou “vetor” é a forma mais simples e comum de dados estruturados da linguagem C. Trata-se simplesmente de um conjunto de variáveis de um mesmo tipo, com a vantagem de estarem todas associadas ao mesmo nome e igualmente acessíveis por um índice.

Em linguagem C, a declaração de um array segue esta forma geral: tipo\_dado nome\_array[tamanho];

Acessando um elemento do vetor Como a variável que armazena a nota de um aluno possui agora o mesmo nome que as demais notas dos outros alunos, o acesso ao valor de cada nota é feito utilizando um índice float notas[100]; notas[0] = 81; notas[1] = 55; ... notas[99] = 72;

O uso de arrays permite usar comandos de repetição sobre tarefas que devem ser realizadas, de forma idêntica, para cada posição do array, apenas modificando o índice do array

Além disso, cada posição do array possui todas as características de uma variável. Isso significa que ela pode aparecer em comandos de entrada e saída de dados, expressões e atribuições. Por exemplo: scanf(“%d”,&notas[5]);//comando de leitura notas[0] = 10;//comando de atribuição notas[1] = notas[5] + notas[0];//expressão

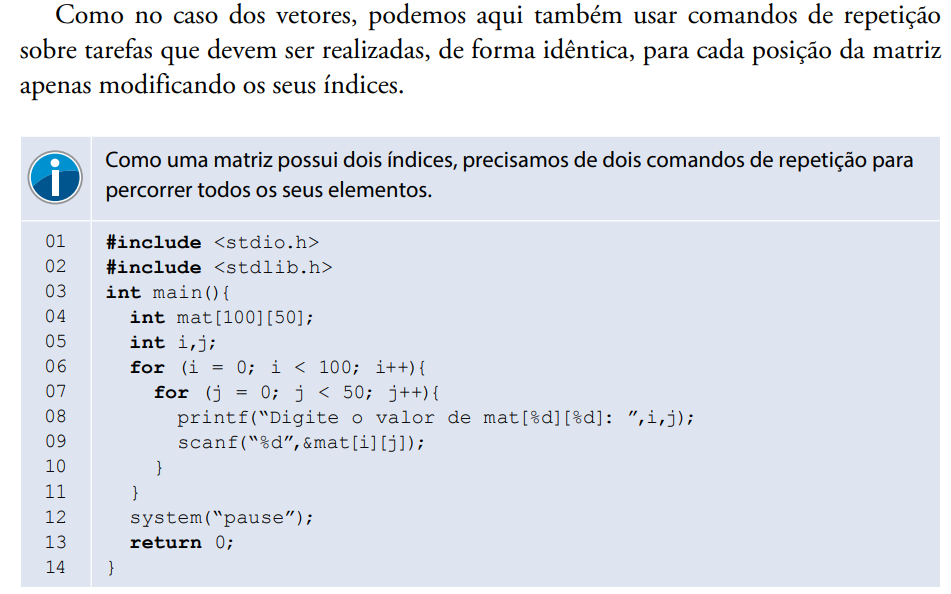
Lembre-se: cada posição do array é uma variável. Portanto, todas as posições do array são igualmente acessíveis, isto é, o tempo e o tipo de procedimento para acessar qualquer uma das posições do array são iguais aos de qualquer outra variável.

Na linguagem C, a numeração do índice do array começa sempre do ZERO e termina sempre em N-1, em que N é o número de elementos definido na declaração do array.Isso significa que, se o array para armazenar as notas dos alunos foi declarado como float notas[100]; as notas dos alunos serão indexadas de 0 a 99: notas[0]; notas[1]; ... notas[99];

Array com duas dimensões – matriz Os arrays declarados até o momento possuem apenas uma dimensão e, portanto, são tratados como uma lista de variáveis. Porém, há casos em que uma estrutura com mais de uma dimensão é mais útil. Por exemplo, quando os dados são organizados em uma estrutura de linhas e colunas, como uma tabela. Para isso, usamos um array com duas dimensões, ou seja, uma “matriz”

Em linguagem C, a declaração de uma matriz segue esta forma geral: tipo\_dado nome\_array[nro\_linhas][nro\_colunas];

Note que cada índice possui o seu próprio par de colchetes ([ ]). Como no caso do vetor, cada dimensão começa no índice ZERO e termina sempre em N-1, em que N é o número de elementos definido para aquela dimensão da matriz



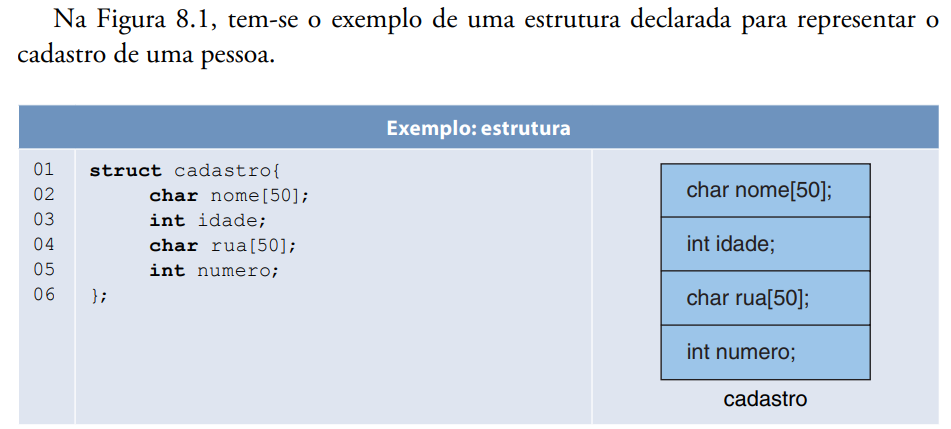
Como cada posição da matriz é uma variável, isso significa que ela possui as mesmas características de uma variável comum. Portanto, ela pode aparecer em comandos de entrada e saída de dados, expressões e atribuições, entre outros comandos: scanf(“%d”,&mat[5][0]);//comando de leitura mat[0][0] = 10;//comando de atribuição mat[1][2] = mat[5][0] + mat[0][0];//expressão

Para criar um novo tipo de dado, um dos seguintes comandos pode ser utilizado: • Estruturas: comando struct • Uniões: comando union

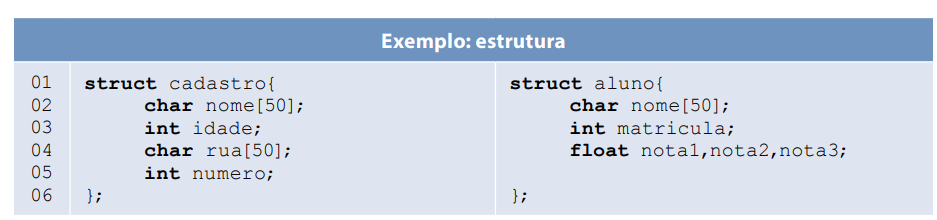
Estruturas: struct Uma estrutura pode ser vista como um conjunto de variáveis sob o mesmo nome, e cada uma delas pode ter qualquer tipo (ou o mesmo tipo). A ideia básica por trás da estrutura é criar apenas um tipo de dado que contenha vários membros, que nada mais são do que outras variáveis. Em outras palavras, estamos criando uma variável que contém dentro de si outras variáveis.

A forma geral da definição de uma nova estrutura utiliza o comando struct: struct nome\_struct{ tipo1 campo1; tipo2 campo2; ... tipon campoN; };

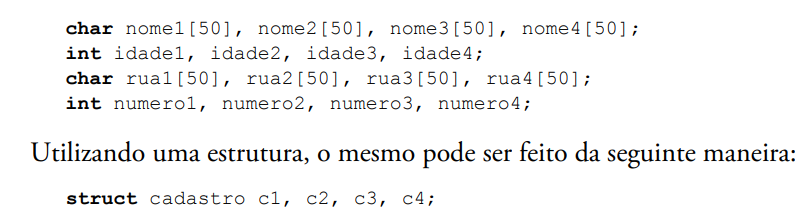
A principal vantagem do uso de estruturas é que agora podemos agrupar de forma organizada vários tipos de dados diferentes dentro de uma única variável.



Como na declaração de variáveis, os nomes dos membros de uma estrutura devem ser diferentes um do outro. Porém, estruturas diferentes podem ter membros com nomes iguais



Por ser um tipo definido pelo programador, usa-se a palavra struct antes do tipo da nova variável declarada.struct cadastro c;



O operador de acesso aos campos da estrutura é o ponto (.). Ele é usado para referenciar os campos de uma estrutura.

Como os campos nome e rua são strings, foi preciso usar a função strcpy() para copiar o valor para esses campos.

Inicialização de estruturas Assim como nos arrays, uma estrutura também pode ser inicializada, independentemente do tipo das variáveis contidas nela. Para tanto, na declaração da variável do tipo da estrutura, basta definir uma lista de valores separados por vírgula e delimitados pelo operador de chaves ({ }). struct cadastro c = { “Carlos”,18,“Avenida Brasil”,1082};

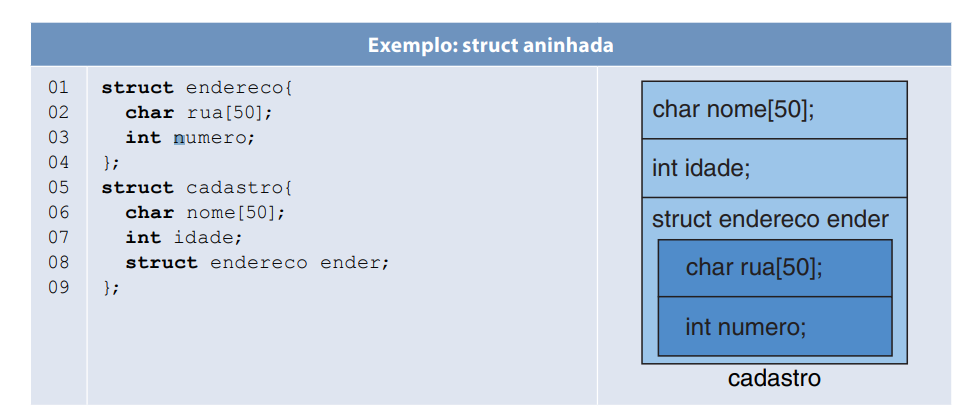
Nesse caso, como nos arrays, a ordem é mantida. Isso significa que o primeiro valor da inicialização será atribuído à primeira variável membro (nome) da estrutura, e assim por diante.

Imagine ter de declarar quatro cadastros para quatro pessoas diferentes: char nome1[50], nome2[50], nome3[50], nome4[50]; int idade1, idade2, idade3, idade4; char rua1[50], rua2[50], rua3[50], rua4[50]; int numero1, numero2, numero3, numero4; Utilizando uma estrutura, o mesmo pode ser feito da seguinte maneira: struct cadastro c1, c2, c3, c4; A representação desses quatro cadastros pode ser ainda mais simplificada se utilizarmos o conceito de arrays: struct cadastro c[4];

Em um array de estruturas, o operador de ponto (.) vem depois dos colchetes ([ ]) do índice do array.

Atribuições entre estruturas só podem ser feitas quando as estruturas são AS MESMAS, ou seja, quando possuem o mesmo nome!

r. Sendo assim, podemos declarar uma estrutura que possua uma variável do tipo de outra estrutura previamente definida. A uma estrutura que contenha outra estrutura dentro dela damos o nome estruturas aninhadas.



Em outras palavras, uma função é uma sequência de comandos que recebe um nome e pode ser chamada de qualquer parte do programa, quantas vezes forem necessárias, durante a sua execução.Um exemplo são as funções básicas de entrada e saída: scanf() e printf().

Duas são as principais razões para o uso de funções: • Estruturação dos programas. • Reutilização de código.

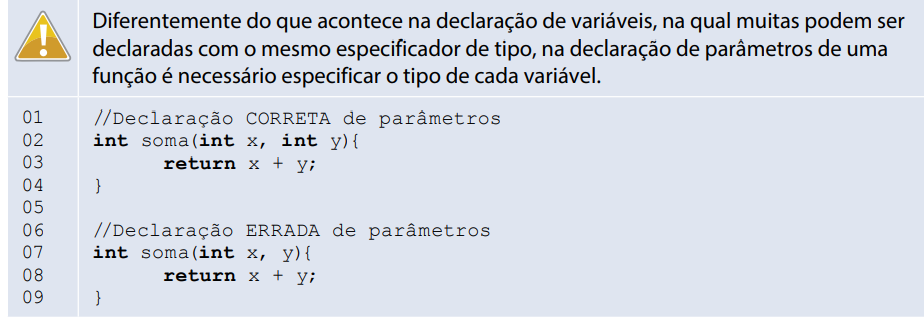
Programas grandes e complexos são construídos bloco a bloco com a ajuda de funções.

O uso de funções evita a cópia desnecessária de trechos de código que realizam a mesma tarefa, diminuindo assim o tamanho do programa e a ocorrência de erros.

Em linguagem C, a declaração de uma função pelo programador segue esta forma geral: tipo\_retornado nome\_função (lista\_de\_parâmetros){ sequência de declarações e comandos }

Com relação ao local de declaração de uma função, ela deve ser definida ou declarada antes de ser utilizada, ou seja, antes da cláusula main()

Declarando os parâmetros de uma função Em linguagem C, a declaração dos parâmetros de uma função segue esta forma geral: tipo\_retornado nome\_função (tipo nome1, tipo nome2, ... , tipo nomeN){ sequência de declarações e comandos }



Dependendo da função, ela pode não possuir nenhum parâmetro. Nesse caso, pode-se optar por duas soluções: • Deixar a lista de parâmetros vazia: void imprime(). • Colocar void entre parênteses: void imprime(void).

O parâmetro de uma função é uma variável local da função e, portanto, só pode ser acessado dentro da função.

Basicamente, o corpo da função é formado por: • Sequência de declarações: variáveis, constantes, arrays etc. • Sequência de comandos: comandos condicionais, de repetição, chamada de outras funções etc.

Operações de leitura e escrita não são proibidas dentro de uma função. Apenas não devem ser usadas se esse não for o foco da função.

A expressão tipo\_retornado estabelece o tipo de valor que a função vai devolver para quem a chamar. Uma função pode retornar qualquer tipo válido na linguagem C: • Tipos básicos predefinidos: int, char, float, double, void e ponteiros. • Tipos definidos pelo programador: struct, array (indiretamente) etc.

Funções com retorno de valor Se a função não for do tipo void, ela deverá retornar um valor. O comando return é utilizado para retornar esse valor para o programa: return expressão;