UNIVERSIDADE FUMEC

Estrutura de Dados

CONCEITOS BÁSICOS E OPERADORES

APRESENTAÇÃO

Prezado aluno(a), seja bem-vindo à disciplina de Estrutura de Dados, a partir de agora iniciaremos juntos uma jornada para compreender melhor como aproveitar e utilizar as estruturas das nossas máquinas em benefícios de desenvolver melhores códigos.

Para esta disciplina utilizaremos, como ferramenta de aprendizagem, a linguagem C, por isso, neste módulo conheceremos os conceitos básicos, necessários à compreensão e utilização das estruturas de dados. Muitos destes conceitos já podem ser do seu conhecimento, mas é sempre bom relembrar para começarmos no mesmo nível.

Além disto, também saberemos quais operadores são permitidos para a linguagem citada e como criar expressões com tais operadores.

Espero que esteja animado para aprender, e ensinar, muito, pois este é o propósito principal. Se for necessário, acesse este material para tirar dúvidas ou fazer alguma consulta, e lembre-se que estarei aqui para auxiliá-lo.

Abraços e bons estudos.

OBJETIVOS DE APRENDIZAGEM

Ao final desse módulo você deverá ser capaz de:

- Assimilar as características básicas da linguagem C;
- Relacionar a estrutura e as necessidades da escrita de programas em C;
- Identificar os operadores da linguagem;
- Aplicar estes operadores em expressões.

FICHA TÉCNICA

UNIVERSIDADE FUMEC

FUMEC VIRTUAL - SETOR DE EDUCAÇÃO A DISTÂNCIA

Gestão Pedagógica
Coordenação
Gabrielle Nunes Paixão
Transposição Pedagógica
Talita Miranda Leite

TECNOLOGIA DA INFORMAÇÃO

Produção Multimídia
Rodrigo Tito M. Valadares
Paulo Roberto Rosa Júnior

Infra-Estrututura e Suporte
Coordenação
Anderson Peixoto da Silva

AUTORIA

Prof.ª Amanda Danielle

BELO HORIZONTE - 2020

CONCEITOS BÁSICOS E OPERADORES

Introdução

Para esta disciplina, nossa ferramenta de aprendizagem será a linguagem C, mas, gostaria de avisá-lo que, mais que desenvolver programas, precisamos compreender a execução e lógica envolvidas nas soluções.

Quando você começar a pensar nas soluções, utilizando as inúmeras possibilidades que a linguagem oferece, perceberá que o código é apenas um complemento para implementar tais resultados.

Vamos lá?! Está preparado para pensar "fora da caixinha"?!

A linguagem C

Primeiro conheceremos um pouco da linguagem escolhida, a linguagem C foi criada por Dennis Ritchie, Martin Richards e Ken Thompson, na década de 70.

É uma linguagem que evoluiu a partir de outras, e sua grande força está na capacidade de controlar a memória e modularizar seu código (Calma!! Falaremos bastante sobre isso.)

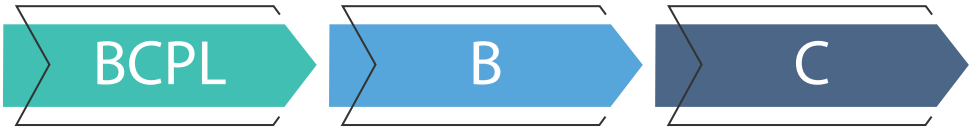


Fig. 1 : Evolução da Linguagem C.

No mercado de trabalho é considerada uma linguagem poderosa, responsável pelo desenvolvimento de diversos programas conhecidos, e muito utilizados, por nós, dentre eles sistemas operacionais, planilhas eletrônicas, processadores de textos, jogos e outros.

IMPORTANTE

Uma característica importante, que não podemos esquecer, é que ela é ***case sensitive***, ou seja, ela diferencia letras maiúsculas de letras minúsculas, portanto, para esta linguagem o "A" e "a" são letras diferentes.

Por isso, se você escrever um comando com alguma letra maiúscula, sendo que a linguagem exige que seja minúscula, este comando será indicado como incorreto. (Não esqueça!! ***Case sensitive***!)

Estrutura Básica

Para escrever qualquer programa em C, é preciso "obedecer" à sua estrutura básica, isto é o que chamamos de sintaxe, que exige alguns itens relevantes:

- **Declarações globais:** são declarações em que todas as funções do código terão acesso, como por exemplo variáveis globais, diretivas de compilação, declaração de estruturas, declaração de macros ou constantes, etc.
- **Funções do desenvolvedor:** são funções que foram desenvolvidas por nós e que serão utilizadas durante o código
- **Função principal:** Responsável por iniciar a execução do código.

Vamos começar por algumas declarações globais que são importantes para o bom funcionamento do código:

DIRETIVAS DE COMPILAÇÃO

O primeiro é a declaração de diretivas de compilação. Uma diretiva é uma instrução enviada ao pré-processador e que será executada antes da compilação do seu código fonte.

Vamos compreender melhor esse processo? Então vem comigo!

TOME NOTA

Pré-processador é o programa que efetua modificações e execuções necessárias à implementação do código;

Código fonte é a escrita do seu programa, obedecendo a sintaxe e com todos os comandos que você precisar utilizar;

E, por fim, compilação, que na verdade, é a tradução do seu código fonte para o código de máquina, afinal, o computador não fala a mesma língua que nós, ele precisa que você envie os comandos em linguagem binária, e quem é responsável por essa tradução é o compilador.

Toda diretiva inicia-se com um # mas ela não pertence a linguagem C, por isso, não precisa finalizar com ; (ponto e vírgula). O ideal é que o texto da sua diretiva seja curto, mas, se for necessário utilizar mais de uma linha, coloque um \ para indicar que termina na linha seguinte. Agora veremos algumas diretivas...

A Diretiva Include

Para nossa utilização inicial, falaremos da diretiva *include*, pois ela será utilizada desde o nosso primeiro programa.

Ao usarmos a diretiva *include*, estamos incluindo o arquivo de biblioteca em nosso código fonte.

Imagine a seguinte situação, antes de efetuar a compilação, o pré-processador inclui o arquivo que você solicitou e só depois compila o código completo, como se tudo pertencesse a sua escrita.

Existem duas formas de declarar, e utilizar, uma diretiva em seu código, observe as diferenças entre elas:

- Para utilizar arquivos que pertencem a linguagem C, e que estão na pasta include, utilize a seguinte sintaxe - `#include <nome do arquivo.h>`.
- Para utilizar arquivos de uma biblioteca que possua funções criadas pelo próprio desenvolvedor, utilize a seguinte sintaxe - `#include "nome do arquivo.h"`.

Para este último caso o pré-processador buscará até 10 níveis internos para localizar o arquivo indicado pelo desenvolvedor.

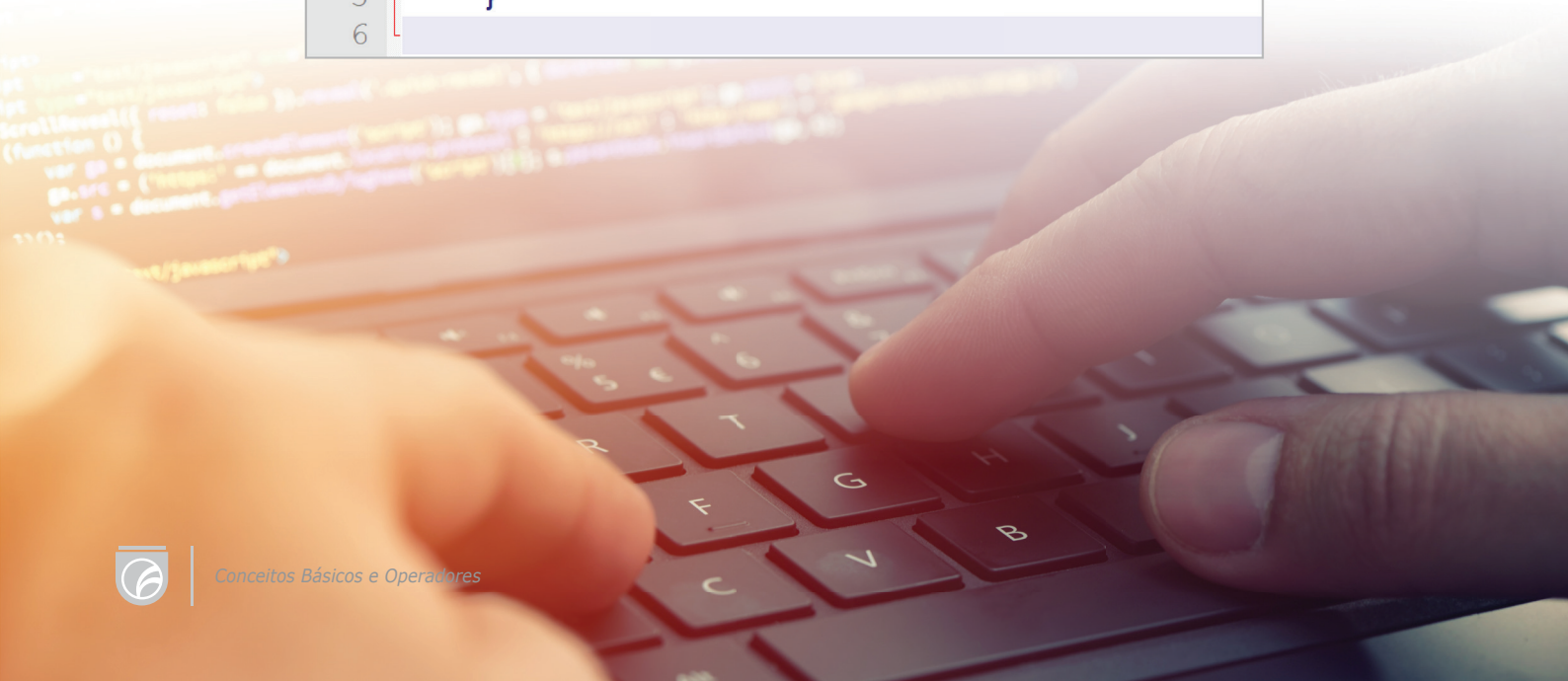
Diretivas comuns em nossa utilização:

`#include <stdio.h>`	Funções de Entrada e Saída
`#include <stdlib.h>`	Biblioteca padrão da linguagem C
`#include <math.h>`	Funções matemáticas
`#include <string.h>`	Manipulação de Strings

Um exemplo comum da necessidade de utilização de diretivas é a função `printf`, ela é responsável por exibir informações na tela do computador, para que o usuário acompanhe a execução.

Esta função pertence a biblioteca `stdio`, portanto, para utilizá-la, você precisa incluir a biblioteca em seu código, através da linha de comando `#include <stdio.h>`.

```
1  #include <stdio.h> //Diretiva
2  int main (){
3      printf ("Meu programa começa aqui .");
4      return 0;
5      }
6
```



A Diretiva Define

A diretiva *define* é utilizada para definir uma constante ou uma macro em seu código, assim, quando você precisar desses itens ele buscará os valores correspondentes e retornará no código.

Um exemplo da necessidade, e declaração, de constante é quando você precisa de um valor em seu código, e tem certeza que durante a execução do programa este conteúdo nunca poderá ser alterado, para estes casos, o ideal é criar constantes, afinal, este valor será o mesmo durante todo o programa.

Para esta declaração utilize a diretiva *define*, assim ela associará o conteúdo definido ao nome da constante, como por exemplo na linha #define TAM 5 – para este caso estamos definindo que a constante identificada como TAM terá o valor 5 em qualquer lugar em que for utilizada.

```
1  #include <stdio.h> //Diretiva
2  #define TAM 5 //Diretiva
3  int main (){
4      printf ("O valor da constante é %d .", TAM);
5      return 0;
6  }
7
```

E sempre que você solicitar a diretiva TAM ela retornará o valor 5 como conteúdo.

A diretiva define também nos permite criar macros em nosso código, a diferença de uma macro para uma constante é que as macros executam algum código e aguardam resultado.

Vamos a um exemplo para que você compreenda melhor... imagine que você pretende calcular o quadrado de um número, passado como parâmetro, para isto você pode criar uma macro e executar no pré-processador, de forma mais eficiente.

Veja no código como ficaria uma macro como essa do exemplo:

```
1  #include <stdio.h> //Diretiva
2  #define QUAD(a) (a) * (a)  //Diretiva
3  int main (){
4      printf ("O quadrado de 7 é  %d .", QUAD(7));
5      return 0;
6  }
7
```

Ao declarar a macro #define `QUAD(a) (a) * (a)` estamos considerando que o nome dela é QUAD e sempre que for utilizada precisará de um valor para efetuar o cálculo, este valor é conhecido como parâmetro. Este parâmetro será multiplicado por ele mesmo e o resultado será retornado para a parte do código que o solicitou.

Mas atenção, observe que todas as diretivas de compilação estão criando itens no pré-processador, por isso, não são compiladas dentro do seu código diretamente.

FUNÇÃO PRINCIPAL

A execução de um programa em C inicia-se através da função principal, conhecida como *main*.

É a partir dela que o código chama qualquer outra função, portanto, todo o programa que executa a linguagem C começa pela função *main*. Quem a executa é o pré-processamento e esta função retorna um valor inteiro a ele, indicando se foi executada com sucesso, retorno zero, ou se houve algum erro, retorno 1.

```
1  int main (){ // Cabeçalho da função principal
2      printf ("Meu programa começa aqui ."); // Bloco de comandos
3      return 0; // Retorno da função - zero indica que finalizou com sucesso
4  } // Finalização da função
5
```

Dentro desta função você poderá incluir qualquer comando que precisar para executar suas tarefas.

Comentários

Em um código fonte, os comentários servem para orientar o desenvolvedor, enviando mensagens ou observações que poderão ser relevantes a ele ou a quem estiver lendo aquele código.

Esses comentários não serão considerados pelo compilador, ou seja, não serão compilados para a execução do programa.

Comentar uma linha

Para comentar uma linha, utilize //, assim, o compilador compreenderá que do início das barras até o final daquela linha você está fazendo um comentário.

Comentar um bloco (várias linhas)

Eventualmente precisamos fazer comentários maiores, com duas ou mais linhas, para isto utilize /* para indicar o início do comentário e */ para indicar o final do comentário.

```
1  Int main (){ // Comentários linha
2      /* Comentário bloco
3         posso utilizar para várias linhas */
4      return 0;
5  }
6
```

OPERADORES

Em linguagens de programação, os operadores permitem a execução de alguma expressão ou avaliação lógica, para que seja interpretada e executada em seu programa.

É comum que o código aguarde o resultado da expressão, escrita com operadores, para continuar a executar as próximas linhas do programa.

Veja a seguir os tipos de operadores disponíveis para utilização.

Operadores aritméticos

Os operadores aritméticos permitem execuções matemáticas ao código. Para isto, é necessário aplicar algum dos seguintes itens:

OPERADORES MATEMÁTICOS		
Operador	**Descrição**	**Exemplos**
+	Soma	15 + 10 69.89 + x x + y
-	Subtração	15 - 1, 69.89 - x x - y
*	Multiplicação	15 * 10 69.89 * x x * y
/	Divisão	15 / 10 69.89 / x x / y
%	Resto	15 % 10 69.89 % x x % y
+ +	Incremento	cont = 10 cont + + incrementa 1 na variável cont, neste caso, o valor final será cont = 11
--	Decremento	cont = 10 cont-- decrementa 1 na variável cont, neste caso, o valor final será cont = 9

ATENÇÃO

Para o operador de resto, no cálculo 15 % 10 = 5, pois sabe-se que o resto de 15 dividido por 10 é 5, então, esse será o resultado desta expressão (15 / 10 = 1 e tem resto 5).

Operadores relacionais

Ao utilizarmos um operador relacional, faremos uma avaliação entre DUAS expressões. Se necessário, cada expressão poderá utilizar operadores aritméticos, sendo que as expressões aritméticas serão solucionadas antes das expressões relacionais.

OPERADORES RELACIONAIS		
Operador	**Descrição**	**Exemplos**
= =	Igualdade	x – y = = 12 * a 12 = = 56.9 -23.8 = = x + y
! =	Diferença	cont ! = 12 1 ! = 5.9 -x + 23 ! = y / 5
>	Maior	soma * 10 > x / 2 1.8 > 0.5 -x > 5.4 / 2
> =	Maior ou igual	y % 2 > = 54 % 3 189.8 > = 65 y > = x * 2
<	Menor	idade < menorIdade 0.8 < 6.5 x - 34 < 23 * 54
< =	Menor ou igual	y - w < = cont * 72 189.8 < = x y < = 72

É importante saber que TODA avaliação relacional terá seu resultado como verdadeiro (true - 1) ou falso (false - 0).

Vamos ver alguns exemplos para compreender melhor?

Considere os seguintes valores para ambos: **soma = 235** e **cont = 0.**

Primeiro exemplo:

```
soma    >=    cont * 10
235     >=    0 * 10
235     >=    0
1 (verdadeiro)
```

Segundo exemplo:

```
soma % 10    ==    cont + 10 * 5
235 % 10     ==    0 + 10 * 5
5            ==    50
0 (falso)
```

OBSERVAÇÕES

soma % 10 - o resto da divisão de 235 por 10 é igual a 5.

O processador sempre executará qualquer avaliação da seguinte forma: “pegará” o resultado, como verdadeiro ou falso, e só depois, dependendo do comando utilizado, ele irá decidir o que fazer.

Operadores lógicos

Vimos que operadores relacionais permitem a verificação entre duas expressões, mas, em alguns casos, precisamos avaliar mais expressões ao mesmo tempo, para isto, utiliza-se os operadores lógicos, eles permitem a execução de mais de um relacionamento. Assim, sua avaliação será ampliada para diversas análises. Vamos verificar quais são esses operadores?

OPERADORES LÓGICOS	
Operador	**Descrição**
&&	E (conjunção)
||	OU (disjunção)
	NÃO (negação)

Agora, para que você compreenda melhor, falaremos de cada operador. Vamos nessa?

OPERADOR LÓGICO E - &&

O operador lógico E, efetua a junção das expressões avaliadas. Estas expressões devem ser lógicas, ou seja, devem ter resultados verdadeiro ou falso. Em C, utilizamos os caracteres && para representá-lo.

Sempre que precisar utilizar o operador lógico E, você deve considerar que todas as expressões testadas devem ser verdadeiras para que o resultado final seja verdadeiro, caso contrário, o resultado será falso.

Portanto, se tivermos 1000 expressões, onde 999 são verdadeiras e 1 falsa, o resultado final será falso. Vamos observar a tabela verdade deste operador?

Expressão 1		Expressão 2		Resultado
23 > 15	&&	76 = = 0.76 * 10	=	Verdadeiro
Verdadeiro	&&	Verdadeiro	=	Verdadeiro
5 < 97 % 10	&&	5.12 != 512 / 100	=	Falso
Verdadeiro	&&	Falso	=	Falso
6.98 >= 13	&&	34 <= 98 + 5	=	Falso
Falso	&&	Verdadeiro	=	Falso
65 % 5 > 13 - 7	&&	12 = = 2287	=	Falso
Falso	&&	Falso	=	Falso

Na tabela foram utilizadas apenas duas expressões, mas pode-se aplicar quantas forem necessárias, desde que sejam “ligadas” por um operador lógico.

OPERADOR LÓGICO OU - ||

O operador lógico OU, efetua a disjunção (separação) das expressões avaliadas. Estas expressões devem ser lógicas, ou seja, devem ter resultados verdadeiro ou falso. Em C, utilizamos os caracteres || para representá-lo.

Sempre que precisar utilizar o operador lógico OU, você deve considerar que NÃO será necessário que todas as expressões sejam verdadeiras, basta que uma seja verdadeira para que o resultado final seja verdadeiro. O resultado final só será falso se TODAS as expressões forem falsas.

Portanto, se tivermos 1000 expressões, onde 999 são falsas e 1 verdadeira, o resultado final será verdadeiro.

Vamos observar a tabela verdade deste operador:

Expressão 1		Expressão 2		Resultado
23 > 15	&&	76 = = 0.76 * 10	=	Verdadeiro
Verdadeiro	&&	Verdadeiro	=	Verdadeiro
5 < 97 % 10	&&	5.12 != 512 / 100	=	Verdadeiro
Verdadeiro	&&	Falso	=	Verdadeiro
6.98 >= 13	&&	34 <= 98 + 5	=	Verdadeiro
Falso	&&	Verdadeiro	=	Verdadeiro
65 % 5 > 13 - 7	&&	12 = = 2287	=	Falso
Falso	&&	Falso	=	Falso

Na tabela foram utilizadas apenas duas expressões, mas pode-se aplicar quantas forem necessárias, desde que sejam “ligadas” por um operador lógico.

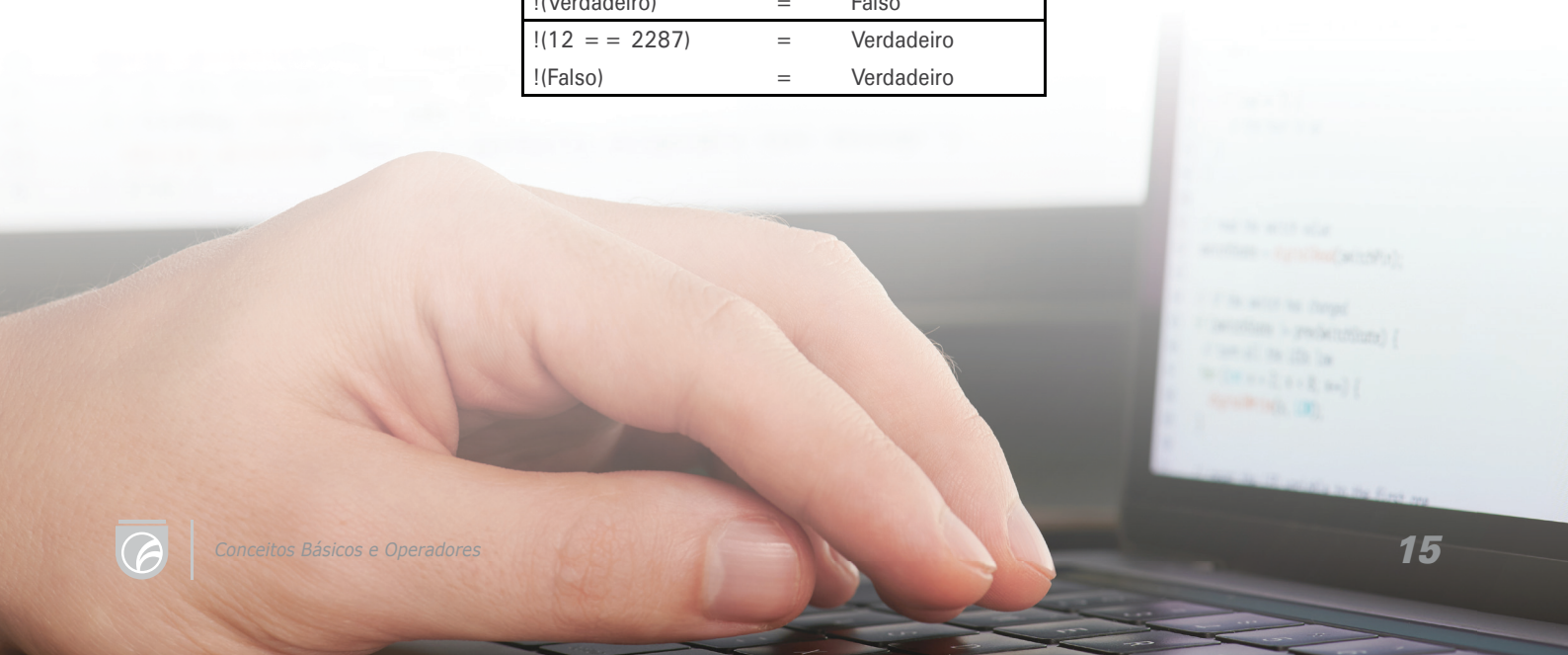
OPERADOR LÓGICO NÃO - !

O operador lógico NÃO efetua uma negação na expressão avaliada. Isso significa que irá inverter o valor original da expressão negada. Em C, utilizamos o caractere ! para representar este operador.

Sempre que precisar utilizar o operador lógico NÃO, a ideia é que o resultado atual será invertido, ou seja, se a expressão for verdadeira, o resultado final é igual a falso, afinal, NÃO verdadeiro será falso (e vice-versa).

Vamos observar a tabela verdade deste operador:

!(Expressão 1)		Resultado
!(23 > 15)	=	Falso
!(Verdadeiro)	=	Falso
!(12 = = 2287)	=	Verdadeiro
!(Falso)	=	Verdadeiro



Síntese

Bem aluno(a), terminamos aqui o conteúdo deste módulo.

Conversamos sobre os conceitos básicos da linguagem C, que nos apoiará na compreensão das Estruturas de Dados.

Iniciamos pela estrutura básica da linguagem e nos itens que são necessários para escrever um programa, você aprendeu sobre as diretivas de compilação, viu que são códigos que não pertencem à linguagem, mas que são agregados a ela quando ocorre a compilação, descobriu também que é a partir da função principal que todos os comandos escritos no código fonte são executados.

Falamos sobre a possibilidade de incluir comentários em seu código, esses comentários não serão considerados durante a compilação, mas são importantes para que as pessoas que leem o código tenham informações sobre ele.

Por fim, você também teve contato com os operadores desta linguagem, e a partir de agora sabe que poderá utilizar operadores aritméticos que executará operações matemáticas em seu código; operadores relacionais que efetuará comparações entre expressões; e, por último, operadores lógicos que permitirá que diversas relações sejam realizadas.

Até mais!

Referências

DEITEL, Paul; DEITEL, Harvey. **C:** como programar. 6. ed. São Paulo: Pearson Prentice Hall, 2011. 820 p.

MIZRAHI, Victorine Viviane. **Treinamento em linguagem C**. São Paulo: Pearson Prentice Hall, 2008. 407 p.

SOFFNER, Renato. **Algoritmos e programação em linguagem C.** São Paulo: Saraiva, 2013. 196 p.