ESTRUTURAS DE CONTROLE EM C

Operações matemáticas em C - Soma, subtração, multiplicação, divisão e módulo (ou resto da divisão) e precedência dos operadores

Operações matemáticas básicas. Fácil não?
Por exemplo, quanto é: **1 + 1 x 2** ?

Pode ser 3: **1 + (1x2) = 1 + 2 = 3**
Ou pode ser 4: **(1+1)x2 = 2x2= 4**

Então, qual a resposta certa? Obviamente é só uma.
Para o computador, um cálculo não pode resultar em dois valores.

A resposta correta é sempre 3. Isso tem a ver com operações matemáticas e precedência em C.

Símbolos matemáticos em C

Em programação C, alguns símbolos matemáticos são que usamos em computação são diferentes daqueles que usamos no dia-a-dia. Os símbolos são os seguintes:
Soma: +
Subtração: -
Multiplicação: *
Divisão: /
Módulo ou resto da divisão: %

Como pode notar, o símbolo de multiplicação é um asterisco, e não um 'x'.
O símbolo da divisão é a mesma barra que usamos para digitar urls.
Sobre módulo (ou resto da divisão), falaremos melhor sobre esse operador matemático ainda nesse artigo de nosso curso.

A seguir, um programa que pede dois números para o usuário e mostra o resultado da soma, subtração, multiplicação e divisão do primeiro pelo segundo número.
Note que é bem simples de se entender e não há segredo:

```
#include <stdio.h>

int main()
{
    float num1, num2, sum, sub, mult, div;

    printf("Digite o primeiro numero: " );
    scanf("%f", &num1);

    printf("Digite o segundo numero: " );
    scanf("%f", &num2);

    //Soma
    sum = num1 + num2;

    //Subtração
    sub = num1 - num2;
```

```
    //Multiplicaçao
    mult = num1 * num2;

    //Divisão
    div = num1/num2;

    printf("%.2f + %.2f = %.2f\n", num1, num2, sum);
    printf("%.2f - %.2f = %.2f\n", num1, num2, sub);
    printf("%.2f * %.2f = %.2f\n", num1, num2, mult);
    printf("%.2f / %.2f = %.2f\n", num1, num2, div);
}
```

Além dos dois números que pedimos ao usuário, criamos mais 4 float para armazenar as operações matemáticas. Fizemos isso para o código ficar organizado, mas você também pode colocar essas operações direto no printf, veja como ficariam os printf:

```
printf("%.2f + %.2f = %.2f\n", num1, num2, num1 + num2);
printf("%.2f - %.2f = %.2f\n", num1, num2, num1 - num2);
printf("%.2f * %.2f = %.2f\n", num1, num2, num1 * num2);
printf("%.2f / %.2f = %.2f\n", num1, num2, num1 / num2);
```

O que é módulo ou operador resto da divisão

Vamos voltar um pouco no tempo, quando você aprendeu a fazer continhas, mais ou menos como na figura:

Pois é, nesse caso o resto da divisão é o 14.
Ela é resultado da seguinte operação: 2009 % 19 = 14
Pois 2009 = 19*105 + 14

Essa operação será bastante usada no decorrer de sua vida como programador C.
Por exemplo, os números pares sempre deixa resto 0 quando divididos por 2 (consegue entender o motivo?).

Precedência dos operadores matemáticos em C

Vamos voltar ao exemplo do início de nosso tutorial: quanto seria 1 + 1 x 2
A resposta é sempre 3 pois o operador de multiplicação é mais importante que a soma.

E caso fosse: 8/4 - 2?
Pode ser: 8/4 - 2 = 2 - 2 = 0
Ou pode ser: 8/4 - 2 = 8/(4 - 2) = 8 / 2 = 4
A resposta é sempre 0, pois o operador de divisão é mais importante que o de subtração.

Mostramos esses exemplos para provar que saber a ordem (ou precedência) dos operadores é muito importante.
Embora a maioria dos cursos (alguns ditos 'bons' e famosos) simplesmente ignore isso, mas você viu que pode obter respostas totalmente diferentes.

Segue a lista dos operadores matemáticos que estudamos em nosso artigo.
A importância vai crescendo de **baixo para cima** e da **esquerda para a direita:**

*** / %**
+ -

Usando parênteses para evitar confusão com a precedência

Caso não seja muito fã ou bom em memorização, existe um recurso que você pode usar que vai deixar claro a ordem das operações, bem como deixar mais organizado seu código em C.
A regra é simples: agrupe e ponha entre parênteses o que você quer calcular.

Por exemplo, se ao invés de 1 + 1 x 2, tivéssemos escrito:
1 + (1 x 2) ?
É bem óbvio que somamos o 1 com o RESULTADO do que está entre parênteses. Ou seja, sempre é calculado primeiro o que está em parênteses:
1 + (1 x 2) = 1 + 2 = 3
(1 + 1) x 2 = 2 x 2 = 4

O outro caso 8/4 - 2, também fica bem evidente e simples de se entender apenas olhando:
(8/4) - 2 = 2 - 2 = 0
8/(4 - 2) = 8/2 = 4

E você achando que sabia as operações básicas de Matemática, hein?

*Atalhos com símbolos matemáticos em C: += , -=, *=, /= e %=*

Essa é uma das lições mais simples e rápida, tanto dessa seção sobre conhecimentos básicos da linguagem de programação C, como do curso C Progressivo inteiro.

Vamos aprender atalhos, maneiras mais rápidas e eficientes de escrever as operações matemáticas em C que você aprendeu no artigo passado de nossa **apostila online de C**.

Um dos artifícios mais usados pelos programadores C, que parece um pouco estranho na primeira vez que vemos, são umas abreviações usadas para descrever as operações matemáticas envolvendo uma mesma variável.

Vamos ver, em detalhes, cada uma dessas abreviações.

Fazendo contas com o valor antigo da variável

Suponhamos que declaramos uma variável inteira de nome ***teste*** e inicializamos ela com o valor 0:
int teste = 0;

Agora vamos fazer com que essa variável tenha valor 1:
teste = 1;

Agora vamos fazer com que seu valor seja aumentado em 2:
teste = 3;

Agora vamos fazer com que seu valor seja aumentado em 2, novamente:
teste = 5;

Simples, não? Até aqui, nada demais.
Mas tem uma coisa que você não deve ter notado: foi necessário que você, programador, fizesse as contas.

"Mas peraí! Um computador é uma máquina que computa, ou seja, ela serve para contar e eu é que tenho que fazer os cálculos?"

Se quisermos adicionar 1 vao valor inicial da variável, é melhor fazermos:
teste = teste + 1;

Ou seja, a variável teste vai receber um novo valor. Que valor é esse?
É o valor 'teste + 1', onde esse 'teste' é o valor anterior.

Ou seja, antes teste=0,
Depois de: teste = teste + 1 , a variável será 'teste=1'.
É como se tivéssemos feito: teste = 0 + 1 = 1

Agora vamos adicionar 2 unidades ao valor de 'teste':
teste = teste + 2;

Como 'teste', anteriormente, era 1, agora 'teste' recebe valor 3:
teste = teste + 2 = 1 + 2 = 3;

E como fazemos para adicionar 3 unidades ao valor de 'teste'?
Fácil: teste = teste + 3;

Ou seja, estamos adicionando unidades, sem se importa e sem se estressar com o valor anterior da variável.

O mesmo é para subtração, multiplicação, divisão e resto inteiro da divisão. Suponhamos que queiramos fazer uma operação com a variável 'x', podemos fazer:
x = x + 2;
x = x - 1;
x = x * 2;
x = x / 2;
x = x % 3;

Agora vamos para os atalhos!

Atalhos matemáticos: += , -= , *= , /= e %=

+=

Em vez de escrever:
x = x + 2;

Podemos escrever:
x += 2;

-=

Em vez de escrever:
x = x - 1;

Podemos escrever:
x -= 1;

*=

Em vez de escrever:
x = x * 2;

Podemos escrever:
x *= 2;

/=

Em vez de escrever:
x = x / 2;

Podemos escrever:
x /= 2;

%=

Em vez de escrever:
x = x % 2;

Podemos escrever:
x %= 2;

Como podem ver, são notações e atalhos simples, mas extremamente úteis e usadas em nosso curso online e gratuito C Progressivo.

Sistema binário e Valores lógicos true ou false

Nessa simples aula de nosso curso C Progressivo, vamos falar sobre o sistema de numeração binária e dos valores lógicos, ou booleanos, true ou false.

Este é um artigo muito importante de nossa **apostila de C**, pois o sistema binário é de suma importância no ramo da programação.

Computadores e o sistema binário

Se já teve algum contato, por menor que seja, com computação, já deve ter ouvido falar no sistema de numeração binária.
Ele é a base de toda a computação, e, obviamente, da programação também. Ou seja, tudo que envolve linguagens de programação e hardware, tem a ver com o sistema binário.

E podemos ir mais além, tudo na tecnologia moderna, como seu computador inteiro, tablet, celular, TV Digital e o que mais existir, é baseado no sistema binário.

No dia-a-dia usamos o sistema decimal, podemos representar qualquer usando os algarismos: 0, 1, 2, 3, 4, 5, 6, 7, 8, e 9.
Já os computadores armazenam todo tipo de informação apenas por combinações dos números 1 ou 0.

Porém, não se assuste, não vamos entrar a fundo na matemática, só é preciso que saiba que para os computadores tudo se resume a números, tudo é bit: fotos, vídeos, textos, programas etc.

Valores lógicos: true ou false, 1 ou 0

E onde o sistema binário entra na programação C?
Pois bem, diferentes de outras linguagens, como Java e C#, a linguagem de programação C é de baixo nível.
Isso quer dizer que vamos entrar mais a fundo nos nossos computadores. Com a linguagem C vamos ter acesso, por exemplo, aos espaços de memória de nosso sistema, por isso não se assuste quando falarmos em bits com certa frequência.

Como dito anteriormente, tudo se resume a 1 ou 0.
Porém, vamos dar outro significado a isso: de agora em diante o 0 representa "falso" e tudo diferente de 0 será, obviamente, "verdadeiro" (true).

Por exemplo: 1 + 1 = 2?
Resposta: verdadeiro. Como representamos algo verdadeiro em valores lógicos?
Por meio de qualquer número diferente de 0: 1, 2, 3, 1 milhão, -10 etc.

E 1 + 1 = 3?
Como representamos isso em valor lógico? Apenas de uma maneira: 0

Exemplo prático dos valores lógicos em computação

Com certeza você já deve ter usado algum programa que deu algum erro (principalmente se você usa o sistema operacional Windows).

Como bem deve se lembrar, deve ter visto algo do tipo: "Erro 144", "Error 221", "Problem: error 404", etc.

Pois bem, em programação quando um programa roda, ele geralmente termina dando um resultado, em forma de número.
O mais comum é que quando o programa termina sem mais problemas, ele retorne o número 0.

Caso o programa termine com algum erro, ele geralmente retorna outro valor diferente de 0.
Por exemplo, caso tenha faltado energia e o programa tenha fechado subitamente, ele resulta no número 1.
Caso o sistema trave, ele gera o número 2.
Caso você digite algo que não deveria, ele produz o número 3.

Pra que tudo isso? Comunicação entre programas.
Sabendo o número de retorno, outros programas ou o próprio sistema saberá exatamente o que ocorreu, pois cada número quer dizer que ocorreu algo diferente.

Usaremos muito, mas muito mesmo esses valores lógicos durante nosso curso de linguagem C.

Operadores Lógicos E (&&), OU (||) e de Negação (!)

Vimos no artigo passado de nossa apostila de C, uma explicação sobre valores lógicos true ou false em computação.

Vamos agora entrar mais em detalhes e ver como é possível representar informações apenas com true ou false, 1 ou 0.
E bem como o artigo passado, usaremos apenas ideias e conceitos, nada de código por hora.

Operador lógico 'E' em linguagem C: &&

Vamos supor que você passou em um concurso, o cargo é para ser programador e trabalhar para o governo.
No edital você leu: é necessário ser brasileiro e ser maior de 18 anos.

Ou seja, você só pode fazer esse concurso se for brasileiro E TAMBÉM se tiver de 18 anos ou mais.
Se for brasileiro mas tiver menos de 18 anos, não pode.
Se tiver mais de 18 anos mas não for brasileiro, também não pode.

Ou seja: todas as condições devem ser obedecidas para termos um resultado positivo (que no caso é poder fazer o concurso).

Vamos agora trazer para o nosso mundo da programação em linguagem C.
Esse E, vamos representar por &&

Então, para fazer o concurso: (ser brasileiro) && (ter 18 anos ou mais)
Vamos supor que você é brasileiro. Vamos representar esse fato como '1', pois é verdade.
Vamos supor que você tem 18 anos, representaremos isso pelo valor lógico '1' também.

Então: 1 && 1 resulta em verdade, ou 1.

E se você for menor?

Nossa expressão fica: 1 && 0, que resulta em resultado negativo, ou 0. Então você não pode fazer o concurso.

Operador lógico 'OU' em linguagem C: ||

Vamos supor que você quer trabalhar em outro país, no Canadá por exemplo.
Para isso ser possível você deve ser canadense OU ser casado com uma canadense.

Se você for canadense, mas não for casado com uma, pode trabalhar? Sim, pode.
Se você não for canadense, mas for casado com uma canadense, pode trabalhar lá? Sim, pode.

Pode porque para isso ser possível (para ser verdade), é necessário ser canadense OU ser casado com uma.
Preenchendo apenas um dos requisitos, você está apto e o resultado é positivo.
E se você for canadense e casado com uma canadense? Ora, é óbvio que pode também.

Esse OU será representado em programação pelo símbolo: ||

Então: "1 || 0" resulta em valor lógico verdadeiro (true, ou 1)
"0 || 1" também, assim como "1 || 1" também vai resultar em valor lógico verdadeiro.

Assim, para expressões que possuem a condição OU (||), se uma das condições for verdadeira, toda a sentença será.
E para a expressão ser falsa, todas as condições devem ser falsas.

Bem óbvio, não?

Operador lógico de negação em linguagem C: !

Agora está bem claro pra você que as condições e expressões ou são verdadeiras (true ou 1) ou são falsas (false ou 0).

Quando queremos negar algo, colocamos o símbolo ! antes do que queremos negar.
Por exemplo, 1 + 1 é dois?
A resposta é sim, ou '1'. Ou '!0', pois o contrário de 0 é 1.

2 + 2 é três? Falso ou 0. Ou '!1', pois o contrário de 1 é 0

Assim como a lição passada, essa é bem simples, não?

Teste Condicional e Controle de Fluxo

O teste condicional IF ELSE

Se notar bem, em nosso curso online de programação C, todos os nossos programas são rodados uma vez, do início até o fim do código, fazendo sempre a mesma coisa.

Mas você sabe que os programas 'reais' não são assim...se clicar em um botão acontecesse uma coisa, se selecionar algo no menu, outra coisa acontece.

Ou seja, os programas rodam dependendo de como o usuário interage com o aplicativo.
E é isso que vamos estudar nesse tutorial: teste condicional.

O que é, para que serve e como usar o teste condicional IF em C

Sem mais delongas, a sintaxe do teste condicional IF é a seguinte:

```
if( condição )
{
  // código
  // do IF
  // aqui
}
```

O 'if', em inglês, quer dizer 'se', no sentido de condição.
Por exemplo: "If you like C, you will like C++ too" (Se você gosta de C, vai gostar de C++ também).

O que o 'if' faz é testar a 'condição' entre parênteses, e se ela for verdadeira será executado o código entre chaves.

Exemplo de código: Condição sempre verdadeira

```
#include <stdio.h>

int main()
{
    if(1)
    {
        printf("Esse teste condicional é verdadeiro");
    }
}
```

Se você rodar esse programa em C verá que o print sempre aparecerá na tela, pois a condição 1 representa sempre verdadeiro, como você viu no artigo Sistema Binário e Valores lógicos 'true' ou 'false' em http://cprogressivo.blogspot.com.br/2013/01/Sistema-binario-e-Valores-Logicos-true-ou-false.html

Na verdade, qualquer valor diferente de 0 sempre resultará que o teste condicional é verdadeiro.

Esse 'verdadeiro' ou 'falso' é uma espécie de 'retorno' que a condição nos dá.
Por exemplo, vamos testar um código em que o teste condicional nunca é executado:

Exemplo de condição: Condição sempre falsa

```
#include <stdio.h>
int main()
{
    if( 0 )
    {
        printf("Somente programadores verão essa mensagem");
    }

    printf("Hello, World");
}
```

Se você rodar, aparecerá na tela somente o "Hello, World" pois o 0 sempre retorna valor 'false' para o teste condicional IF. Como o resultado é falso, o que está dentro do IF não é executado.

O que é, para que serve e como usar o ELSE em C

A palavra '*else*' em inglês quer dizer 'ou', e ela está totalmente ligada ao IF, que já estudamos.

Se IF significa '*se'* e o ELSE significa '*outro*', você pode pensar e concluir para que serve o ELSE?

O ELSE só existe na presença do IF e só aparece depois que o IF aparecer.
A característica principal do ELSE é que ele não recebe nenhuma condição (ou seja, não tem parênteses depois do ELSE, como tem no IF).

Na verdade ele recebe uma condição, que é implícita. A condição dele é contrária do IF.
Como uma condição é sempre TRUE ou FALSE, no par IF e ELSE um deles vai rodar se a condição for TRUE e o outro só roda se a condição FALSE.

A sintaxe dessa importantíssima dupla é a seguinte:

```
if ( condição )
{
  //se a condição
  //for verdadeira
  //esse código rodará
}
else
{
  //caso a condição
  //seja FALSA
  //é esse código que rodará
}
```

Ou seja, se a condição retornar um valor lógico VERDADEIRO, o IF rodará.
Porém, caso a condição retorne um valor lógico FALSO, o ELSE que rodará.

Então, veja o código a seguir, raciocine e tente adivinhar qual print irá rodar, antes de executar

o programa:

Exemplo de código: rodando o ELSE

```
#include <stdio.h>

int main()
{
    if( 0 )
    {
        printf("Somente programadores verão essa mensagem");
    }
    else
    {
        printf("Hello, world!");
    }
}
```

Ora, 0 retorna valor lógico falso, então o IF não será executado.
Sempre que o IF não é executado, o ELSE será. Portanto, veremos a mensagem "Hello, World" na tela.

Exercício de C:
Faça o contrário do último exercício.
Ou seja, que o print "Hello, world!" seja exibido na tela, dentro do bloco do IF e que o bloco do ELSE (com a frase "Somente programadores verão essa mensagem") nunca seja rodado.

Fazendo testes e comparações - operador de igualdade (==), maior (>), menor (<), maior igual (>=), menor igual (<=), de diferença (!=) e de módulo, ou resto da divisão (%)

No artigo passado, ensinamos sobre o uso do teste condicional IF e ELSE, que será extremamente usado por nós em nosso curso, e por você em sua carreira de programador C.

Porém, fizemos testes de condições simples e sem muita utilidade. Vamos aprender agora, em nossa **apostila de C**, a usar alguns operadores que nos permitirão fazer coisas bem interessantes apenas com IF ELSE.

Testar e comparar, testar e comparar...a essência da programação

A primeira vista não damos tanta importância e nem entendemos porque temos tanto que testar e comparar coisas em programação. Pois bem, o curso C Progressivo explica isso melhor pra você, com exemplos reais.

Quando você entra em um site de conteúdo adulto e te perguntam sua idade: se você clicar em 'tenho mais de 18 anos', você entra, se clicar em 'tenho menos de 18 anos', eles te encaminham para o site do Restart.

Pois bem, eles fizeram um **teste**: se fizer isso, acontece isso, se fizer aquilo acontece outra coisa.

Às vezes é perguntada nossa data de nascimento em alguns sites. Informamos os números, eles guardam em alguma variável e depois comparam: se o número for menor que uma data específica, você é de maior. Caso os números que você forneceu sejam maiores que essa data, é porque você ainda é de menor.

Isso foi uma comparação: pegamos um dado (sua data de nascimento) e comparamos com outro (a data de 18 anos atrás).

Durante um jogo, o programa fica o tempo inteiro testando o que o jogador digitou. Se digitou a tecla pra direita, vai pra direita, se foi pra cima, vai pra cima, se não apertar nada, nada ocorre e assim a vida segue.

Se convenceu da importância dos testes e comparações? Então vamos testar e comparar.

O operador de igualdade em C: ==

Se quisermos comparar uma variável ***x*** com uma variável ***y*** fazemos:

x == y

Sempre que ver isso, leia como se fosse uma pergunta: *x é igual a y?*

Ou seja, se é uma pergunta (comparação), **sempre** retorna um valor booleano. Ou seja, ou é falso ou é verdadeiro.

É comum que alguns estudantes e iniciantes em programação confundam o '=' com o '=='.
O '=' é uma atribuição, estamos atribuindo um valor, estamos dizendo que aquela variável vai receber um valor, e ponto final.

Já o '==' é um teste, queremos saber se o valor é igual ou não.
Ou seja, ele vai retornar TRUE ou FALSE. É como se estivéssemos fazendo uma pergunta, e que estamos esperando uma resposta.

Exemplo de código:
Faça um programa em C que pergunte ao usuário quanto é 1 + 1, usando o teste condicional **IF ELSE**, e que responda se ele acertou ou errou.

```c
#include <stdio.h>

int main(void)
{
    int resultado;

    printf("Quanto eh 1 + 1? ");
    scanf("%d",&resultado);
```

```
    if(resultado == 2)
    {
        printf("Parabens, voce acertou\n");
    }
    else
    {
        printf("Amigo, estude mais matemática...\n");
    }
}
```

Ou seja, nós perguntamos o valor de 1+1 e armazenamos na variável inteira ***resultado***.
Depois comparamos esse resultado com 2.

Se o usuário tiver digitado 2, a comparação ***resultado == 2*** retornará valor booleano VERDADEIRO (true) e o ***IF*** ocorre.
Caso o que o usuário digitou não seja 2, a comparação retornará valor booleano FALSO (false) e será o ***ELSE*** que o nosso programa em C irá rodar.

Simples, não? Continuemos...

O operador *maior que* em C: >

Se quisermos saber se uma variável ***x*** é maior que uma variável ***y*** fazemos:

x > y

Caso *x* seja maior que *y*, essa comparação retornará valor lógico VERDADEIRO (true).
Caso *x* seja menor, ou igual, a *y*, essa comparação retornará valor lógico FALSO (false).

Exemplo de código:
Crie um aplicativo em C que pergunte a idade do usuário e diga se ele é maior ou menor de idade, usando o comparador *maior que*.

```
#include <stdio.h>

int main(void)
{
    int idade;

    printf("Digite sua idade: ");
    scanf("%d",&idade);

    if(idade > 17)
    {
        printf("Voce eh de maior, pode entrar! \n");
    }
    else
    {
        printf("Sinto muito, voce nao pode entrar\n");
    }
}
```

A explicação do código C é bem simples: para ser de maior, você deve ter mais que 17 anos, se tiver, a comparação *idade* > *17* retornará TRUE e o ***IF*** irá rodar.
Caso você tenha 17 anos ou menos, você é de menor e o código salta para rodar o ***ELSE***.

Pronto, agora você já pode criar aqueles sistemas de sites que perguntam se você é de maior ou não.
Muito útil essa linguagem de programação C, não?

O operador *menor que* em C: <

Se quisermos saber se uma variável ***x*** é menor que uma variável ***y*** fazemos:

x < y

Caso *x* seja menor que *y*, essa comparação retornará valor lógico VERDADEIRO (true).
Caso *x* seja maior, ou igual, a *y*, essa comparação retornará valor lógico FALSO (false).

Exemplo de código:
Crie um aplicativo em C que pergunte a idade do usuário e diga se ele é maior ou menor de idade, usando o comparador *menor que*.

```c
#include <stdio.h>

int main(void)
{
    int idade;

    printf("Digite sua idade: ");
    scanf("%d",&idade);

    if(idade < 18)
    {
        printf("Sinto muito, voce nao pode entrar\n");
    }
    else
    {
        printf("Voce eh de maior, pode entrar!\n");
    }
}
```

A explicação do código C é o oposto da explicação do exemplo anterior, note a lógica inversa: se você tiver menos que 18 anos, você será menor e o teste ***idade < 18*** retornará valor lógico VERDADEIRO e o ***IF*** será executado.
Se você não tem menos que 18 anos, é porque tem mais ora. Então, nosso programa em C executará o ***ELSE***.

O operador *maior igual a* em C: >=

Até o momento fizemos somente a análise do tipo 'maior ou menor'. Porém, às vezes queremos incluir o próprio número na comparação...ou seja, queremos saber se uma variável é maior ou igual que outra.

Se quisermos saber se uma variável ***x*** é maior ou igual que uma variável ***y*** fazemos:

x >= y

Caso *x* seja maior ou igual que *y*, essa comparação retornará valor lógico VERDADEIRO (true).
Caso *x* seja menor a *y*, essa comparação retornará valor lógico FALSO (false).

Exemplo de código:
Crie um aplicativo em C que pergunte a idade do usuário e diga se ele tem que fazer ou não o exame de próstata, usando o operador *maior igual a*. Lembrando que é recomendável fazer o exame a partir dos 45 anos.

```
#include <stdio.h>

int main(void)
{
    int idade;

    printf("Digite sua idade: ");
    scanf("%d",&idade);

    if(idade >= 45)
    {
        printf("Eh amigo, ja era, tem que fazer o exame da
prostata...\n");
    }
    else
    {
        printf("Voce nao precisa fazer o exame de
prostata...ainda\n");
    }
}
```

Note quem tem mais de 45 anos é necessário fazer o exame. Porém, quem também TEM, exatos, 45 anos, também precisa fazer!
Também poderíamos ter feito: ***idade > 44***
Porém não fica muito legal o número 44 aparecer. Por quê?
Ora, a informação que temos da idade é 45, então vamos trabalhar com 45. É uma questão de organização apenas, mas que pode facilitar muito a vida de quem lê um código em C.

Por exemplo, suponha que você foi contratado por uma empresa para ser programador C e vai substituir outro programador (que não sabia muito a linguagem C, por isso perdeu o emprego). Se você bater o olho no código e ver o número **18** logo você associa com maioridade.
Porém, se tiver o número **17** no meio do código fica difícil deduzir o que o antigo programador tentou fazer...aí você vai ter que quebrar a cabeça pra entender o código dele.

Mas como você está cursando o C Progressivo, você será treinado para ser um programador organizado também. Vamos continuar...

O operador *menor igual a* em C: <=

Se quisermos saber se uma variável ***x*** é menor ou igual que uma variável ***y*** fazemos:

x <= y

Caso *x* seja menor ou igual que *y*, essa comparação retornará valor lógico VERDADEIRO (true).
Caso *x* seja maior a *y*, essa comparação retornará valor lógico FALSO (false).

Exemplo de código:
Crie um aplicativo em C que pergunte a idade do usuário e diga se ele tem que fazer ou não o exame de próstata, usando o operador *menor igual a*. Lembrando que é recomendável fazer o

exame a partir dos 45 anos.

```
#include <stdio.h>

int main(void)
{
    int idade;

    printf("Digite sua idade: ");
    scanf("%d",&idade);

    if(idade <= 44)
    {
        printf("Voce nao precisa fazer o exame de
prostata...ainda\n");
    }
    else
    {
        printf("Eh amigo, ja era, tem que fazer o exame da
prostata...\n");
    }
}
```

Agora fizemos o contrário. Comparamos usando a expressão ***idade <= 44*** para verificar que o usuário não tem 45 anos ou mais.

Analisando esse código e o do exemplo passado, qual você usaria para criar tal programa?
O anterior, é claro, pois ele lida com o número 45.

O operador *diferente de* em C: !=

Se quisermos saber se uma variável ***x*** é diferente de uma variável ***y*** fazemos:

x != y

Caso *x* seja diferente de *y*, esse teste retorna valor VERDADEIRO.
Caso *x* seja igual a *y*, esse teste retorna valor FALSO.

Exemplo de código:
Crie um programa em C que pergunte um número de 1 até 10 ao usuário, e faça com que ele tente acertar o número secreto. No meu caso, o número secreto é 7.

```
#include <stdio.h>

int main(void)
{
    int numero;

    printf("Adivinha a senha, entre 1 e 10: ");
    scanf("%d",&numero);

    if(numero != 7)
    {
```

```
        printf("Errado! Saia da Matrix!\n");
    }
    else
    {
        printf("Parabens, voce entrou no sistema\n");
    }
}
```

A explicação é simples: enquanto o usuário digitar um número diferente de 7, o programa entra no ***IF*** e diz que o usuário errou a senha.
Se ele não digitar um número de 7, é porque ele digitou o 7, ora! Então, a comparação ***numero != 7*** retorna falso e o ***ELSE*** que será executado.

O operador *módulo ou resto da divisão* em C: %

Resto da divisão é o valor que sobra/resta da divisão para que o quociente permaneça inteiro.

O símbolo de resto da divisão é %.

Veja, o que está em vermelho é o resto inteiro da divisão:
5 % 2 = 2*2 + 1
7 % 3 = 3*2 + 1
7 % 4 = 1*4 + 3

8 % 3 = 2. Pois: 8 = 3*2 + 2

Em matemática chamamos de 'mod', de módulo: 5 mod 3 = 2

No Brasil, é aquele 'resto' que deixávamos lá embaixo nas continhas do colégio:

Sem dúvidas, a maior utilidade desse operador é saber a multiplicidade de um número (ou seja, se é múltiplo de 2, 3, 5, 7, etc).
Por exemplo, um número par é aquele que, quando dividido por 2, deixa resto nulo.

Exemplo de código:
Crie um programa em C que pergunte um número ao usuário e diga se ele é par. Use o operador módulo ou resto da divisão ***%***.

```c
#include <stdio.h>

int main(void)
{
    int numero;

    printf("Digite um inteiro: ");
    scanf("%d",&numero);

    if(numero % 2 == 0)
    {
        printf("Eh par\n");
    }
    else
    {
        printf("Nao eh par\n");
    }
}
```

Em programação, para saber se um número é par, escrevemos: ***numero % 2 == 0***
Leia: se o resto da divisão de 'numero' por 2 deixar resto 0, retorne valor TRUE.

Erro comum de iniciantes em programação C

Praticamente todos os programadores, quando estão iniciando seus estudos, cometem o erro de confundir o operador de ***atribuição*** com o operador de ***comparação de igualdade***.

Por exemplo, qual a diferença entre:

x = 1

x == 1

?

A resposta é simples: no primeiro exemplo estamos atribuindo, no segundo estamos comparando.

Leia assim:

x = 1: 'Hey C, faça com que a variável x receba o valor 1, atribua 1 ao x'

x == 1: "Hey C, o valor armazenado na variável x é 1'?

Ou seja, na comparação há o retorno de valor lógico VERDADEIRO ou FALSO.

Na atribuição não, simplesmente atribuímos um valor e essa operação retorna sempre true.

Exercícios sobre operadores de comparação em C:

1. Faça o programa do exemplo do ***!=*** , mas com o operador **==**
2. Faça o programa do exame de próstata usando o operador **<**

3. Faça o programa do exame de próstata usando o operador **>**
4. Faça o programa do teste de maioridade usando o operador **<=**
5. Faça o programa do teste de maioridade usando o operador >**=**
6. Faça um programa que pergunte um número ao usuário e diga se ele é ímpar. Use o operador **%**.

Questões sobre IF e ELSE

Agora que já ensinamos o que é e como usar o teste condicional IF-ELSE e operadores relacionais em linguagem C(para fazer testes e comparações), agora vamos propor alguns exercícios, em nossa apostila de C, e mostrar suas soluções, com códigos comentados, para você fixar seus conhecimentos nesses conceitos tão importantes em C.

Lembramos que esses exercícios fazem parte do curso, além de ensinarmos técnicas de programação e assuntos importantes através deles.

Exercícios sobre IF e ELSE em C

Embora venhamos a mostrar a solução dos exercícios, **não olhe a solução antes de tentar**. É simples: você só aprende se tentar. Se estiver só lendo, copiando e colando os códigos no Code::Blocks, você nunca aprenderá. Tente, tente de novo, depois tente mais um pouco. Só então olhe a solução.

0. Faça um programa que peça dois números ao usuário e mostre qual o maior e qual o menor

1. Faça um programa que receba três inteiros e diga qual deles é o maior e qual o menor. Consegue criar mais de uma solução?

2. Escreva um programa em C que recebe um inteiro e diga se é par ou ímpar.
Use o operador matemático % (resto da divisão ou módulo) e o teste condicional if.

3. Escreva um programa que pergunte o raio de uma circunferência, e sem seguida mostre o diâmetro, comprimento e área da circunferência.

4. Para doar sangue é necessário ter entre 18 e 67 anos. Faça um aplicativo na linguagem C que pergunte a idade de uma pessoa e diga se ela pode doar sangue ou não. Use alguns dos operadores lógicos OU (||) e E (&&).

5. Escreva um programa que pergunte o dia, mês e ano do aniversário de uma pessoa e diga se a data é válida ou não. Caso não seja, diga o motivo. Suponha que todos os meses tem 31 dias e que estejamos no ano de 2013.

Desafio 1. Crie um programa em C que peça um número ao usuário e armazene ele na variável *x*. Depois peça outro número e armazene na variável *y*.
Mostre esses números. Em seguida, faça com que *x* passe a ter o valor de *y*, e que *y* passe a ter o valor de *x*.
Dica: você vai precisar usar outra variável.

Clique aqui para conferir as soluções, com códigos comentados, dessas questões sobre o teste

condicional IF ELSE em Linguagem C.

Desafio 2. Escreva um programa que pede os coeficientes de uma equação do segundo grau e exibe as raízes da equação, sejam elas reais ou complexas.
Solução do desafio 2

Desafio 3. Crie um programa em C que recebe uma nota (entre 0.0 e 10.0) e checa se você passou direto, ficou de recuperação ou foi reprovado na matéria.
A regra é a seguinte:
Nota 7 ou mais: passou direto
Entre 4 e 7: tem direito de fazer uma prova de recuperação
Abaixo de 4: reprovado direto
Solução do desafio 3

Operadores de Incremento (++) e Decremento (--) - Diferença entre a=b++ e a=++b

Antes de iniciarmos nossos estudos sobre os laços, o curso C Progressivo irá apresentar algumas ferramentas que serão úteis e bastante utilizadas.

São operadores matemáticos que facilitarão nossas vidas de programador C, e que iremos usar diversas vezes em nossa apostila online.

Contando em linguagem C

Em muitas aplicações C precisamos contar, ou controlar coisas:

- quantas teclas foram digitadas
- quantas vidas você ainda tem em jogo
- quanto tempo se passou
- quantos visitantes o site C Progressivo recebeu
- etc.

Até agora, temos contado de um em um. Imagine, gerenciar e contar um sistema de cadastro com milhões de inscritos, como o do Enem?

Ainda bem que você é programador C e não se assustou, pois sabe que não vai fazer isso, e sim seu escravo: o computador.
Lembre-se: a relação do programador C com a máquina não é de usuário, e sim de mestre para escravo. Com C, você domina tudo o que ocorre com seu computador.

Então, boa parte das contas são feitas utilizando os 'laços', que ensinaremos bem em breve e vai utilizar para sempre.
Com ele, você vai fazer com que o computador faça o tanto de contas que quiser...10, 20, mil, 1 milhão...o Google realiza trilhões por dia.
Você pode até fazer um looping infinito e travar seu computador. Uns chamam de hackear, eu prefiro chamar de estudo e vou mostrar como fazer.

Esses laços, porém, tem que ser incrementado. Geralmente eles têm um início, uma condição e um fim.
Como esse fim é atingido? Através de uma ou mais variável que vão crescendo ou diminuindo, ou através de expressões lógicas.
Quando se atinge um certo ponto, o laço vai parar.

É aí que entra os operadores de incremento e decremento.

Somando e Subtraindo variáveis em C

O incremento em uma unidade quer dizer:
a = a + 1

a = a + 1, em C, é algo totalmente diferente da Matemática convencional.
Vamos ver na prática, que depois explico.

Façamos isso em um aplicativo em C:
a=1;
a = a + 1;
Logo após atribuir 1 ao 'a', imprima, veja o resultado e tente descobrir o que aconteceu.
Substitua por: a = a + 2;
Ou a = a + 3;

```c
#include <stdio.h>

int main()
{
        int a=1;

        printf("a = %d\n", a);
        a = a + 1;
        printf("a = %d\n", a);

}
```

Hackers são pessoas que descobrem as coisas sozinhas (e não criminosos ou vândalos, como é difundido), fazem isso.
Testam, pensam e descobrem como as coisas funcionam.

Explicação:
Quando fazemos a =
é porque vamos atribuir um valor ao 'a', um novo valor, INDEPENDENTE do que ele tinha antes.
E qual o valor que vamos atribuir? 'a + 1'!
Ou seja, o 'a' vai ser agora o seu antigo valor mais 1!

Se a=2, e fazemos a = a + 3, o que estamos fazendo?
Estamos dizendo que o 'a' vai mudar, vai ser seu antigo valor (2), mais 3! Ou seja, agora a=5
Faz sentido, não?

a++ e a--
Usaremos muito, mas MUITO mesmo o incremento e o decremento de unidade:
a = a + 1;
a = a - 1;

Porém, é chato ou dá preguiça ficar escrevendo a=a+1 e a=a-1 o tempo inteiro.
Então, inventaram atalhos para isso!
a = a + 1 pode ser representado por a++ ou ++a
a = a - 1 pode ser representado por a-- ou --a

Existe uma diferença entre a-- e --a, ou entre a++ e ++a, que vamos explicar no próximo tópico.
E como já dizia o filósofo: só creio compilando e vendo.

```c
#include <stdio.h>

int main()
{       int a=1,
            b=1;

        printf("Valor inicial de a = %d\n", a);
        printf("Valor inicial de b = %d\n", b);

        printf("\nIncrementando: a++\n");
        a++;

        printf("Decrementando: b--\n");
        b--;

        printf("\nAgora a = %d\n", a);
        printf("Agora b = %d\n",b);

}
```

Diferença entre os operadores entre a=++b e a=b++

Já vimos o uso dos operadores de decremento e incremento, em C.

Você já estudou e testou que usar isoladamente:
a++ e ++a não surte efeito.

Porém, surte na hora da atribuição.

Vamos lá, ao nosso 'teste hacker':
Seja b=2;
Só olhando e raciocinando, você seria capaz de descobrir quanto seria 'a' em cada caso:
a = b++;
a = ++b;

Explicações
1. a = b++

Isso é apenas um atalho em C, uma forma mais simples de escrever as seguintes linhas:
a = b;
b++;

2. a = ++b
Analogamente, é o atalho que usamos em C para representar as seguintes linhas de código:
++b;
a = b;

Em ambos os casos, 'b' é incrementado. A diferença é que 'a' recebe o valor de 'b' antes do incremento em um e depois do incremento em outro. Não precisa decorar nada, é só analisar a lógica das coisas e verá que tudo faz sentido na linguagem C.
Veja:

No primeiro exemplo: a = b++
O que vem primeiro, 'b' ou incremento? Primeiro o 'b'. Depois o de incremento ocorre, '++'.
Ou seja, 'a' vai receber o valor de 'b' primeiro, então a=2
Só depois que 'b' vai ser incrementado e vai se tornar b=3.

No segundo exemplo: a = ++b

O que vem primeiro? 'b' ou incremento?
Primeiro o incremento, '++', depois que aparece o 'b', então só depois acontece a atribuição..
Assim, primeiro ocorre o incremente, então b=3.
Só depois é que esse valor é atribuído para a.

Como diria o velho ditado: compilais e verás a verdade.
Eis o código:

```c
#include <stdio.h>

int main()
{
        int a,
            b=1;

        printf("b = %d\n",b);
        printf("a = b++\n");

        a = b++;

        printf("\nAgora: \na = %d\n",a);

        printf("b = %d\n",b);
        printf("\na = ++b\n");

        a = ++b;

        printf("\nAgora: a = %d",a);
}
```

O que é e como usar o laço WHILE em C

Com o teste condicional IF ELSE, explicado aqui na **apostila C Progressivo**, você passou a ter mais controle sobre suas aplicações em C, pois agora você pode escolher o que será rodado em seu programa, dependendo dos valores lógicos que esse teste condicional faz.

Vamos agora introduzir outro importante assunto em linguagem de programação C: laços. Também conhecidos por loopings.

Com laços e testes, você passará a ter total controle do fluxo em seus códigos.

Repetir e repetir: o que são e para que servem os laços em C

Repetir é uma das coisas mais importantes na programação. Na verdade, elas não se repetem indefinidamente, elas se repetem dependendo dos testes que fazem.

Por exemplo, seu computador faz várias repetições para saber se você está conectado à internet.
Se esse teste der resultado positivo, nada acontece.
Porém, se ele testar e ver que a conexão não existe mais, ele manda uma mensagem dizendo que você perdeu a conexão.

Fazer testes você já está careca de saber.

Porém, como faria todos esses testes? Colocaria um if else a cada segundo, durante 1h?
E se o usuário passar 2h conectado? Ou 1 dia?

E se você fosse contratado para criar um programa em C que diz se cada aluno em uma sala de aula passou.
Se essa sala tem 30 alunos, você vai criar 30 testes condicionais?
E se a sala tiver 60 alunos? Ou se você for contratado para testar as notas de todos os 2mil alunos de uma escola?
2mil blocos de IF ELSE?

Óbvio que não. Lembre-se: você programa em C, então você é quem manda no seu computador.
Você fará sua máquina repetir essas coisas automaticamente.

O laço while: definição e sintaxe

While, em inglês, significa 'enquanto'.
A função do laço while é repetir um determinado trecho ENQUANTO uma determinada condição for verdadeira.

Por exemplo, enquanto (while) o personagem tiver *life*, o jogo deve continuar.
Enquanto (while) a mp3 não terminar, ela deve continuar tocando.
Enquanto (while) um jogador não ganhar, perder ou der empate, o jogo da velha deve continuar rodando.
Enquanto o usuário não digitar 0, seu programa deve continuar rodando.

E poderíamos dar milhões de exemplos do quanto os laços, como o *while*, são importantes e essenciais em linguagem C.

Vamos, então, aprender a usá-lo.
A sintaxe é a seguinte:

```
while( condição )
{
    //código que
    //será repetido
}
```

Ou seja, o código do bloco só será rodado enquanto a condição der resultado verdadeiro.
Primeiro o while testa a condição, se for verdadeira, ele roda.
Ao término, ele testa novamente e roda novamente se for verdadeira.
Caso, em algum momento, o teste forneça valor lógico FALSO, o while vai parar e seu programa segue o fluxo normalmente.

Essas 'rodadas' de 'testa e roda' do while, e de outros laços, são chamadas de iterações.

E se a condição for sempre verdadeira? Como em:

```
while( 1 )
{
    printf("O que acontece?\n");
}
```

Acontece um looping infinito. Esse printf, ou qualquer código naquele lugar, rodaria indefinidamente...só pararia quando você fechasse o aplicativo ou reiniciasse sua máquina.

Na grande maioria dos códigos em C, porém, essa condição vai mudando.
Geralmente é um número que cresce ou decresce, e ao atingir certo ponto, o while recebe FALSE como resultado do teste da condição, e o while termina.

Vamos ver alguns exemplos.

Exemplo comentado de código:

Programa em C que conta de 1 até 10 usando o laço while

Primeiro declaramos uma variável, e inicializamos ela com o valor inicial, que é 1.
Logo em seguida ,fazemos com que o while dê um printf, mostrando o valor dessa variável ENQUANTO ela for MENOR ou IGUAL a 10.
Após imprimir o número 10, a variável é incrementada e passa a ter valor 11.
Como 11 não é menor ou igual a 10, o laço recebe valor lógico FALSE e termina sua execução.

```c
#include <stdio.h>

int main(void)
{
        int numero=1;

        while(numero <= 10)
        {
            printf("%d\n", numero);
            numero++;
        }

}
```

Outros testes que resultariam positivo para esse exemplo:
while(numero < 19);
while(numero != 11);

Exemplo: Programa em C que conta de 10 até 1, usando o laço while
Agora vamos fazer o contrário do exemplo passado.
A lógico é a mesma: veja, raciocine e tente entender sozinho:

```c
#include <stdio.h

int main(void)
{
        int numero=10;

        while(numero >= 1)
        {
            printf("%d\n", numero);
            numero--;
        }

}
```

Exemplo comentado de código:

Crie um aplicativo em C que peça ao usuário o primeiro elemento de uma P.A, a razão da P.A e o número de elementos a serem exibidos.

O termo inicial P.A será armazenado na variável **init**.
A razão da P.A será armazenado na variável **rate**, e o número de elementos da P.A será armazenado na variável **term**.

Vamos usar duas variáveis de apoio, para contar.
Uma delas é a variável num, que representará o valor de cada termo da P.A. O valor dela será mudado a cada iteração, percorrendo todos os valores da P.A.
Já a variável **count** será usada para contar os termos da P.A. Os termos começam no primeiro (1) e terminar no último (term).

A fórmula para saber o valor do elemento 'an' de uma P.A é:
an = a1 + (n - 1) * razão

Assim, nosso código em C fica:

```c
#include <stdio.h>

int main(void)
{
        int init,
            rate,
            term;

        int num,
            count=1;

        printf("Digite o numero inicial da P.A: ");
        scanf("%d", &init);

        printf("Digite a razao da P.A: ");
        scanf("%d", &rate);

        printf("Digite o numero de termos da P.A: ");
        scanf("%d", &term);

        while(count <= term)
        {
            num = init + (count - 1) * rate;
            printf("Termo %d: %d\n",count, num);
            count++;
        }

}
```

Exemplo comentado de código:

Crie um aplicativo em C que peça ao usuário o primeiro elemento de uma P.G, a razão da P.G e o número de elementos a serem exibidos.
O raciocínio é análogo ao do exemplo anterior. Mas o termo 'an' de uma P.G é diferente da P.A, e é dado por:
an = a1 * (razao)^(n-1)

A única diferença aqui é que vamos usar a função *pow* da biblioteca *math.h.*
O uso dela é bem simples: *pow(a,b)* significa 'a elevado a b'.

Então, nosso código em C fica:

```c
#include <stdio.h>
#include <math.h>

int main(void)
{
        int init,
            rate,
            term;

        int num,
            count=1;

        printf("Digite o numero inicial da P.G: ");
        scanf("%d", &init);

        printf("Digite a razao da P.G: ");
        scanf("%d", &rate);

        printf("Digite o numero de termos da P.G: ");
        scanf("%d", &term);

        while(count <= term)
        {
            num = init*pow(rate,(count-1));
            printf("Termo %d: %d\n",count, num);
            count++;
        }
}
```

Questões sobre o laço WHILE em C

Na aula passada, em nosso curso online C Progressivo, aprendemos sobre o tão famoso e importante laço WHILE, da linguagem C.

Vamos agora propor algumas questões para você treinar e fixar seus conhecimentos.
Tente usar somente o laço while, e nenhum outro.

Não vá olhar a solução nem o código antes de tentar, bastante, resolver por conta própria. Do contrário, nunca aprenderá C.

Apostila C Progressivo - Exercícios sobre o laço WHILE

0. Programa em C dos patinhos da Xuxa
Xuxa, a rainha dos baixinhos, criou uma música que tem o segunte formato:

n patinhos foram passear
Além das montanhas
Para brincar
A mamãe gritou: Quá, quá, quá, quá
Mas só n-1 patinhos voltaram de lá.

Que se repete até nenhum patinho voltar de lá.
Ao final, todos os patinhos voltam:

A mamãe patinha foi procurar
Além das montanhas
Na beira do mar
A mamãe gritou: Quá, quá, quá, quá
E os n patinhos voltaram de lá.

Crie um programa em C que recebe um inteiro positivo do usuário e exibe a música inteira na tela, onde o inteiro recebido representa o número inicial n de patinhos que foram passear.

1. Programa em C que mostra os números ímpares
Escreva um aplicativo em C mostra todos os números ímpares de 1 até 100.

2. Programa em C que mostra os números pares
Escreva um aplicativo em C mostra todos os números pares de 1 até 100.

3. Programa em C que mostra os números pares e ímpares
Escreva um aplicativo em C que recebe inteiro e mostra os números pares e ímpares (separados), de 1 até esse inteiro.

4. Programa em C que calcula a média das notas de uma turma
Escreva um programa que pergunte ao usuário quantos alunos tem na sala dele.
Em seguida, através de um laço while, pede ao usuário para que entre com as notas de todos os alunos da sala, um por vez.

Por fim, o programa mostra a média, aritmética, da turma.

5. Achando o maior número
Achar o maior, menor, média e organizar números ou sequências são os algoritmos mais importantes e estudados em Computação. Em C não poderia ser diferente.
Em nosso curso, obviamente, também não será diferente.

Escreva um programa em C que solicita 10 números ao usuário, através de um laço while, e ao final
mostre qual destes números é o maior.

6. Achando os dois maiores números
Escreva um programa em C que solicita 10 números ao usuário, através de um laço while, e ao

final
mostre os dois maiores números digitados pelo usuário.

7. Quadrado de asteriscos
Escreva um programa que lê o tamanho do lado de um quadrado e imprime um quadrado daquele tamanho com asteriscos. Seu programa deve funcionar para quadrados com lados de todos os tamanhos entre 1 e 20.
Para lado igual a 5:
*****
*****
*****
*****
*****

8. Quadrado de asteriscos e espaços em branco
Escreva um programa que lê o tamanho do lado de um quadrado e imprime um quadrado daquele tamanho com asteriscos e espaços em branco. Seu programa deve funcionar para quadrados com lados de todos os tamanhos entre 1 e 20.
Para lado igual a 5:
*****
* *
* *
* *
*****

9. Tabuada em C
Escreva um programa que pergunta um número ao usuário, e mostra sua tabuada completa (de 1 até 10).

10. Tabela ASCII em C
Se você lembrar bem, quando estudamos as variáveis do tipo caractere, *char*, dissemos que, na verdade, elas eram representadas por inteiros de 0 até 255.
Mostre a tabela completa do código ASCII.

SOLUÇÕES DAS QUESTÕES SOBRE O LAÇO WHILE EM C

O laço FOR: o que é, para que serve e como usar o FOR - Cast em C: o que é e como usar o Casting

O curso C Progressivo vai te ensinar agora o que é, provavelmente, o laço mais importante e usado na linguagem de programação C: o laço FOR, que será usado várias vezes durante nossa apostila.

O que é o laço FOR

Se você notar bem em nossa aula sobre o laço WHILE, vai notar que precisamos inicializar uma variável - chamamos ela de 'count' - para usar dentro dos parênteses do WHILE, e incrementamos ou decrementamos essa variável ao fim do laço WHILE.

Com o tempo isso se torna um pouco incômodo, pois tem que inicializar e fazer operações em lugares diferentes de seu código.

Talvez agora você não veja muitos problemas e complicações, mas com o passar do tempo e suas aplicações em C forem ficando mais complexas, isso de inicializar e mudar o valor das variáveis se torna um pouco bagunçado.

No decorrer desse tutorial de C, e de outros artigos do site, você verá que existem muitas ferramentas que servem como atalho para os programadores.
Esses atalhos visam aumentar a eficiência do programador C, para que não perca muito tempo com detalhes.

Podemos ver o laço FOR como um desses atalhos, pois, em suma, o que é possível fazer com o laço WHILE, é possível fazer com o laço FOR.

Por que não usar o WHILE, então?
Porque o laço FOR é mais simples e eficiente de ser usado, na maioria das vezes.

Mas não há nada que nos impeça de usar o laço WHILE, e algumas vezes ele até mais útil que o FOR.

A sintaxe do laço FOR: como usar o for

A sintaxe é a seguinte:

```
for(inicio_do_laço ; condição ; termino_de_cada_iteração)
{
 //código a ser
 //executado aqui
}
```

Isso quer dizer que, ao iniciar o laço for, ele faz o que está no trecho "inicio_do_laço".
Geralmente se usa para inicializar algumas variáveis (o que fazíamos antes de iniciar o laço WHILE).

Após inicializar, o for testa a condição.
Se ela resultar verdadeira, o código é executado e em seguida o 'termino_de_cada_iteração" é executado.
A condição é testada novamente, e se for verdadeira, executa o 'termino_de_cada_iteração" também, e assim continua até que a condição resulte no valor lógico FALSO (0) e o laço FOR termina.

Vamos ver na prática como usar o laço FOR através de exemplos de programas em C.

Exemplo:
Contando de 1 até 10 com o laço FOR

No exemplo a seguir, apenas declaramos a variável 'count', e iniciamos ela dentro do laço for.
O primeiro teste é se '1 <= 10', como é verdadeiro, o código que imprime o número é

executado.
Após o término da execução, o count é incrementado em 1. Agora count=2.

Na nova iteração, testamos a seguinte condição '2 <= 10', que resulta em valor lógico VERDADEIRO, por tanto o código é executado e o número 2 impresso. Agora incrementamos a variável 'count', que agora vale 3.

E assim por diante, até imprimirmos o valor 10 e 'count' ser incrementada e se tornando count=11.
No próximo teste, a condição agora é FALSA e o laço for termina.

```
#include <stdio.h>

int main(void)
{
        int count;

        for(count=1 ; count <= 10 ; count++)
        {
            printf("%d\n", count);
        }
}
```

Exemplo:
Contagem regressiva de 10 até 1, usando o laço FOR
O raciocínio é igual ao do exemplo anterior, porém vamos decrementar a variável que vai se iniciar com 10.
Veja como ficou nosso código em C:

```
#include <stdio.h>

int main(void)
{
        int count;

        for(count=10 ; count >= 1 ; count--)
            printf("%d\n", count);
}
```

Assim como no teste condicional IF ELSE, e no laço WHILE, se o código que vem após o laço tem apenas uma linha, não é necessário usar as chaves.
Se você colocar duas, ou mais, linhas de código após o IF, ELSE, WHILE ou FOR, apenas a primeira linha fará parte do teste/laço.
Muita atenção pra isso! Se não quiser correr riscos, você pode usar sem problema algum o par de chaves.

Exemplo:
Contagem progressiva e regressiva no mesmo laço FOR
Ora, se no laço WHILE podemos usar, inicializar, comparar, testar e operar matematicamente várias variáveis, também podemos fazer isso no laço FOR.

Agora inicializamos duas variáveis, testamos se ambas obedecem a uma determinada condição, e usamos dois operadores matemáticos, tudo ao mesmo tempo. Veja esse código em C e tente entender como funciona:

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

```c
int main(void)
{
        int up,
            down;

        printf("CRESCENTE \tDECRESCENTE\n");
        for(up=1, down=10 ; up<=10 && down >=1 ; up++, down--)
        {
            printf("    %d \t\t    %d\n", up, down);
        }
}
```

Exemplo:
Crie um programa em C que gera os elementos de uma P.A pedindo ao usuário o número de elementos da P.A, sua razão e seu elemento inicial.

Lembrando que a fórmula do n-ésimo termo da P.A é: an = a1 + (n-1)*razao
Fica fácil ver que:

```c
#include <stdio.h>

int main(void)
{
        int termos,
            razao,
            inicial,
            count;

        printf("Número de termos da P.A: ");
        scanf("%d", &termos);

        printf("Razão da P.A: ");
        scanf("%d", &razao);

        printf("Elemento inicial da P.A: ");
        scanf("%d", &inicial);

        for(count = 1; count <= termos ; count++)
            printf("Termo %d: %d\n", count, (inicial + (count-
1)*razao) );

}
```

Exemplo:
Crie um programa em C que gera os elementos de uma P.G pedindo ao usuário o número de elementos da P.G, sua razão e seu elemento inicial.

O exemplo dessa questão é igual ao que demos artigo sobre o laço WHILE.
Porém, vamos apresentar agora o casting.

A função pow, da biblioteca math.h recebe dois números decimais e retorna um decimal também, do tipo double.
Porém, não queremos trabalhar com o tipo double, e sim com o tipo inteiro, então temos que dizer e sinalizar isso pra função pow, de alguma maneira.

Essa maneira é o cast, que é colocar o tipo de variável que queremos obter, antes do retorno da função ou antes de uma variável.
No nosso caso, queremos que a função pow() retorne um inteiro, então fizemos: (int) pow(...)

Ou seja, basta colocarmos o tipo que queremos entre parênteses, e teremos esse tipo de variável.
Mas só fizemos isso porque o resultado de nossa P.G é inteiro. Caso usássemos decimais e colocássemos o cast pra inteiro, iríamos ficar só com a parte inteira do número decimal e perderíamos os valores decimais.

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

int main(void)
{
        int termos,
            razao,
            inicial,
            elemento,
            count;

        printf("Número de termos da P.G: ");
        scanf("%d", &termos);

        printf("Razão da P.G: ");
        scanf("%d", &razao);

        printf("Elemento inicial da P.G: ");
        scanf("%d", &inicial);

        for(count = 1; count <= termos ; count++)
            printf("Termo %d: %d\n", count, inicial*
(int)pow(razao,count-1) );
}
```

Questões sobre o laço FOR

Como já havíamos explicado no artigo sobre o laço FOR em C, esse laço é uma maneira mais simples e eficiente de fazer coisas que eram possíveis de se fazer com o laço WHILE.

O curso C Progresseivo vai propor agora as mesmas questões que já havíamos colocado para o laço WHILE, mas agora você deve tentar resolver apenas usando o laço FOR.

Apostila C Progressivo - Exercícios sobre o laço FOR

0. Programa em C dos patinhos da Xuxa
Xuxa, a rainha dos baixinhos, criou uma música que tem o seguinte formato:

n patinhos foram passear

Além das montanhas
Para brincar
A mamãe gritou: Quá, quá, quá, quá
Mas só n-1 patinhos voltaram de lá.

Que se repete até nenhum patinho voltar de lá.
Ao final, todos os patinhos voltam:

A mamãe patinha foi procurar
Além das montanhas
Na beira do mar
A mamãe gritou: Quá, quá, quá, quá
E os n patinhos voltaram de lá.

Crie um programa em C que recebe um inteiro positivo do usuário e exibe a música inteira na tela, onde o inteiro recebido representa o número inicial n de patinhos que foram passear.

1. Programa em C que mostra os números ímpares
Escreva um aplicativo em C mostra todos os números ímpares de 1 até 100.

2. Programa em C que mostra os números pares
Escreva um aplicativo em C mostra todos os números pares de 1 até 100.

3. Programa em C que mostra os números pares e ímpares
Escreva um aplicativo em C que recebe inteiro e mostra os números pares e ímpares (separados), de 1 até esse inteiro.

4. Programa em C que calcula a média das notas de uma turma
Escreva um programa que pergunte ao usuário quantos alunos tem na sala dele.
Em seguida, através de um laço while, pede ao usuário para que entre com as notas de todos os alunos da sala, um por vez.

Por fim, o programa mostra a média, aritmética, da turma.

5. Achando o maior número
Achar o maior, menor, média e organizar números ou sequências são os algoritmos mais importantes e estudados em Computação. Em C não poderia ser diferente.
Em nosso curso, obviamente, também não será diferente.

Escreva um programa em C que solicita 10 números ao usuário, através de um laço while, e ao final
mostre qual destes números é o maior.

6. Achando os dois maiores números
Escreva um programa em C que solicita 10 números ao usuário, através de um laço while, e ao final
mostre os dois maiores números digitados pelo usuário.

7. Quadrado de asteriscos
Escreva um programa que lê o tamanho do lado de um quadrado e imprime um quadrado daquele tamanho com asteriscos. Seu programa deve funcionar para quadrados com lados de todos os tamanhos entre 1 e 20.

Para lado igual a 5:
*****
*****
*****
*****
*****

8. Quadrado de asteriscos e espaços em branco
Escreva um programa que lê o tamanho do lado de um quadrado e imprime um quadrado daquele tamanho com asteriscos e espaços em branco. Seu programa deve funcionar para quadrados com lados de todos os tamanhos entre 1 e 20.
Para lado igual a 5:
*****
* *
* *
* *
*****

9. Tabuada em C
Escreva um programa que pergunta um número ao usuário, e mostra sua tabuada completa (de 1 até 10).

10. Tabela ASCII em C
Se você lembrar bem, quando estudamos as variáveis do tipo caractere, *char*, dissemos que, na verdade, elas eram representadas por inteiros de 0 até 255.
Mostre a tabela completa do código ASCII.

SOLUÇÕES DAS QUESTÕES SOBRE O LAÇO FOR

Os comandos CONTINUE e BREAK: pausando e alterando o fluxo de laços e testes condicionais

Até o momento, em nosso curso de C, sempre percorremos os laços do início até o fim, no intervalo que escolhemos.

Porém, nem sempre isso é necessário. Pode ser que no meio do caminho consigamos obter o resultado que esperávamos e a continuação do laço seria mera perda de tempo.

Para controlar melhor os laços, vamos apresentar os comandos BREAK e CONTINUE, que usaremos ao longo de toda nossa apostila de C.

O comando BREAK em C: como usar

Break, em inglês, significa pausa, ruptura ou parar bruscamente.

E é basicamente isso que esse comando faz com os laços: se você colocar o comando break dentro de um laço e este for executado, ele vai parar imediatamente o laço e o programa continua após o laço.

Vamos dar um exemplo prático, e você verá melhor como funciona e qual a utilidade do comando break em C, que também existe em diversas outras linguagens de programação, como C++ e Java.

Sua principal função é evitar cálculos desnecessários.
Muitas vezes rodamos um looping através de um laço (como FOR ou WHILE), para encontrar alguma informação. O break serve para terminar o laço em qualquer momento (como no momento em que encontrarmos essa informação, ao invés de ter que esperar que o looping termine).

Exemplo de código comentando em C

Ok, assumimos.
A explicação que demos pode parecer um pouco confusa à primeira vista.

Mas como sempre, você vai entender melhor o uso dos comandos BREAK e CONTINUE através de exemplos resolvidos e comentados de questões. Vamos mostrar alguns exemplos, e você entenderá melhor para que serve e como usar os comandos BREAK e CONTINUE, em C.

Ache o primeiro número, entre 1 e 1 milhão que é divisível por 11, 13 e 17.
Isso quer dizer que temos que encontrar o menor número que, quando dividimos ele por 11, por 13 e por 17 dê resto da divisão nulo.

Uma possível solução é a seguinte:

```
#include <stdio.h>

int main()
{
        int count,
            numero=0;

        for(count=1 ; count<=1000000 ; count++)
        {
            if(numero == 0)
                if((count%11==0) && (count%13==0) && (count%17==0))
                    numero=count;
        }

        printf("O numero e: %d", numero);

}
```

A variável 'count' é a que irá contar de 1 até 1 milhão.
O número que queremos será armazenado em 'numero' e inicializamos com valor 0.
Enquanto 'numero' tiver valor 0, é porque o número que estamos procurando ainda não foi achado, pois quando ele for achado a variável 'numero' irá mudar de valor e passará a ser o número que queremos.

Note que esse valor ficará sempre armazenado na variável 'numero', pois agora o primeiro teste condicional nunca mais será satisfeito.

Você vai ver que o número em questão é 2431.
E o que acontece depois que descobrimos esse número?
Ora, o laço continua a contar de 2431 até 1000000, pra nada.
Então o computador está gastando processamento à toa.

Aqui na minha máquina, essa operação durou 10milisegundos
Vamos dar uma otimizada nisso usando o comando BREAK.

Exemplo de código comentado em C

Ache o primeiro número, entre 1 e 1 milhão que é divisível por 11, 13 e 17. Use o comando BREAK.

A idéia do break é simples: quando acharmos o valor que queremos, damos um break e o programa sai do laço.
O código vai ficar assim:

```
#include <stdio.h>

int main()
{
        int count,
            numero;

        for(count=1 ; count<=1000000 ; count++)
            if((count%11==0) && (count%13==0) && (count%17==0))
            {
                numero=count;
                break;
            }


        printf("O numero e: %d", numero);

}
```

Note que agora só precisamos de um teste condicional IF.
Aqui na minha máquina, essa operação levou 2milisegundos.
Ou seja, 5x mais rápida. Isso se deve ao fato do laço não precisar mais ir de 2432 até 1 milhão.
Quando ele achou o número que eu queria, ele parou de contar.

Você pode pensar: "Ah, 8milisegundos não fazem a mínima diferença".
Bom, pra esse tipo de aplicação simples, realmente não faz mesmo, é praticamente

instantâneo, ainda mais na linguagem C que é reconhecidamente uma das mais rápidas.

Agora imagine um jogo de alta performance ou o código de um Sistema Operacional, onde alguns códigos e rotinas são rodados milhões ou até bilhões de vezes? Esses 5ms virariam segundos ou minutos de lag ou travamento.
Por isso é sempre bom fazer um código limpo e eficiente, sem perder processamento à toa.

O comando break será mais utilizado no próximo artigo do C Progressivo, sobre o teste condicional SWITCH.

O comando CONTINUE em C: como usar

O comando CONTINUE, quando inserido dentro de algum laço, faz com que a iteração atual seja cancelada, e o laço prossegue na próxima iteração.

Ou seja, o BREAK faz todo o laço parar.
Já o CONTINUE, faz somente com que a iteração atual pare, pulando pra próxima.

Exemplo:
Faça um aplicativo em C que some todos os números, de 1 até 100, exceto os múltiplos de 5.

Fazemos o laço for percorrer de 1 até 100.
Testamos se cada número deixa resto 0 quando dividido por 5. Caso deixe, é porque é múltiplo de 5 e não devemos somar.
Para não somar, simplesmente pulamos essa iteração.

Porém, se não for múltiplo de 5, é porque a iteração não foi pulada, ela continua, então vamos somar esse número na soma total.

```
#include <stdio.h>

int main()
{
        int count,
            soma;

        for(count=1, soma=0 ; count<=100 ; count++)
        {
            if( count%5 ==0)
                continue;

            soma = soma + count;
        }


        printf("Soma %d", soma);

}
```

Diferente do BREAK, que pode ser usado em laços e no SWITCH (que veremos a seguir), o CONTINUE é utilizado em laços somente.

O teste condicional SWITCH: o que é, para que serve e como usar o switch em C

Provavelmente você nunca notou mas, a todo instante, estamos fazendo escolhas quando usamos um computador ou outro dispositivo digital qualquer.

Escolhemos ícones, teclas do teclado, que botão vamos clicar, que menus vamos abrir, que opção do menu vamos selecionar etc.

O curso C Progressivo vai ensinar, agora, uma ferramenta importantíssima em programação: o uso do switch para escolhas.

SWITCH em C: o que é e como usar o comando

SWITCH é um comando em C que serve para fazer testes condicionais, testando igualdades, onde podemos usar várias opções de comparações.

Assim como o nosso conhecido par IF ELSE, o comando switch também serve para fazer testes condicionais.

Imagine que você quer testar um valor digitado pelo usuário com 10 números.
Você poderia fazer com IF, tranquilamente.
Porém, seu código iria ficar enorme e você teria que digitar várias vezes IF (...), IF(...)

Visando reduzir isso, vamos aprender como usar o comando *switch*, que iremos usar várias vezes durante nossa apostila de C, para criar menus, por exemplo, onde iremos exibir uma série de opções, o usuário vai escolher uma e vamos saber qual opção ele escolheu através de um comando *switch*.

A sintaxe do switch é a seguinte:

```
switch( opção )
{
    case opção1:
            comandos caso a opção 1 tenha sido escolhida
            break;

    case opção2:
            comandos caso a opção 2 tenha sido escolhida
            break;

    case opção3:
            comandos caso a opção 3 tenha sido escolhida
            break;

    default:
            comandos caso nenhuma das opções anteriores tenha sido
escolhida
}
```

O switch vai comparar a variável 'opção' com os 'case'. Se ele achar uma opção (case) que seja igual, ele vai rodar o código que vem após esse case, e antes do próximo case.
Caso nenhum case seja igual a 'opção', o código que está default é o que será rodado.

Caso a 'opção' seja um char, coloque entre aspas simples ' ', caso seja string coloque entre aspas duplas " " e caso seja um número, não é necessário colocar nenhum tipo de aspas.

Exemplo de código:

Crie uma calculadora usando a instrução SWITCH, que pergunte qual das operações básicas quer fazer (+, -, * e /), em seguida peça os dois números e mostre o resultado da operação matemática entre eles.

```c
#include <stdio.h>

int main()
{
        char operacao;
        float num1,
              num2;

        printf("Escolha sua operação [+ - * / ]: ");
        scanf("%c",&operacao);

        printf("Entre com o primeiro número: ");
        scanf("%f",&num1);

        printf("Entre com o segundo número: ");
        scanf("%f",&num2);

        switch( operacao )
        {
            case '+':
                printf("%.2f + %.2f = %.2f", num1, num2, num1 + num2);
                break;

            case '-':
                printf("%.2f - %.2f = %.2f", num1, num2, num1 - num2);
                break;

            case '*':
                printf("%.2f * %.2f = %.2f", num1, num2, num1 * num2);
                break;

            case '/':
                printf("%.2f / %.2f = %.2f", num1, num2, num1 / num2);
                break;

            default:
                printf("Você digitou uma operacao invalida.");

        }

}
```

O switch sem o break

No exemplo passado, você viu que somente um dos case era selecionado para cada operação que efetuamos.

Porém, isso só ocorre por conta do break em cada case.

Se você tirar o break, o switch identifica o case que é igual a 'operacao', executa ele e TODOS OS QUE ESTÃO ABAIXO até o fim, ou até encontrar um break!

É como se os case se acumulassem.

Rode o exemplo a seguir:

```
#include <stdio.h>

int main(void)
{
        int continuar=1;
        char opcao;

        while(continuar)
        {
            system("clear || cls");
            printf("Repetir? (S/s)im (N/n)ao\n");
            scanf(" %c",&opcao);

            switch(opcao)
            {
                case 's':
                case 'S':
                    break;

                case 'n':
                case 'N':
                    continuar = 0;
            }

        }

}
```

O comando system("clear") serve para limpar a tela em sistemas operacionais do tipo Linux, e system("cls") limpa a tela caso você use Windows. Então system("clear || cls") limpa a tela, independente de qual sistema você esteja usando.

Nota-se que, enquanto continuar=1, o laço WHILE continua a ocorrer e só termina quando 'continuar' receber valor 0.

Se digitarmos 's' o primeiro case é selecionado. Como ele não tem break, o próximo também ocorre, que é o case caso opcao='S'.
Esse tem break. Ou seja, pra continuar a repetir basta digitar 'S' ou 's'.

Se digitar 'n', vai cair no case onde opcao='n' e onde opcao='N', pois não tem break no opcao='n'.
Então, 'continuar' recebe valor 0 e o laço WHILE termina.

Exemplo:
Suponha que você atrasou uma conta. A cada mês que você deixa de pagar, será cobrado 1% de juros no valor inicial.
Ou seja, se você atrasar um mês, irá pagar 1%. Se atrasar 3 meses, irá pagar 3% etc.
Vamos supor que você pode atrasar, no máximo, 5 meses.

O programa pede, como entrada, dois valores:
- um float: com o valor de sua dívida inicial (valor_i)
- um inteiro: de 0 até 5, que são os meses de atraso.

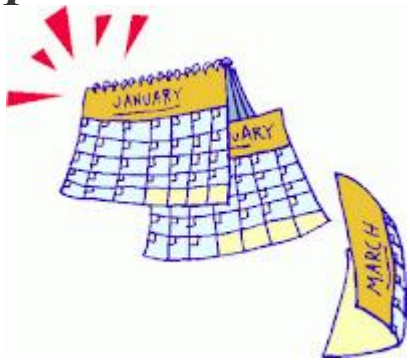
Faça um programa em C que calculo o juros de atraso. Use switch e case acumulados.

```c
#include <stdio.h>

int main(void)
{
        float valor_i,
              valor_f;
        int   juros=0;

        int meses;

        printf("Qual o valor inicial da dívida: ");
        scanf("%f", &valor_i);

        printf("Você vai atrasar quantos meses [1-5]?: ");
        scanf("%d", &meses);

        switch( meses )
        {
            case 5:
                juros++;
            case 4:
                juros++;
            case 3:
                juros++;
            case 2:
                juros++;
            case 1:
                juros++;
                break;
            default:
                printf("Você não digitou um valor válido de meses\n");

        }
        printf("Juros: %d%\n",juros);
        valor_f=( (1 + (juros/100.0))*valor_i);
        printf("Valor final da dívida: R$ %.2f\n", valor_f);

}
```

Caso tenha atrasado 5 meses, o valor da variável 'juros' é incrementado 5 vezes.

Se atrasou 4 meses, o 'juros' é incrementado 4 vezes, e assim por diante.

Programa em C que diz quantos dias o mês possui



Vamos criar agora um programa que pede o mês ao usuário, e retorna quantos dias esse mês possui.
Vamos resolver usando apenas SWITCH e cases acumulados.

Exercício:
Crie um programa em C que receba um inteiro, de 1 até 12, representando os meses do ano e retorne o número de dias do mês.

Use switch e não use break.
Use acúmulo de case e suponha que fevereiro tenha sempre 28 dias.

Como programar: aplicativo que diz quantos dias um mês possui em C

Inicialmente, a variável 'dias' é declarada como tendo 31 dias.

Caso seja o mês 4, 6, 9 ou 11, é subtraído 1 da variável 'dias' e o programa informa que esses meses possuem 30 dias.

Caso seja o mês 2, é subtraído 2 de 'dias', ficando 28 dias para o mês de fevereiro.

Caso não seja nenhum desses meses, não cai no switch, então continua com 31 dias (que são os meses 1, 3, 5, 7, 8, 10 e 12).

Assim, nosso código C fica:

```
#include <stdio.h>

int main()
{
        int mes,
            dias=31;

        printf("Digite o mes [1-12]: ");
        scanf("%d", &mes);

        if(mes>12 || mes<1)
```

```
            printf("Mes invalido");
        else
            switch( mes )
                {
                    // fevereiro: subtraímos 2 dias aqui e 1 dia no
próximo case
                    case 2:
                        dias -=2;

                    //meses que possuem 30 dias: só subtraímos 1 dia
                    case 4: case 6: case 9: case 11:
                        dias--;

                }

        printf("O mes %d possui %d dias", mes, dias);

}
```

O que é o laço DO WHILE e como usar em C

Vamos apresentar o último, e não menos importante, laço de nosso curso C Progressivo: o laço do ... while.

Como o nome pode sugerir, ele tem muito a ver com o laço WHILE, que já estudamos e treinamos seu uso em tutoriais passados de nossa **apostila de C**.

Para que serve o laço DO WHILE em C

Se você notar bem, em nosso artigo sobre o laço WHILE, pode reparar que esse laço só roda se a condição for verdadeira.
Se essa condição for uma variável ou comparação, por exemplo, devemos sempre nos certificar que, inicialmente, essa condição vai ser verdadeira.

Geralmente inicializamos as variáveis ou pedimos algum dado para o usuário, tudo para nos certificarmos que o programa vai entrar no WHILE, o que as vezes pode ser um pouco incômodo e nada eficiente.

O que o laço DO WHILE faz é executar, PELO MENOS UMA VEZ, o que está dentro dele e só ao final da execução é que ele faz o teste, usando o velho e conhecido laço WHILE.
Ou seja, temos a garantia que o laço vai ser executado uma vez, sempre precisar inicializar variável ou pedir dados ao usuário antes do WHILE (pois fazer isso pode bagunçar o sistema).

Como declarar e usar o laço DO WHILE em C

Lembrando que DO, em inglês e nesse contexto, significa "faça" e WHILE significa "enquanto".

Ou seja, esse laço quer dizer "faça isso" -> código -> "enquanto essa condição for verdadeira, repita" (note como os nomes dos comandos não são escolhidos de forma aleatória, tudo em programação C faz sentido e foi criado com propósitos de facilidade a vida do programador e aumentar sua eficiência).

Resumindo, ele sempre executa seu código ao menos uma vez, o que é uma mão na roda em certas ocasiões.

E qual a diferença do laço DO WHILE para o WHILE ?
Bom, para o WHILE rodar é necessário que a condição seja sempre verdadeira.
Logo, se ela for inicialmente FALSE, o looping while nunca irá rodar.

Já o DO WHILE vai rodar sempre, ao menos uma vez, mesmo que a condição existente no WHILE seja falsa (se for falsa, vai rodar só uma vez e termina).

A sintaxe desse comando é a seguinte

```
do
{
   //código que
   //será repetido
   //aqui

} while (condição);
```

Vale realçar aqui que esse laço, diferente dos outros, **POSSUI PONTO E VÍRGULA** no final! Não se esqueçam!
No mais, não há segredos, é igual ao WHILE, a lógica é a mesma. Sabendo bem o WHILE, saberá bem o laço DO WHILE.

Vamos mostrar um exemplo em que o DO WHILE é bem usado em programação C: exibição de menu.

Exemplo de uso do laço DO WHILE: Criando um MENU

Crie um menu que apresente algumas opções ao usuário, usando o laço DO WHILE e o teste condicional SWITCH em C, para selecionar as opções.
Uma dessas condições é o término da exibição do menu.

Vamos usar apenas uma variável, o inteiro 'opcao' que irá selecionar a opção que o usuário quiser.
Dentro do escopo do do {}, mostramos o menu de opções e pedimos pro usuário digitar algo.

A condição do while é a própria variável.
Se o usuário digitar 0, o WHILE irá receber FALSE como condição e irá encerrar.
Caso o WHILE receba qualquer outra coisa diferente de 0, receberá valor lógico TRUE e irá continuar o laço DO WHILE.

Pois bem, já sabemos o que ocorre quando 0 é digitado.
Então, dentro do SWITCH, quando a opção 0 é selecionada, apenas limpamos a tela e damos

um aviso de saída.

Caso 1 ou 2 seja digitado, idem: limpamos a tela e mostramos uma mensagem.

Caso 3 ou 4 seja escolhido, limpamos a tela e não acontece nada, simplesmente o menu é exibido novamente.

Se o usuário não digitar nenhuma dessas opções, é porque ele digitou algo errado e vai cair no case DEFAULT do SWITCH, e informamos que ele errou na escolha da opção.

Assim, nosso código de menu fica:

```
#include <stdio.h>

int main(void)
{
        int opcao;

        do
        {
            printf("\t\t\nMenu do curso C Progressivo\n");
            printf("0. Sair\n");
            printf("1. Dar Boas vindas\n");
            printf("2. Dar Oi\n");
            printf("3. Repetir o menu\n");
            printf("4. Ler mais uma vez o menu\n");

            printf("Opcao: ");
            scanf("%d", &opcao);

            switch( opcao )
            {
                case 0:
                        system("cls || clear");
                        printf("Saindo do menu...\n");
                        break;
                case 1:
                        system("cls || clear");
                        printf(" Bem-vindo ao menu do portal C
Progressivo! \n");
                        break;
                case 2:
                        system("cls || clear");
                        printf(" Oi! \n");
                        break;
                case 3:
                case 4:
                        system("cls || clear");
                        break;
                default:
                        system("cls || clear");
                        printf("Opcao invalida! Tente novamente.\n");
            }
        } while(opcao);

}
```

Obviamente esse é um exemplo simples, que não faz nada, mas mostra bem o funcionamento do DO WHILE.

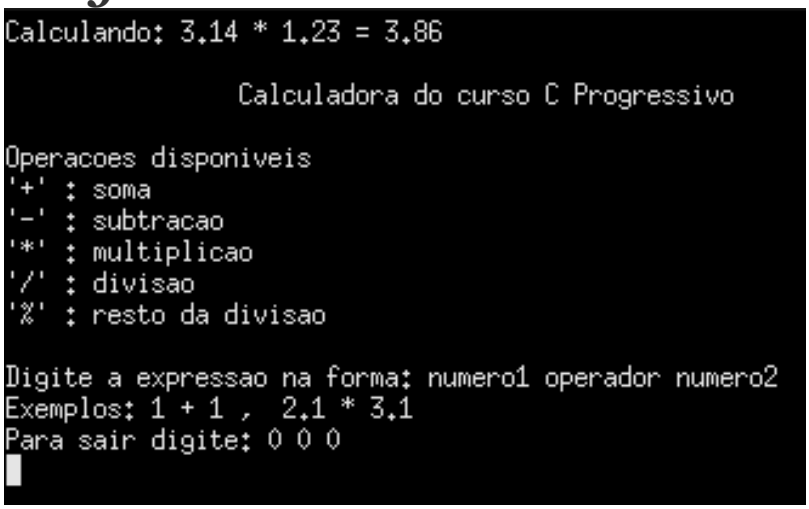
E menus são, obviamente, muito usados em programação.

Em nossa apostila online de C, vamos usar menus em aplicativos de simulação de um sistema bancário, em jogos e vários outros aplicativos. Portanto, é essencial que tenha entendido bem o funcionamento do DO WHILE com SWITCH.

Exercício de C:

No artigo sobre o teste condicional SWITCH em C, mostramos como usar ele para fazer uma calculadora.
Refaça essa calculadora, com o mesmo SWITCH, mas agora mostrando as operações matemáticas como opções de um menu, dentro de um laço DO WHILE.

Programa: Criando uma calculadora em C



No artigo passado, sobre o laço DO WHILE em C, propomos para você um exercício:

No artigo sobre o teste condicional SWITCH em C, mostramos como usar ele para fazer uma calculadora.
Refaça essa calculadora, com o mesmo SWITCH, mas agora mostrando as operações matemáticas como opções de um menu, dentro de um laço DO WHILE.

Vamos resolver esse exercício e comentar totalmente seu código, agora em nossa **apostila de C**!

Como criar uma calculadora em C

Você já saiu do básico e aprendeu todos os testes condicionais e laços na linguagem.

Já está na hora de fazer algo realmente útil, que você possa se orgulhar e até mostrar aos amigos.

Ou seja, vamos fazer uma calculadora que fazer as operações de soma, subtração, multiplicação, divisão e resto da divisão.

Tudo isso com base nos conhecimentos que acumulamos até aqui.
Obviamente, não será uma calculadora complexa e de utilidade geral, é mais para termos

noção de como usar os conhecimentos que aprendemos até aqui.

Ela também não é uma aplicação 'robusta', que seja blindada e segura. Ela é facilmente 'hackeável'.
Por exemplo, ela está programada para receber números, então se você digitar uma letra, irá 'quebrar' nosso programa.

Mas ao passo que vamos estudando e evoluindo em nossa apostila, iremos criar aplicações cada vez melhores e mais seguras

Pois bem, vamos começar!

A lógica de uma calculadora na linguagem C

Vamos usar apenas três variáveis nesse aplicativo C: dois floats (que vão representar os dois números) e uma variável do tipo char, que irá armazenar o tipo de operação o usuário quer: +, -, *, / ou %

Conforme pedimos, o menu será exibido através do laço DO WHILE.

Lembrando que temos que colocar uma condição para esse laço continuar. Você pode criar a sua, a minha foi a seguinte: se o usuário digitar num1=0, oper=0 e num2=0, o programa termina.

Pois bem, mostramos o menu ao usuário (meros *printf*, que você já domina totalmente) e demos exemplos de como deve ser a entrada do usuário. Que deve ser: **numero operador numero**

Se colocarmos os 3 *scanf* em sequência, dá para pegar o primeiro número, o char e em seguida o segundo número, mas o usuário tem que entrar com os dados da seguinte maneira: digita o primeiro número, dá um espaço, digita o operador e dá outro espaço.

Após isso, limpamos a tela e mostramos ao usuário a operação matemática que ele escolheu:

numero operador numero2 =

E o resultado dessa operação? Hora, vai depender do operador e números que ele tenha escolhido.

Essa resposta vai ser fornecida através do switch.

Mandamos o operador para o switch, que vai selecionar a operação e mostrar um printf com o resultado.

Caso o usuário tenha digitado um operador inválido, o switch manda isso pro default.

Lá no default temos que fazer uma espécie de tratamento de informação, para saber se o usuário digitou um operador inválido ou se digitou 0 0 0, pois se tiver digitado 0 0 0 é porque ele quer terminar o programa.

Isso é facilmente resolvido com o nosso conhecido e amado teste condicional IF ELSE.

```
#include <stdio.h>

int main(void)
{
        float num1,
              num2;
        char oper;

        do
        {
            printf("\t\tCalculadora do curso C Progressivo\n\n");

            printf("Operacoes disponiveis\n");
            printf("'+' : soma\n");
            printf("'-' : subtracao\n");
            printf("'*' : multiplicao\n");
            printf("'/' : divisao\n");
            printf("'%%' : resto da divisao\n");

            printf("\nDigite a expressao na forma: numero1 operador
numero2\n");
            printf("Exemplos: 1 + 1 ,  2.1 * 3.1\n");
            printf("Para sair digite: 0 0 0\n");


            scanf("%f", &num1);
            scanf(" %c",&oper);
            scanf("%f", &num2);

            system("cls || clear");

            printf("Calculando: %.2f %c %.2f = ", num1,oper,num2);


            switch( oper )
            {
                case '+':
                        printf("%.2f\n\n", num1 + num2);
                        break;

                case '-':
                        printf("%.2f\n\n", num1 - num2);
                        break;

                case '*':
                        printf("%.2f\n\n", num1 * num2);
                        break;

                case '/':
                        if(num2 != 0)
```

```
                        printf("%.2f\n\n", num1 / num2);
                    else
                        printf("Nao existe divisao por 0\n\n");
                    break;

                case '%':
                        printf("%d\n\n", (int)num1 % (int)num2);
                        break;

                default:
                        if(num1 != 0 && oper != '0' && num2 != 0)
                            printf(" Operador invalido\n\n ");
                        else
                            printf(" Fechando calculadora!\n ");
            }

        }while(num1 != 0 && oper != '0' && num2 != 0);

}
```

Exercícios envolvendo testes e laços em C

Parabéns! Se chegou até aqui, é porque estudou bastante e persistiu em nossa apostila de C! Esses passos iniciais em programação C são, sem dúvidas, os mais difíceis, e todos já tiveram problemas e dificuldades, inclusive todos nós da equipe do curso C Progressivo.

Antes de estudar funções, porém, você precisa fazer um treino especial, com mais uma leva de questões.

Exercícios sobre testes condicionais e laços

Para fechar mais uma seção com chave de ouro, vamos propor algumas questões para você fazer, mas não vamos dizer "faça usando isso", "faça daquele jeito" ou "aplique esse conhecimento".

Afinal, na vida real como programador C, ninguém vai te dizer como você deve fazer seu trabalho.
Você tem que usar sua criatividade, tem que pensar e desenrolar.

Essas questões, portanto, não tem apenas uma única e correta solução.
Cada programador vai ter ideias diferentes e maneiras diferentes de resolver o problema.

Pense, relembre as aulas e tente resolver da maneira mais simples, eficiente e bonita...ou simplesmente tente resolver, já será um passo importante.

Use:
Teste condicional IF ELSE
Laço WHILE
Laço FOR
Teste condicional SWITCH, com ou sem CONTINUE e BREAK
Laço DO WHILE

Fique à vontade para criar sua própria solução, da maneira que mais lhe convier.

0. Escreva um programa em C que recebe 'n' números do usuário, e recebe o número 'n' também, e determine qual destes números é o menor.

1. Escreva um programa em C que recebe um inteiro 'n' do usuário e calcula o produto dos números pares e os produtos dos números ímpares, de 1 até n.

2. Faça um programa em C que recebe um inteiro do usuário e calcula seu fatorial.
O fatorial de 'n' é dado por:
n*(n-1)*(n-2)...*3*2*1
e é representado por n!

3. Crie um aplicativo bancário em C que pede o valor do depósito inicial, o valor do investimento mensal e o número de meses que o dinheiro vai ficar rendendo na poupança. Após isso, calcule o lucro obtido, sabendo que o juro da poupança é de 0,5%.

4. Crie um programa em C que peça um número inteiro ao usuário, e imprima a seguinte tabela:
1
2 4
3 6 9
4 8 12 16
...
5. Escreva um programa que peça um númer 'n' ao usuário, e que gere um novo **n** de acordo com a seguinte regra:

• se n é par, n = n / 2
• se n é impar, n = 3 * n + 1
• imprime n
• O programa deve parar quando x tiver o valor • igual a 1. Por exemplo, para n = 13, a saída será:
40 -> 20 -> 10 -> 5 -> 16 -> ˜8 -> 4 -> 2 -> 1

Desafio dos números de FIbonacci
Crie um aplicativo em C que peça um número inteiro ao usuário - **'n'** - e exiba o n-ésimo termo da série de Fibonacci, sabendo que o primeiro termo é 0, o segundo é 1 e o próximo número é sempre a soma dos dois anteriores.

Desafio do diamante de asteriscos:
Escreva um aplicativo em C que peça um número inteiro ímpar ao usuário e desenhe um diamante no seguinte formato:

```
    *
   ***
  *****
 *******
*********
 *******
  *****
   ***
    *
```

Nesse caso, o número é 9, pois há 9 colunas e 9 asteriscos na linha central.

Super hiper mega desafio de Fibonacci
Fazer o desafio dos números de FIbonacci, mostrador anteriormente, usando apenas duas variáveis.

Soluções e códigos comentados das questões