

Tutorial de Conceitos Básicos da Linguagem C

O necessário para programar em C

Dando início a nossa [apostila de Programação C online](#), esse artigo tem por objetivo esclarecer as dúvidas iniciais:

- Como começar a programar em C?
- O que preciso baixar para programar em C?
- Onde vou compilar e rodar meus programas em C?

Ferramentas Necessárias para programar em C

- Compilador
- Debugger
- Editor de texto

Você vai escrever seus códigos de programação C em qualquer editor de texto e vai usar o compilador.

O compilador converte seu código para código de máquina (um código que só a máquina entende, para rodar no seu computador - o famoso binário) e o debugger faz o debugging, ou seja, checka se há erros no seu código.

Porém, fazer isso tudo manualmente dá muito trabalho. Existe um tipo de programa que faz isso tudo sozinho.

É a IDE, *Integrated Development Environment*, ou seja, o ambiente de desenvolvimento integrado.

A título de informação, vamos apresentar três IDE's: o Dev-C++, Visual Studio e o Code::Blocks, porém, **aconselhamos o uso do Code Blocks para iniciantes**.

Que programa escolher para programar em C

- Dev-C++: desatualizado e com muitos erros

Este é o mais usado e indicado nas faculdades e na Internet. Mas se é o mais usado e indicado, por que o [curso C Progressivo](#) não indica?

Porque ele é obsoleto! Ele costumava ser bom, e por isso era muito indicado.

MAS ELE PAROU DE SER DESENVOLVIDO!

O PROJETO DO DEV-C++ FOI ABANDONADO!

Mas continuaram a usar e recomendar, principalmente para iniciantes.

Porém, conforme você for avançando, ele ficará nitidamente ruim e desatualizado, irá prejudicar MUITO você!

Infelizmente, seu debugger é cheio de erros! Você poderá se prejudicar caso erre e o Dev-cpp não te alerte sobre os erros.

- **Microsoft Visual Studio: bom, poderoso e pago**

O Visual Studio é tão poderoso que os desenvolvedores da Microsoft fazem o próprio Windows e seus programas/sistemas são feitos usando o Visual Studio. Porém, é da Microsoft. Ou seja, pra usar tudo que a ferramenta tem a oferecer, você tem que pagar - e muito.

A Microsoft, como forma de marketing, porém, lançou uma versão gratuita do Visual Studio, o Visual Studio Express.

Eu, particularmente, acho ele muito pesado para um iniciante. Quem está começando não vai usufruir nem 10% do que ele tem a oferecer, embora tenha baixado centenas MB.

É como matar uma mosca com uma bala de canhão.

Vá com calma. Caso tenha interesse, no futuro, e queira criar aplicações gráficas para Windows (inclusive para o Windows 8), Windows Phone, tecnologia .NET e web, você pode começar a usufruir melhor os recursos dessa poderosa ferramenta de desenvolvimento.

Porém, é sempre bom se informar:

[Site do Visual Studio Express](#)

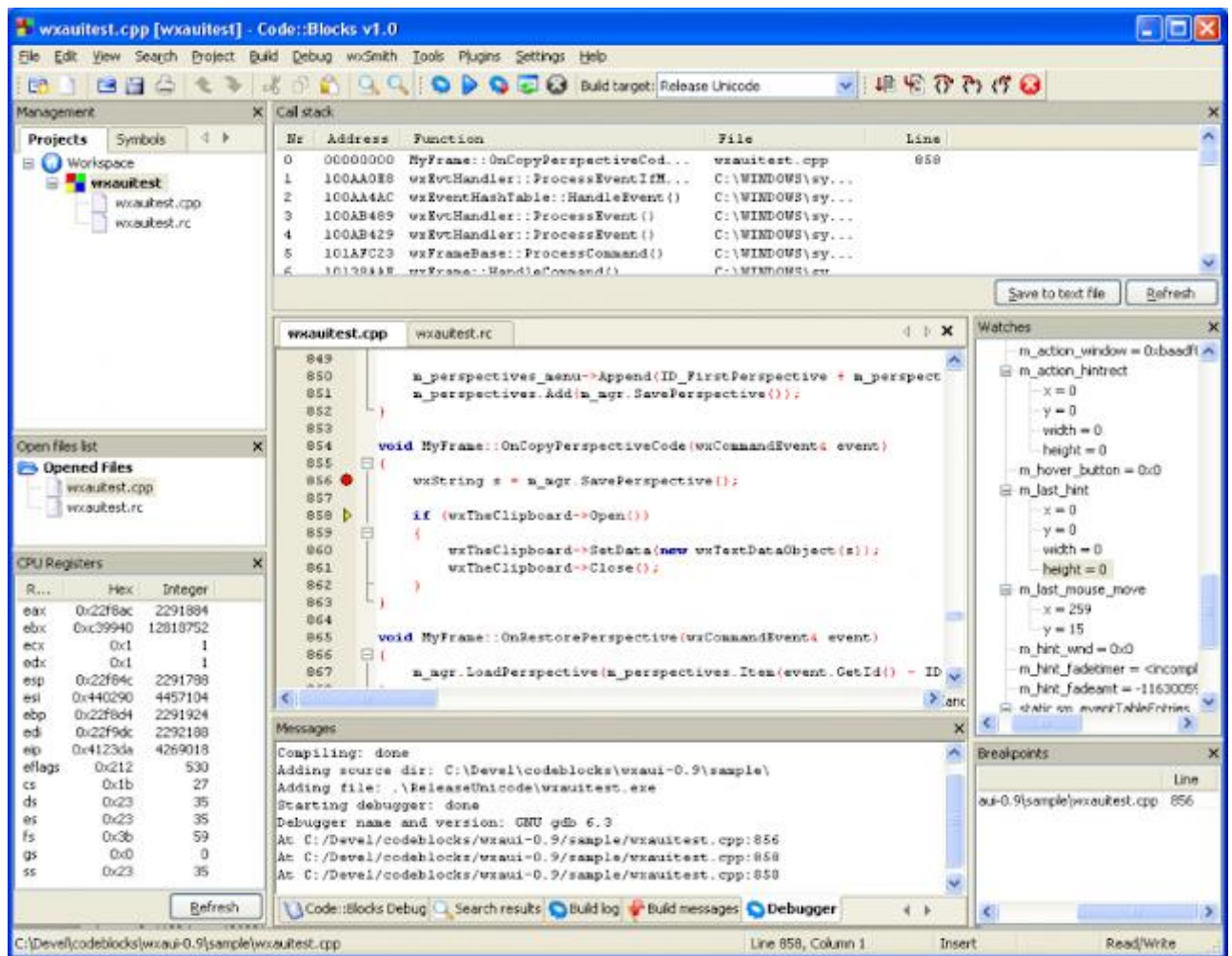
[Informações sobre a tecnologia .NET e cursos oferecidos pela Microsoft para seus produtos](#)

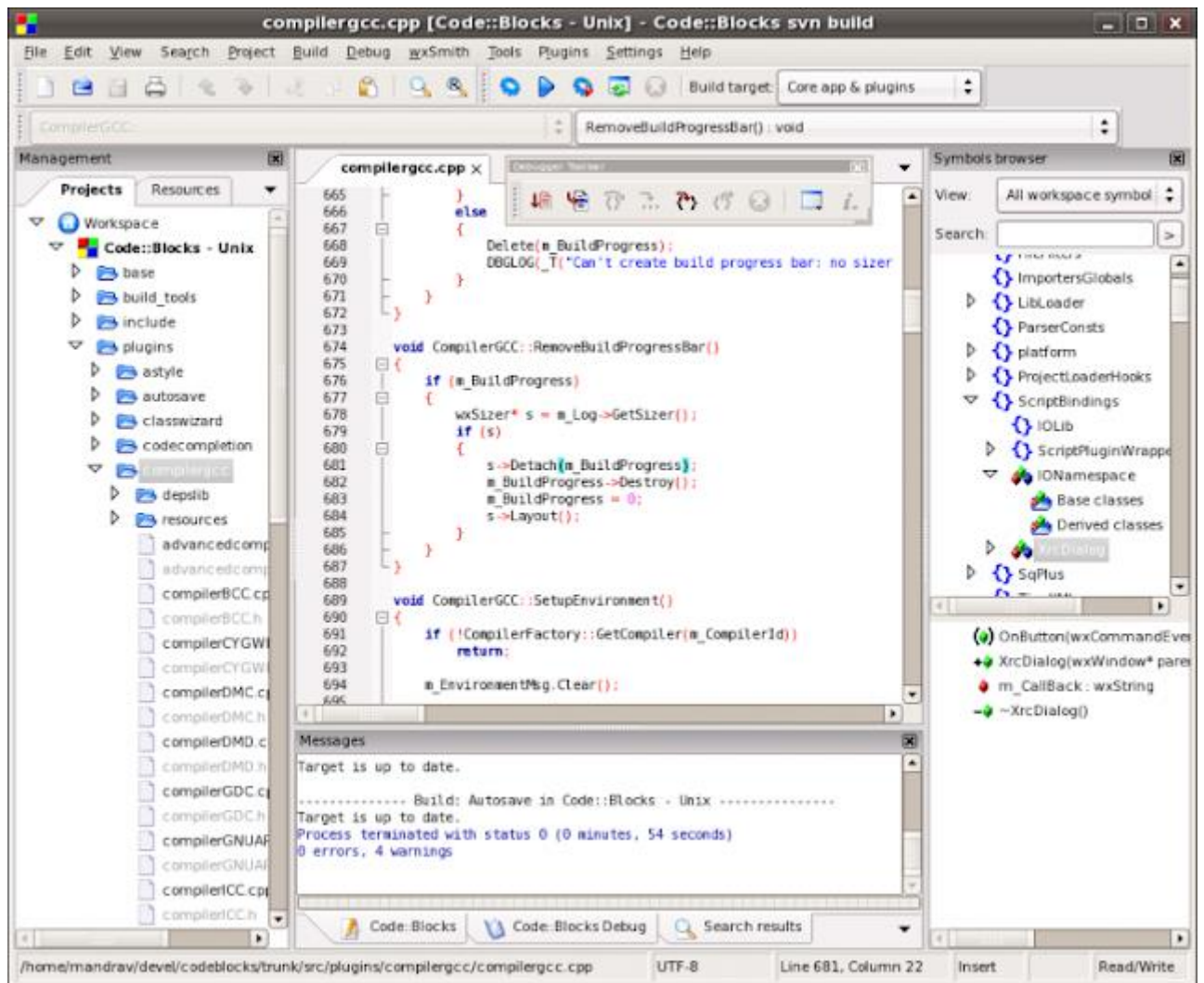
- **Code::Blocks: gratuito, leve, open source e cross platform**

Veja as razões do Code::Blocks ser melhor e mais recomendado para iniciantes:

- gratuito
- leve
- open source (é possível ver seu código-fonte, como foi feito)
- cross platform (funciona em várias plataformas, como Windows e Linux)
- está atualizado
- está em desenvolvimento
- é possível expandir suas funcionalidades através dos plugins
- é leve, principalmente se comparado com o Microsoft Visual Studio

Screenshots do Code::Blocks:





Porque usaremos o Code::Blocks no curso de Programação C

Assim como o Code Blocks, o curso de Programação C, *C Progressivo*, é gratuito. Então não apoiaremos a pirataria nem o uso de software pagos.

Felizmente, existem milhões de pessoas ao redor do mundo empenhadas em criar ferramentas boas que não deixam a desejar em **absolutamente nada** em relação as pagas.

Instalando o Code::Blocks, você já tem o debugger, compilador e editor de texto.

Isso mesmo. Não precisa baixar mais nada, somente a IDE.

Ao escrevermos o código, o Code Blocks já organiza automaticamente nosso código e quando colocarmos o programa para rodar, ele nos mostrará onde os erros estão.

Caso exista erros, o programa rodará diretamente do Code Blocks.

Então sem perda de tempo, baixe o programa.

O site do programa é: <http://www.codeblocks.org/>

Navegue até a seção de download e escolha sua plataforma, Windows, Linux ou Mac OS X, bem como sua versão (Windows 7 ou XP, por exemplo):

<http://www.codeblocks.org/downloads/26>

Baixe a opção `mingw-setup.exe` que inclui adicionalmente o compilador GCC / G++ e o depurador.
Não há segredos na instalação.

E pronto, você já está pronto para começar a programar em C com o curso online de C do site **C Progressivo**.

Seja bem-vindo à linguagem de Programação C, a linguagem mais usada do mundo.

Criando e compilando seu primeiro programa na Linguagem C

No [artigo passado](#) do curso C Progressivo você baixou e instalou a IDE Clode::Blocks, que é o programa necessário (mais recomendado e melhor que o Dev-C++) para iniciar seus programas na linguagem C.

Ao final desse artigo, você irá criar seu primeiro programa na linguagem C e será, oficialmente, um programador C :)

Como criar e compilar um programa em linguagem C

Passo 1: Inicie um novo arquivo

Dependendo da linguagem em que você instalou o Code::Blocks

Vá em: **File -> New -> Empty File**

Ou em: **Arquivo -> Novo -> Arquivo vazio**

Note que apareceu uma tela em branco, que é onde você vai digitar seu código.

Não digite nada ainda. Você até pode, mas é um erro fazer isso, o ideal é salvar o arquivo primeiro.

Vou explicar o motivo no passo 2.

Passo 2: Salve seu arquivo com a extensão `.c`

O Code::Blocks não serve somente para a linguagem C, serve para a linguagem C++ também.

Como você vai programar em C, seus arquivos devem ter a extensão `'.c'`.

Clique no disquete, símbolo universal de Salvar e escolha um nome e coloque a extensão `.c`, por exemplo: **programa1.c**

Após salvo, o Code::Blocks vai permitir indentar automaticamente o seu código, ou seja, vai organizar ele e escrever algumas coisas por você, além de mostrar algumas coisas com cores

diferentes, o que facilita e deixa o código mais organizado, **coisa que não aconteceria caso não tivesse salvo antes o programa** (você entenderá melhor isso no próximo passo).

Passo 3: Programando

Agora vamos programar! Ou seja, escrever o código!

Escreva EXATAMENTE isso:

```
#include <stdio.h>

int main(void)
{
    printf("Meu primeiro programa - C Progressivo!\n");
    return 0;
}
```

Passo 4: Compilando e Rodando

Note no canto superior esquerdo, os seguintes botões:



O primeiro é o 'Build', o segundo é o 'Run' e o terceiro faz os dois 'Build and Run'.

Clique no terceiro, que ele irá construir (compilar) e executar seu programa, ou aperte F9.

Você obterá a seguinte tela:

```
Meu primeiro programa - C Progressivo!
```

```
Process returned 0 (0x0)   execution time : 0.094 s
Press any key to continue.
```

E eis o seu programa na Linguagem C.

No próximo artigo explicarei, detalhadamente, o que significa cada parte do código que você escreveu e executou.

Passo 5: Caso tenha obtido erros

Caso não tenha conseguido rodar seu programa, provavelmente deve estar usando outra IDE que não seja o Code::Blocks, então experimente trocar a linha:

```
main()
```

Por

```
int main()
```

Ou

```
void main()
```

Se os erros persistirem, leia a mensagem de erro. Muito provavelmente você digitou algo errado.

Em C, 'main' é diferente de 'Main', ou seja, C é case *sensitive*.

Código comentado do nosso primeiro programa em C

No artigo passado nós [criamos e compilamos nosso primeiro programa em C](#) de nosso curso de programação. Porém, não explicamos exatamente o que aconteceu. Simplesmente mostramos o que escrever e o que fazer.

Em nosso curso você verá sempre, **sempre**, as coisas bem explicadas. Esse é o diferencial do curso online C progressivo.

Veja agora os comentários, linha por linha, do código C.

Como criar seu primeiro programa em C

No tutorial passado de nossa [apostila de C](#), ensinamos como criar, compilar e executar o seu primeiro programa em C.

E mostramos que seu código é:

```
#include <stdio.h>

int main(void)
{
    printf("Meu primeiro programa - C Progressivo!\n");

    return 0;
}
```


Pois bem, vamos agora explicar o que é isso tudo.

O que é e para que serve **#include <stdio.h>**

Como já dissemos aqui e na seção [Comece a Programar](#) do site [Programação Progressiva](#), mais especificamente sobre a [Linguagem C](#), ela é extremamente versátil e poderosa. O kernel - miolo do sistema - do Linux e do Windows (e da maioria dos sistemas operacionais) é feito, em sua maioria, com a linguagem C. Programas de alto rendimento, jogos e microcontroladores também podem funcionar sob linguagem C.

O C não está "preparado" ou "no ponto" para você fazer todo e qualquer tipo de aplicação, pois isso o deixaria extremamente lento e pesado, pois teria que incluir muitas funcionalidades. Quando for programar você vai dizer ao C o que vai fazer e que funções da linguagem C você vai precisar em sua aplicação. E isso é feito, em parte, por meio da importação de bibliotecas.

Por exemplo, existem bibliotecas gráficas (Allegro, OpenGL, etc), que são usadas para criar aplicações gráficas (como jogos ou programas com janelas, botões etc), existem bibliotecas para usar as funções do sistema operacional windows (biblioteca *windows.h*), existem bibliotecas para fazer cálculos matemáticos (*math.h*) e por aí vai.

Não existe um número específico de bibliotecas, e podemos, inclusive, criar as nossas. Então é óbvio que o C não vai carregar todas elas automaticamente. Por isso precisamos dizer ao compilador o que vamos usar.

Para fazer isso, usamos o **'#include'**, que é chamado de diretiva e em seguida escrevemos o nome da biblioteca. No caso, vamos usar a biblioteca **'stdio.h'**. Dentro dessa biblioteca existe um código em C. É como se você dissesse ao C: 'ei C, adicione o arquivo stdio.h no programa, vou precisar das funções contidas nele'.

O **std** de 'stdio' é de *standard*(padrão) e o **io** é de Input/Output (entrada e saída).

Por entrada, entenda o sistema receber dados, como você digitar algo.

Por saída, entenda como um resultado que o sistema te dá, como uma mensagem na tela. Existem diversos tipos de entrada e saída, mas essa biblioteca trata das entradas e saídas padrões, como o nome diz.

No nosso caso, vamos usar a [função printf como saída de dados](#).

O que é e para que serve **int main(void) { }**

main() é a função principal. Sempre que compilamos um código em C, seu início se dará por meio dessa função. Tudo começa a partir dela.

O código da função é tudo aquilo que fica entre as chaves: {}

Ou seja, seus códigos em C **sempre devem possuir a função main**.

Nesse nosso primeiro programa, ele simplesmente deve exibir uma mensagem na tela.

Obviamente, o comando para isso deve estar dentro da função `main()`, senão o código não será executado. Teste: coloque o `printf` em outro local e veja o Code Blocks apontar esse erro.

O que é e para que serve `printf("Meu primeiro programa - C Progressivo!\n");`

Print, em inglês, é imprimir. Se acostume com essa notação.

O que nossa função `printf()` faz é imprimir uma mensagem na tela. Essa mensagem ou texto, nós chamamos, em programação, de String. Note que:

String -> "Meu primeiro programa - C Progressivo\n"

Não é uma string -> Meu primeiro programa - C Progressivo\n

Se colocar uma frase sem as aspas duplas, obterá um erro, pois a função `printf` é feita para receber e exibir uma string. Se você não usar as aspas, não estará passando uma string para a `printf`, portanto seu programa irá mostrar um erro.

Note que só podemos usar a função `printf()` porque importamos a biblioteca `stdio.h`

Essa função está declarada lá nessa biblioteca. O nosso programa final, o executável, também utiliza o código da `printf()`, porém é inútil ficar repetindo código.

Ao invés disso, guardamos nossos códigos para que possamos reutilizar depois.

Essa é uma função das bibliotecas em C.

Mais adiante, em nosso curso online de C, aprenderemos mais sobre como criar nossas próprias funções, bibliotecas, a manusear strings e o `printf`.

\n: New line, adicionando uma linha em branco

Já o caractere '\n' é o *New Line*, ou seja, ele imprime uma linha.

Ou pula de linha. É como se tivéssemos pressionado enter no terminal de comando, pois faz o cursor saltar uma linha.

Aprenderemos mais sobre o \n nosso tutorial sobre a [função printf\(\)](#).

Experimente adicionar mais \n e ver o resultado!

```
#include <stdio.h>

int main(void)
{
    printf("Meu primeiro programa - C Progressivo!\n\n\n");

    return 0;
}
```

A função `printf()` e alguns caracteres especiais

Você aprendeu no artigo passado [como criar seu primeiro programa](#) na fantástica linguagem de programação C: um aplicativo que exibiu uma mensagem na tela.

Pra isso, você usou a função **printf**. Porém, essa função é cheia de recursos, e vamos ensinar mais alguns agora.

Vamos mostrar uma porção de códigos, mas não vamos mostrar o resultado. Para você aprender a programar nunca fique copiando e colando, vá lá e digite os códigos.

Só assim você aprenderá.

Caracteres Especiais

Você viu que para exibirmos uma mensagem, basta escrever o texto entre aspas duplas "".

Porém, como vamos ver nesse [tutorial de C](#), não é o bastante somente escrever as coisas entre aspas duplas, e que tudo vai sair exatamente como você planejou.

Teste, por exemplo, exibir uma mensagem com os seguintes caracteres: " ou \

O código ficaria assim:

```
#include <stdio.h>

int main(void)
{
    printf("Aspas duplas \"\n");
    printf("Barra: \"");

    return 0;
}
```

Como explicamos antes, o que aparece no console (telinha preta) é o que está entre aspas: "isso aqui aparece".

Se você colocar aspas duplas você terá aspas duplas, o C vai interpretar que é pra ser exibido o que estiver entre a primeira e a segunda aspas, que nosso caso é o texto "Aspas duplas :". E a terceira aspas?

A terceira não era pra estar lá, o \n é um erro, por isso gera um erro na hora de compilar.

Por que o C não exhibe esses caracteres?

Porque o nosso conceito de caracteres é diferente do conceito que os computadores têm. Para nós, o caractere " é aspas duplas. Para o C, é um caractere que delimita o que vai aparecer no console.

Ou seja, para o C, esses caracteres não fazem parte do alfabeto, não estão lá por motivos de leitura para humanos, são símbolos que representam outras coisas.

Então, como exibir esses caracteres especiais na forma de texto?

Usando o caractere de \

Quando queremos que o caractere de aspas duplas apareça no console, colocamos o \ antes:

\"

O C vai interpretar que, com esse símbolo \ antes, o caractere " deve ser exibido na tela.

Como exibir o caractere \

Coloque uma barra antes dele também.

Sim, vai ficar: \\

Quando o C encontra essas duas barras, ele entende que deve se exibir o caractere \

Nosso programa fica assim:

```
#include <stdio.h>

int main(void)
{
    printf("Aspas duplas \" \n");
    printf("Barra: \\");

    return 0;
}
```

Em suma, o que esse caractere faz é 'avisar' ao C que o próximo caractere, que vem logo após o \ terá um significado diferente.

Note que já usamos isso antes, para imprimir uma quebra de linha: \n

Você sabe que ao escrever \n o C não irá imprimir a barra nem o n, ele vai imprimir uma quebra de linha, ou um 'Enter'.

Sinais sonoros e outros caracteres especiais

Carriage return: \r

Esse caractere especial faz com que o cursor se mova para o início da linha:

```
#include <stdio.h>

int main(void)
{
    printf("Carriage return: \r");
    getchar();

    return 0;
}
```

O comando **getchar()** faz com que o programa espere que o usuário digite alguma tecla. Assim o programa só termina se você digitar algo. Usamos isso para mostrar que o *carriage return* faz com o cursor vá para o início da linha.

Tabulação horizontal (TAB): \t

```
#include <stdio.h>

int main(void)
{
    printf("Antes do \\t \\t Depois do \\t");

    return 0;
}
```

Sons: \7 e \a

Os leitores mais antigos, que jogaram video-games como o NES ou Atari certamente vão se lembrar desse aviso sonoro, que também era utilizado em sistemas operacionais antigos e jogos de computador feitos no terminal:

```
#include <stdio.h>

int main(void)
{
    printf("\7 \a");

    return 0;
}
```

Como comentar seus códigos em C - Comentários e Delimitadores



A medida que seus códigos na **linguagem C** forem aumentando, eles ficarão incrivelmente difíceis de serem entendidos por outra pessoa.

Sim, futuramente seu código será lido/alterado por outra pessoa, provavelmente você. Para facilitar esse processo, você pode fazer 'comentários' em seus programas de C, explicando o que cada trecho de código faz.

Além de serem importantes, são considerados uma boa prática de programação. Então, se você quer ser um bom programador C, deixe seus códigos comentados e bem explicados.

Comentando códigos em C - Usando //

Sempre que quiser comentar alguma linha de seu código C, inicie a linha com duas barras: //

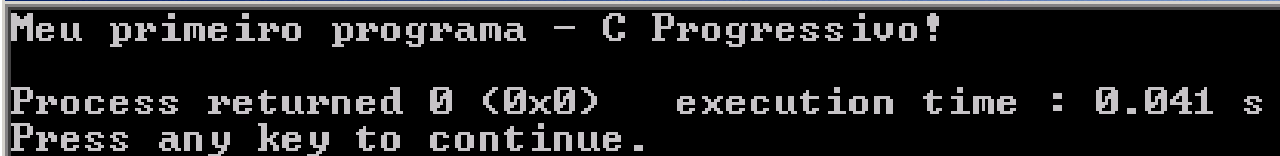
Não importa o que você escreva depois dessas duas linhas, elas não serão vistas. Ela é bastante usada antes de algum trecho de código C, explicando o que as próximas linhas fazem no programa, evitando assim que quem esteja lendo tenha que quebrar a cabeça para adivinhar o que o programador tentou fazer.

Veja o código exemplo:

```
#include <stdio.h>

int main()
{
    //O seguinte trecho mostra uma mensagem na tela
    //Essas linhas comentadas não irão aparecer na tela
    printf("Meu primeiro programa - C Progressivo!\n");
}
```

O resultado do código compilado é:



```
Meu primeiro programa - C Progressivo!
Process returned 0 (0x0)   execution time : 0.041 s
Press any key to continue.
```

No que as linhas que começam com //foram totalmente ignoradas. Você pode colocar uma ou milhão de linhas: na hora de compilar e linkar, o compilador C irá excluir todas as linhas começadas em // Logo, os comentários servem apenas para a leitura humana, não afetando em absolutamente nada o desempenho de seu aplicativo C.

Outro uso comum dos comentários é na criação de cabeçalhos de seus programas em C, como no exemplo abaixo:

```
#include <stdio.h>
// Código que exibe uma mensagem na tela
// Por Fulano de Tal
// Em 21/12/2112
// Para Curso C Progressivo
int main()
{
    //O seguinte trecho mostra uma mensagem na tela
    //Essas linhas comentadas não irão aparecer na tela
    printf("Meu primeiro programa - C Progressivo!\n");
}
```

Como usar os delimitadores ***/* */*** em linguagem de programação C

Imagine agora que você precisa fazer um comentário de mais de 20 linhas. Isso é bem comum entre estudantes que estão resolvendo alguma questão e colam o enunciado e ideia da solução no corpo do código.

E aí, vai escrever 20 vezes as duas barras `//` ?
Claro que não, use os delimitadores `/*` e `*/`.

Tudo o que você quer ver comentando coloca entre `/*` e `*/`, e tudo que será dentro será ignorado.

A diferença desses delimitadores para as duas barras `//`, é que as barras ignoram o que tem naquela linha, já os delimitadores `/**/` ignoram TUDO que estejam entre eles, seja uma letra, linha ou milhões de linhas.

Veja um exemplo que possui dezenas de linhas entre os delimitadores `/**/`, mas ao compilarmos e executarmos esse código em C, somente o *printf* é que faz realmente algo:

```
#include <stdio.h>

int main()
{
    /* Adoro estudar a linguagem de programação C enquanto escuto metal!
    Iron Maiden - Fear Of The Dark
    I am a man who walks alone
    And when I'm walking in a dark road
    At night or strolling through the park

    When the light begins to change
    I sometimes feel a little strange
    A little anxious when it's dark

    Fear of the dark, fear of the dark
    I have a constant fear that something's always near
    Fear of the dark, fear of the dark
    I have a phobia that someone's always there

    Have you run your fingers down the wall
    And have you felt your neck skin crawl
    When you're searching for the light?
    Sometimes when you're scared to take a look
    At the corner of the room
    You've sensed that something's watching you

    Fear of the dark, fear of the dark
    I have a constant fear that something's always near
    Fear of the dark, fear of the dark
    I have a phobia that someone's always there

    Have you ever been alone at night
    Thought you heard footsteps behind
    And turned around and no one's there?
    And as you quicken up your pace
```

```
You'll find it hard to look again  
Because you're sure that someone's there
```

```
Fear of the dark, fear of the dark  
I have a constant fear that something's always near  
Fear of the dark, fear of the dark  
I have a phobia that someone's always there
```

```
Watching horror films the night before  
Debating witches and folklores  
The unknown troubles on your mind  
Maybe your mind is playing tricks  
You sense, and suddenly eyes fix  
On dancing shadows from behind
```

```
(2x)  
Fear of the dark, fear of the dark  
I have constant fear that something's always near  
Fear of the dark, fear of the dark  
I have a phobia that someone's always there
```

```
When I'm walking in a dark road  
I am a man who walks alone */  
    printf("Colocamos a letra da música Fear of the Dark, da banda  
Iron Maiden - mas voce nao viu !)\n");  
}
```

Use comentários com moderação

Embora seus comentários não prejudiquem o funcionamento de seu programa em C, evite usar ele desnecessariamente, comentando *printf* e outras coisas óbvias: 'agora exibimos a mensagem', 'agora somamos os números' etc.

Use comentários para explicar o que é o programa, para que serve, qual a utilidade de um algoritmo que é mais complexo, elaborado ou grande.

O tipo de dado inteiro (int) na Linguagem C

Vamos estudar agora um dos tópicos mais básicos, bastante usado e importante na linguagem de programação C: os tipos de dados.

Para começar, vamos iniciar falando sobre os números inteiros em C, um tipo de dado muito importante, que usaremos durante toda nossa **apostila de C**.

Como declarar variáveis inteiras na linguagem C

Variáveis são os nomes que vamos dar a determinado bloco de memória.

Sempre que você quiser usar um dado (um número, um caractere, um texto, etc.) na linguagem C, você precisará declarar uma variável.

Ao fazer isso, você estará selecionando um espaço na memória (lembre-se que o C é uma linguagem poderosíssima e age no hardware de sua máquina).

Por exemplo, suponha que você queira armazenar sua pontuação em um jogo que você desenvolveu em C.

Se esse valor for inteiro, você precisará declarar uma variável (poderia usar o nome 'pontos', por exemplo), e o C irá alocar um espaço de 2 ou 4 bytes de memória. Aquele espaço em memória serve somente para sua variável, é o endereço dela.

Para declarar uma variável inteira fazemos:

```
int nome_de_sua_variavel;
```

Ao fazer isso, estamos reservando um espaço em memória para guardar um número. Em vez de lidarmos com o número da posição da memória, vamos usar seu nome, que é `nome_de_sua_variavel`;

Vale realçar que os tipos (no caso, usamos o tipo *int*), são sempre em letras minúsculas e vêm antes do nome da variável.

Como imprimir um número inteiro na tela, usando a linguagem C

Vamos mostrar agora como imprimir o valor de uma variável na **função printf**.

Dentro do printf, entre as aspas duplas, os números são representados por: %d, e após o fechamento das aspas, colocamos uma vírgula e o real valor da variável que é representada por %d.

Por exemplo, vamos supor que você queira fazer um programa que imprima na tela: "Eu tenho 20 anos", mas sem digitar o 20.

Ele ficaria assim:

```
#include <stdio.h>

int main()
{
    printf("Eu tenho %d anos", 18);
}
```

Podemos colocar vários números inteiros representados por %d dentro das aspas, e seus valores serão os que vem após a vírgula, na mesma ordem.

Por exemplo, um programa em C que imprima "Eu nasci em 2112, no dia 21 do mes 12":

```
#include <stdio.h>

int main()
{
    printf("Eu nasci no ano %d, no dia %d do mes %d", 2112, 21, 12);
}
```

Agora que já sabemos declarar um número e imprimir ele na tela, vamos fazer os dois. Vamos criar uma variável inteira chamada 'cprogressivo' e imprimir seu valor no printf:

```
#include <stdio.h>

int main()
{
    int cprogressivo;
    printf("O valor da variavel 'cprogressivo' eh %d", cprogressivo);
}
```

Aqui notamos duas coisas interessantes:

1. Digitamos 'cprogressivo' dentro do escopo da função printf, e o que aconteceu? Apareceu o nome 'cprogressivo', e não o valor dessa variável inteira. Como dissemos, para aparecer o valor dentro de um printf, usamos o %d.

2. Apareceu um número totalmente aleatório e louco (conhecido como lixo, em C), não foi? Por quê?

Ao declarar a variável 'cprogressivo', nós reservamos um espaço em memória. Dentro da printf nós imprimimos o seu conteúdo, e naquela posição da memória de seu computador havia aquele número lá armazenado (não me pergunte o porquê, mas saiba que para o computador é tudo número: textos, imagens, som, etc.).

No meu printf apareceu o número 62, e na sua máquina?

Como inicializar variáveis inteiras na linguagem C

Não nos serve para nada aqueles números aleatórios que você imprimiu com o último código. Variáveis só servem se pudermos controlar seu valor, e para isso temos que inicializá-las, ou seja, colocar um valor dentro dessa variável.

Para isso, usamos o símbolo de igualdade '='.

A inicialização pode ocorrer de duas formas:

1. Junto com a declaração da variável

```
int idade = 18;
```

2. Depois da declaração, em qualquer lugar do programa

```
int idade;
```

```
...
```

```
idade = 18;
```

Por exemplo, um programa que mostre na tela o salário que você deseja receber:

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

```
int main()
```

```
{
```

```
    int salario = 5;
    printf("Eu pretendo ganhar %d mil reais como programador C",
salario);
}
```

Note que você pode alterar o valor da variável 'salario', e esse valor sempre será impresso no printf.

Em breve vamos estudar em detalhes as operações matemáticas na Linguagem C, mas por hora vamos mostrar como é possível usar alguns cálculos com e nas variáveis inteiras, dentro ou não do printf.

Leia o seguinte programa e tente entender cada trecho do seguinte código, que está comentado.

Ele mostra que é possível declarar variáveis inteiras em várias partes do código, que podemos fazer operações matemáticas com as variáveis, dentre outras coisas.

```
#include <stdio.h>

int main()
{
    int soma = 1+1;
    printf("1 + 1 = %d \n", soma);

    int numero1 = 10;
    int numero2 = 20;

    soma = numero1 + numero2;

    printf("%d + %d = %d \n", numero1, numero2, soma);

    printf("%d - %d = %d\n", soma, numero1, soma-numero1);
}
```

Modificadores do tipo inteiro - short, long, signed e unsigned

Trabalhar com a linguagem de programação C é gratificante devido ao poder e liberdade que você tem, já que podemos atuar no 'talo' do sistema, inclusive selecionando espaços de memória.

Porém, isso tem um custo: estudo e cuidados adicionais em relação a grande maioria das outras linguagens de programação.

Por exemplo, o tamanho que um variável do tipo inteira (int) pode ocupar em diferentes computadores e sistemas. Falaremos mais detalhes sobre essas variações neste artigo de nossa **apostila de C**.

O tamanho que cada variável em C pode ocupar em memória

Vimos que uma variável do tipo int (inteira) em C, geralmente ocupa 2 ou 4 bytes na memória de seu computador.

Essa diferença, à priori, pode parecer insignificante. Mas é porque estamos no início de nossos

estudos na linguagem C, e por hora, nossos códigos e programas são bem simples e pequenos.

Mas a maior parte dos sistemas operacionais, como o Linux e o Windows, são feitos em C e aí essa diferença pode se tornar um problema, principalmente por questões de portabilidade. Ou seja, você pode compilar/rodar seu código em C no seu computador e obter um resultado, mas pode rodar em outro computador ou sistema (de diferentes configurações) e obter resultados totalmente diferentes (e erros, às vezes).

O grande diferencial da linguagem de programação Java é que ela não roda em sua máquina, e sim em uma máquina virtual.

Essa máquina virtual (JVM - Java Virtual Machine) é simulada em todos os dispositivos, portanto, o Java roda da mesma maneira em todos os sistemas (porém, geralmente suas aplicações são bem mais lentas que aquelas feitas em C ou C++, por exemplo).

Para saber mais sobre Java, oferecemos um curso completo de Java no site Java

Progressivo: <http://www.javaprogressivo.net/>

Se quiser saber quanto vale o valor do int, ou de qualquer outra variável, use a função 'sizeof(tipo)', e troque 'tipo' por um tipo de variável.

O exemplo a seguir mostra como descobrir o valor da variável int em C, usando a função sizeof:

```
#include <stdio.h>

int main()
{
    printf("O tamanho do inteiro em meu computador: %d bytes",
    sizeof(int));
}
```

Tendo um maior controle sobre o tamanho dos inteiros - **short e long**

Vamos apresentar agora dois modificadores do tipo inteiro em C: short e long, eles alteram o tamanho de bytes do inteiro.

A diferença entre os inteiros e modificadores está na faixa de valores armazenadas.

Um inteiro de 1 byte (char) armazena do número -128 a +127

Um inteiro de 2 bytes armazena do número -32 768 a +32 767

Um inteiro de 4 bytes armazena do número -2 147 483 648 a +2 147 483 647

Um inteiro de 8 bytes armazena do número -9 223 372 036 854 775 808 a +9 223 372 036 854 775 807

'short' em inglês, significa curto e 'long', longo.

Colocando uma dessas palavras antes da 'int', você definirá um tamanho e uma faixa de valores para suas variáveis.

Por exemplo, se criar a variável inteira 'numero' como short, deverá fazer:

```
short int numero;
```

De modo análogo para uma variável com o modificador 'long':

```
long int numero;
```

Para saber o tamanho de bytes de seu sistema, use novamente a função sizeof:

```
#include <stdio.h>

int main()
{
    printf("int : %d bytes\n", sizeof(int));
    printf("short int: %d bytes\n", sizeof(short));
    printf("long int: %d bytes\n", sizeof(long));
}
```

Controlando a faixa de valores dos inteiros por meio do sinal: **signed e unsigned**

Todos nós sabemos que os números, como os inteiros, podem assumir tanto valores negativos como positivos.

Porém, muitas vezes, valores negativos (ou positivos) podem ser inúteis, chegando até a atrapalhar em termos de computação.

Por exemplo, o tamanho de memória é sempre positivo, não existe um bloco de -2 bytes em sua máquina.

Então, para declarar que um número seja apenas positivo (incluindo o 0), usamos o modificador unsigned:

```
unsigned int teste;
```

Analogamente para especificar que o inteiro possui valores positivos e negativos:

```
signed int teste;
```

Em C, por padrão, ao declararmos uma variável do tipo int, ele será automaticamente do tipo signed.

Portanto, a declaração passada é inútil, serve apenas para fins didáticos.

Caso queiramos apenas números positivos, a faixa de valores negativos que apresentamos é inútil e podemos desconsiderar ela, aumentando a faixa de valores positivos.

Então, ao declararmos inteiros com o modificador unsigned, as faixas de valores passam a ser:

Um inteiro de 1 byte (char) armazena do número 0 a +255

Um inteiro de 2 bytes armazena do número 0 a +65 535

Um inteiro de 4 bytes armazena do número 0 a +4 294 967 295

Um inteiro de 8 bytes armazena do número 0 a +18 446 744 073 709 551 615

Quando usar short, long, signed e unsigned

O curso C Progressivo visa ensinar o básico, então, não será necessário usarmos esses modificadores ao longo de nossa apostila online.

No futuro, quando você for um programador profissional, será um ótimo e diferenciado costume usar esses modificadores.

Quando usar o short int em C

Você deverá usar o short quando quiser armazenar, sempre, valores pequenos.

Por exemplo, suponha que você vá fazer um banco de dados para o Governo Federal, onde terá que criar milhões de cadastros.

Para armazenar idades, usar short int. Ora, valores inteiros de idades são pequenos e raramente passam do 100.

Então não desperdice memória à toa, use short!

Quando usar o long int em C

O long é bastante usado para cálculos de cunho acadêmico, como científico e estatístico.

É comum também usarmos o long int para armazenar números como RG e CPF, que são compostos de vários dígitos.

Quando usar unsigned int em C

Você pode usar o unsigned para se certificar que essa variável inteira nunca irá receber um valor negativo, como para armazenar dados de memória, idade, o life de um jogo etc.

Para saber mais sobre esse limites e padrões da linguagem C, acesse:

<http://en.wikipedia.org/wiki/Limits.h>

Exercício:

Crie um programa em C que mostre o tamanho das seguintes variáveis em seu computador:

- int
- short int
- long int
- signed int
- unsigned int
- short signed int
- short unsigned int
- long signed int
- long unsigned int

Os tipos float e double - números decimais (ou reais) em C

No artigo passado de nosso curso de C, estudamos sobre o [tipo inteiro \(int\)](#), como declarar, imprimir e inicializar tal tipo de dado.

Agora faremos o mesmo, mas para os números decimais, também conhecidos como números reais, que são os tipos float e double.

O que são e para que servem os tipos **float** e **double**

Em nosso dia-a-dia, muitas vezes usamos dados em forma de números decimais, como por exemplo:

o rendimento da poupança (0,57%, 1,01%),
valores monetários (R\$ 1,99) ,
as notas da faculdade (10,0 , 5,4 , 0,5),
constantes e outros números matemáticos ($\pi = 3,14...$) etc.

Não podemos, porém, armazenar essas informações em variáveis inteiras na linguagem C. Ao invés disso, precisamos declarar usando os tipos de dados float e double, que são tipos especialmente feitos para que possamos trabalhar com números reais (decimais).

Vamos aprender como usar tais tipos de variáveis neste artigo de nossa apostila de C.

Como declarar e inicializar variáveis do tipo float e double na linguagem C

Lembre-se que, para inteiros, declaramos da seguinte maneira: `int idade`, `int mes` etc.

Analogamente, para floats e double:

```
float pi;  
float juros;  
double tamanho_de_uma_bacteria;  
double area_de_uma_circunferencia;
```

A declaração também não é diferente da que fizemos com inteiros.

Podemos inicializar valores tanto na declaração das variáveis como somente depois:

```
float pi = 3.14;  
double juros = 1.32101;
```

Ou

```
float pi;  
double juros;  
pi = 3.14;
```



```
juros = 1.32101;
```

Se você é novo no mundo da programação e nunca teve contato com uma língua estrangeira, certamente achará tais inicializações de variáveis estranha. Mas esse é uma regra bem importante:

Na linguagem de programação C, usamos o PONTO (.), e não a vírgula para separar a parte inteira da decimal.

Ou seja, no Brasil escrevemos: 1,99 e 0,57
Em programação é: 1.99 e 0.57

Qual a diferença entre float e double

Em nossos exemplos, usamos tanto float como double para representar números reais. Temos, então, duas opções iguais para representar esses números decimais? Na verdade não, há uma diferença.

Variáveis float exigem, geralmente, 4 bytes de memória para serem armazenadas enquanto double necessitam de 8 bytes.

Essa diferença serve para termos uma melhor precisão na hora de realizar cálculos.

O número PI, por exemplo, é irracional. Ou seja, ela possui uma quantidade INFINITA de casas decimais.

Obviamente, um cálculo com o uso do pi nunca é totalmente preciso. Além do mais, computadores tem uma quantidade de memória limitada.

Então, nos seus trabalhos escolares você deve declarar e usar uma variável do tipo float para representar o número pi:

```
float pi = 3.14;
```

Se quiser ser mais preciso pode fazer até: `pi = 3.1415;`

Já um Engenheiro Civil ou um Físico da NASA terá que usar uma precisão maior, pois quanto mais casas decimais, mais correto

será seu resultado. Então, eles usariam:

```
double pi = 3,14159265358979323
```

Ok, agora você sabe a diferença entre um float e um double - apenas a precisão.

Mas qual a diferença entre 0 e 0.0? E a diferença de 1 e 1.00?

Você sabe que um inteiro ocupa 2 bytes na memória, e que um float ocupa 4 bytes.

Além do tamanho alocado em sua máquina, qual outra diferença que faz esses valores diferentes?

Sim, o ponto. Ou seja, a parte decimal.

Fazer: `int erro = 0`

É totalmente diferente de: `float erro = 0.0;`

Uma vez declarado um inteiro, não poderá usar decimais nele. Mesmo sabendo que `0 = 0.0`

Já os decimais podem ser trabalhados com inteiros.

Por exemplo:

```
double erro = 0.00
int juros = 1
```

Podemos fazer: $juros + erro = 1.00$

Ou seja, quando fazemos uma operação matemática de um decimal com inteiro, obteremos sempre um decimal.

Assim, o resultado dessa operação deverá sempre ser armazenado em um float ou em um double.

Veremos mais sobre isso quando estudarmos operações matemáticas na linguagem C.

Imprimindo números reais float e double na tela por meio do printf

Vimos na aula passada que representamos inteiros como %d dentro das aspas, de um printf.

Para variáveis decimais ou reais, como o float e o double usamos: %f

Vejamos um exemplo que mostra um valor de pi com precisão simples (float) e outro com precisão dupla(double):

```
#include <stdio.h>

int main()
{
    float pi = 3.14;
    double piDouble = 3.1415926535897932384626433832795;
    printf("Valor de pi %f\n", pi );
    printf("Valor de pi mais preciso %f\n", piDouble );
}
```

Aqui notamos uma coisa curiosa no segundo valor, é exibido: 3.141593

Ou seja, o C não mostrou todo o valor da variável double 'piDouble' e ainda arredondou!

Podemos resolver isso da seguinte maneira. Supondo que você queira que seja exibido 6 casas decimais:

Ao invés de usar '%f' coloque: '%.7f'

Ou seja, esse 0.7f diz ao C o seguinte "Após o ponto, exiba 7 casas decimais".

Teste e veja o resultado:

```
#include <stdio.h>

int main()
{
    float pi = 3.14;
    double piDouble = 3.1415926535897932384626433832795;
    printf("Valor de pi %f\n", pi );
    printf("Valor de pi mais preciso %.7f\n", piDouble );
}
```

Será exibido: 3.1415927

Agora veja o seguinte: a variável 'pi' tem somente duas casas decimais depois do ponto.

O que ocorre se eu ordenar ao printf que imprima com 5 casas decimais?
Programa e veja você o que acontece:

```
#include <stdio.h>

int main()
{
    float pi = 3.14;
    double piDouble = 3.1415926535897932384626433832795;
    printf("Valor de pi %.5f\n", pi );
    printf("Valor de pi mais preciso %.7f\n", piDouble );
}
```

Como imprimir números na forma exponencial em C

Outra maneira de imprimir variáveis decimais é usando exponenciais.

Podemos inicializar uma variável da seguinte maneira:

`float numero = xEy;`

Isso significa: x vezes 10 elevado a y = $x * 10^y$

Ou seja, $1E6 = 1 \text{ vezes } 10^6 = 1 \text{ milhão}$

E `float numero = xE-y`

Significa: x vezes 10 elevado a -y = $x * 10^{(-y)}$

Por exemplo: $2E-3 = 2 \text{ vezes } 10^{(-3)} = 0.002$

Veja o seguinte código e tente adivinhar sua saída. Logo após, rode o programa para ver se acertou:

```
#include <stdio.h>

int main()
{
    float salarioSonho = 1E6,
          salarioReal = 10E-3;
    printf("Sonhei que meu salario era de R$%.2f, \nmas acordei e lembrei que era %.2f centavos", salarioSonho, salarioReal);
}
```

Nesse último exemplo, note como declaramos mais de uma variável.

Em vez de fazer:

`float variavel1;`

`float variavel2;`

`float variavel3.`

Você pode fazer:

`float variavel1, variavel2, variavel3;`

Ou, para ficar mais legível:

`float variavel1,`

```
variavel2,  
variavel3.
```

O tipo char - escrevendo na linguagem C

Agora que você já sabe como lidar com inteiros e decimais na linguagem C, está na hora de estudarmos como escrever caracteres.

Como declarar o tipo char em C

Para armazenar caracteres vamos usar um tipo especial de dados, o char (de *character* - caractere, em inglês).

O tipo char serve para armazenar UM, e somente UM, caractere.

Para declarar, usamos a seguinte sintaxe;

```
char nomeDaVariavel;
```

Ao fazermos isso, estamos alocando 1 byte de memória para guardar nosso caractere.

Se você quiser armazenar mais caracteres, temos que usar as Strings, que são um conjunto de caracteres, usados para escrever textos maiores.

Nós estudaremos as Strings mais à frente, em nossa apostila de C.

Como inicializar variáveis char em C

Para guardar uma letra no seu char, temos que fazer uma operação especial: sempre colocar o caractere entre aspas simples.

Por exemplo, para guardar a letra C, escrevemos:

```
char letra = 'C';
```

A sintaxe linguagem de programação é case sensitive, ou seja, minúsculo é diferente de maiúsculo.

Por exemplo:

```
char letraMinuscula = 'c';  
char letraMaiuscula = 'C';
```

Esses dois caracteres, embora representem a mesma letra, são totalmente diferentes, pois uma é maiúscula e a outra é minúscula.

Como imprimir caracteres e textos na tela

Vamos agora aprender como mostrar na tela letras e caracteres, por meio da função printf.

Assim como usamos %d para mostrar inteiros, %f para float e double, vamos usar um símbolo especial para caracteres: %c

Por exemplo, um programa que exiba o texto "C Progressivo" na tela tem o seguinte código:

```
#include <stdio.h>

int main()
{
    char letra0='C', letra1=' ', letra2='P', letra3='r', letra4='o',
    letra5='g',
    letra6='r', letra7='e', letra8='s', letra9='i',
    letra10='v', letra11='o', letra12='\n';

    printf("%c%c%c%c%c%c%c%c%c%c%c%c%c", letra0, letra1, letra2, letra3, letra4,
    letra5, letra6, letra7, letra8, letra8, letra9, letra10, letra11, letra12);
}
```

Note 3 coisas interessante que mostramos:

1 - letra1=' '

Sim, espaço também é caractere!

2 - Repetimos letra8 duas vezes no printf, por quê?

Ora, a letra 's' se repete duas vezes na frase: "C Progressivo".

Em vez de gastar memória à toa para representar o mesmo caractere, o certo é repetir duas vezes o char.

3 - letra12='\n'

Sim, a quebra de linha, ou ENTER, é um caractere também!

Notou também a trabalheira e o tanto de código, apenas pra escrever isso? Inviável!!

Em breve, em nosso curso C Progressivo, iremos aprender sobre strings, ou vetores de caracteres, que é uma maneira bem mais simples de se escrever e manipular textos em C.

A tabela ASCII em C

Digite e rode o seguinte programa em C:

```
#include <stdio.h>

int main()
{
    char ascii = 67;
```

```
printf("%c%", ascii);
}
```

Estranho o resultado, não? Inicializamos um caractere sem aspas simples e digitamos dois números?

Se notar bem, não inicializamos o char com um caractere, pois não usamos aspas simples, iniciamos como se char fosse um inteiro.

Por que isso?

Ver char como inteiros não seria errado.

O que acontece é que cada caractere em programação recebe um número, como se fosse uma identificação.

A letra 'C' por exemplo, tem o número 67 como sua identificação.

Ou seja, para o computador:

char ascii = 67;

ou char ascii = 0x43;

ou char ascii = 'C';

É a mesma coisa. No primeiro caso, representamos o caractere como um número decimal.

No segundo, o caractere é representado para o um número no formato hexadecimal.

Altere o número para outro e veja os resultados.

Você verá que cada caractere, incluindo o ENTER, TAB ou aviso sonoro é identificado com um número.

A tabela ASCII nada mais é que uma lista completa dos números que identificam os caracteres, veja só:

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040	€#32;	Space	64	40	100	€#64;	@	96	60	140	€#96;	`
1	1	001	SOH (start of heading)	33	21	041	€#33;	!	65	41	101	€#65;	A	97	61	141	€#97;	a
2	2	002	STX (start of text)	34	22	042	€#34;	"	66	42	102	€#66;	B	98	62	142	€#98;	b
3	3	003	ETX (end of text)	35	23	043	€#35;	#	67	43	103	€#67;	C	99	63	143	€#99;	c
4	4	004	EOT (end of transmission)	36	24	044	€#36;	\$	68	44	104	€#68;	D	100	64	144	€#100;	d
5	5	005	ENQ (enquiry)	37	25	045	€#37;	%	69	45	105	€#69;	E	101	65	145	€#101;	e
6	6	006	ACK (acknowledge)	38	26	046	€#38;	&	70	46	106	€#70;	F	102	66	146	€#102;	f
7	7	007	BEL (bell)	39	27	047	€#39;	'	71	47	107	€#71;	G	103	67	147	€#103;	g
8	8	010	BS (backspace)	40	28	050	€#40;	(72	48	110	€#72;	H	104	68	150	€#104;	h
9	9	011	TAB (horizontal tab)	41	29	051	€#41;)	73	49	111	€#73;	I	105	69	151	€#105;	i
10	A	012	LF (NL line feed, new line)	42	2A	052	€#42;	*	74	4A	112	€#74;	J	106	6A	152	€#106;	j
11	B	013	VT (vertical tab)	43	2B	053	€#43;	+	75	4B	113	€#75;	K	107	6B	153	€#107;	k
12	C	014	FF (NP form feed, new page)	44	2C	054	€#44;	,	76	4C	114	€#76;	L	108	6C	154	€#108;	l
13	D	015	CR (carriage return)	45	2D	055	€#45;	-	77	4D	115	€#77;	M	109	6D	155	€#109;	m
14	E	016	SO (shift out)	46	2E	056	€#46;	.	78	4E	116	€#78;	N	110	6E	156	€#110;	n
15	F	017	SI (shift in)	47	2F	057	€#47;	/	79	4F	117	€#79;	O	111	6F	157	€#111;	o
16	10	020	DLE (data link escape)	48	30	060	€#48;	0	80	50	120	€#80;	P	112	70	160	€#112;	p
17	11	021	DC1 (device control 1)	49	31	061	€#49;	1	81	51	121	€#81;	Q	113	71	161	€#113;	q
18	12	022	DC2 (device control 2)	50	32	062	€#50;	2	82	52	122	€#82;	R	114	72	162	€#114;	r
19	13	023	DC3 (device control 3)	51	33	063	€#51;	3	83	53	123	€#83;	S	115	73	163	€#115;	s
20	14	024	DC4 (device control 4)	52	34	064	€#52;	4	84	54	124	€#84;	T	116	74	164	€#116;	t
21	15	025	NAK (negative acknowledge)	53	35	065	€#53;	5	85	55	125	€#85;	U	117	75	165	€#117;	u
22	16	026	SYN (synchronous idle)	54	36	066	€#54;	6	86	56	126	€#86;	V	118	76	166	€#118;	v
23	17	027	ETB (end of trans. block)	55	37	067	€#55;	7	87	57	127	€#87;	W	119	77	167	€#119;	w
24	18	030	CAN (cancel)	56	38	070	€#56;	8	88	58	130	€#88;	X	120	78	170	€#120;	x
25	19	031	EM (end of medium)	57	39	071	€#57;	9	89	59	131	€#89;	Y	121	79	171	€#121;	y
26	1A	032	SUB (substitute)	58	3A	072	€#58;	:	90	5A	132	€#90;	Z	122	7A	172	€#122;	z
27	1B	033	ESC (escape)	59	3B	073	€#59;	;	91	5B	133	€#91;	[123	7B	173	€#123;	{
28	1C	034	FS (file separator)	60	3C	074	€#60;	<	92	5C	134	€#92;	\	124	7C	174	€#124;	
29	1D	035	GS (group separator)	61	3D	075	€#61;	=	93	5D	135	€#93;]	125	7D	175	€#125;	}
30	1E	036	RS (record separator)	62	3E	076	€#62;	>	94	5E	136	€#94;	^	126	7E	176	€#126;	~
31	1F	037	US (unit separator)	63	3F	077	€#63;	?	95	5F	137	€#95;	_	127	7F	177	€#127;	DEL

Source: www.LookupTables.com

Então podemos escrever 'oi' da seguinte maneira:

Sim: 111,105 quer dizer 'oi'. Legal, não?

Por exemplo, descubra o que está dito no programa abaixo:

Exercício:

A função scanf - recebendo números do usuário

O programa simplesmente roda e mostra coisas na tela.

Porém, na vida real não é assim. Praticamente todos possuem algum tipo de interação com o usuário: recebem dados, cliques, arrastamos e soltamos etc.

Nessa lição iremos aprender a usar a função scanf e obter dados do usuário. Vamos começar a 'conversar' com o computador.

Recebendo números inteiros com a função scanf

Assim como fizemos para trabalhar com inteiros na [função printf](#), vamos usar novamente o símbolo %d para representar os int.

Suponho que queiramos pedir um inteiro ao usuário, primeiro temos que declarar um inteiro. Vamos supor um de nome 'numero'.

Para o usuário armazenar um número nessa variável 'numero', usamos a seguinte sintaxe:

```
scanf("%d", &numero);
```

Essa função nos diz "armazene na variável 'numero' um inteiro". O erro mais comum é esquecer o &, cuidado!

Por exemplo, vamos escrever um programa em C que pede um número ao usuário e o mostra na tela:

```
#include <stdio.h>

int main()
{
    int numero;
    printf("Digite um numero: ");
    scanf("%d", &numero);

    printf("O numero digitado foi: %d", numero);
}
```

Exemplo de código- Como usar a função scanf()

Escreva um programa em C que peça dois números inteiros e mostre sua soma.

Primeiro criamos três variáveis inteiras: num1, num2 e resultado.

Essas variáveis que vão armazenar os números que o usuário fornecer e o resultado da soma.

Após isso, usamos a função scanf() para pegar do usuário os dois número.

Em seguida, armazenamos a soma desses números na variável 'resultado', e exibimos essa variável num printf.

Veja como ficou o nosso código em C:

```
#include <stdio.h>

int main()
{
    int num1, num2, resultado;
    printf("Digite um numero: ");
    scanf("%d", &num1);
```

```
printf("Digite outro numero: ");
scanf("%d", &num2);

resultado = num1 + num2;

printf("%d + %d = %d", num1, num2, resultado);
}
```

Recebendo mais de um número dentro de uma única scanf

Para evitar o trabalho de ter que escrever a scanf cada vez que você deseje receber um número do usuário, você pode colocar mais de um %d dentro do escopo da scanf.

```
#include <stdio.h>

int main()
{
    int num1, num2;

    printf("Insira dois numeros: ");
    scanf("%d %d", &num1, &num2);

    printf("Você digitou: '%d' e '%d'", num1, num2);
}
```

No caso acima, num1 vai ser o número que você digitou antes de dar enter, espaço ou tab. E num2 será o número que você digitou após dar enter, espaço ou tab

Exercício: Faça um programa que peça dois inteiros ao usuário e que mostre a diferença (subtração) entre o primeiro e segundo número.

Recebendo números reais ou decimais com a função scanf

Para receber números do tipo float ou double, fazemos exatamente como na seção anterior, sobre inteiros, somente com uma diferença que talvez você já saiba qual é: usamos %f ao invés de %d.

Faz todo sentido, não?

Exemplo: Crie um programa em C que peça dois números decimais ao usuário e mostre

o produto deles, com precisão de dois números.

Lembrando que o símbolo de multiplicação é o asterisco, *:

```
#include <stdio.h>

int main()
{
    float num1, num2, resultado;
    printf("Digite um numero: ");
    scanf("%f", &num1);

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

    resultado = num1 * num2;

    printf("%.1f + %.1f = %.2f", num1, num2, resultado);
}
```

Exercício: Faça um programa em C que peça dois números do tipo double ao usuário e mostre o resultado da divisão do primeiro pelo segundo e exiba esse resultado com 3 casas decimais.

PS: o símbolo de divisão é /

Recebendo letras do usuário - As funções scanf, getchar, fgetc egetc

No artigo passado de nosso curso de C, aprendemos [como usar a função scanf](#) para receber dados do usuário, e com isso fazer programas bem mais dinâmicos.

Porém, lá tratamos só de números (tanto inteiros, como decimais). Mas, muitas vezes, nós precisamos digitar textos em programas.

Por isso vamos aprender como receber letras do usuário.

Vamos explicar o que são e como funcionam as funções scanf, getchar e fflush.

Recebendo caracteres em C através da função **scanf()**

A maneira é idêntica a que fizemos quando usamos a função scanf() para capturar números digitados pelo usuário, que como ensinado no artigo passado de nossa apostila.

Mas com os caracteres, ao invés de usar %d (para números inteiros) ou %f (para números floats), vamos usar %c.

Pois é assim que representamos variáveis do tipo char em C.

Portanto, um programa que pede um caractere ao usuário e o imprime na tela é feito da seguinte maneira:

```
#include <stdio.h>

int main()
{
    char letra;

    printf("Insira um caractere: ");
    scanf("%c",&letra);
    printf("Você digitou: '%c'", letra);
}
```

Recebendo caracteres em C por meio da função **getchar()**

A função scanf() é bem poderosa e flexível.

Com ela, podemos pegar uma infinidade de dados do usuário, e inclusive escolher o que vai ser 'capturado', limitar o tanto de coisas que pode ser escrito, e uma série de outras funcionalidades que vocês irão aprender durante nosso curso de C.

Porém, há mais opções que facilitam nossa vida.

Existe uma função que faz o mesmo papel da scanf e é voltada para o uso com caracteres, é a getchar().

Ela é mais simples, pois não precisar usar %c ou &, como fazemos na scanf(), e foi feito especialmente para ser usado com caracteres.

get -> pegar

char -> caractere

Para usar, fazemos:

```
seu_caractere = getchar();
```

Veja como é seu uso em um código que pede um caractere para o usuário, armazena no char 'letra', e em seguida exibe esse mesmo caractere.

```
#include <stdio.h>

int main()
{
    char letra;

    printf("Insira um caractere: ");
    letra = getchar();
    printf("Você digitou: '%c'", letra);
}
```

Recebendo caracteres em C por meio das funções fgetc e getc

Essas funções também servem para armazenar caracteres, porém são mais gerais que a `getchar()`, pois podem receber dados de outras fontes, além da padrão (o teclado - `stdin`).

Seus usos são semelhantes ao `getchar()`, porém temos que dizer ao C por onde o programa deve receber as informações.

No nosso caso, e pelo teclado, então fica: `fgetc(stdin)` e `getc(stdin)`

Por exemplo, um aplicativo em C que recebe duas letras e mostra na tela seria:

```
#include <stdio.h>

int main()
{
    char letra1, letra2;

    printf("Insira um caractere: ");
    letra1 = getc(stdin);

    printf("Insira outro caractere: ");
    letra2 = getc(stdin);

    printf("Você digitou: '%c' e '%c'", letra1, letra2);
}
```

É importante que você faça esse exemplo e veja o resultado. Agora teste com as outras funções: `scanf` e `getchar`
Estranho esse problema, não?

Isso tem a ver com uma região especial de memória, uma espécie de memória temporária chamada `buffer`, que vamos estudar mais no [próximo artigo](#) do [curso C Progressivo](#).

Buffer: o que é, como limpar e as funções `fflush` e `__fpurge`

No [artigo passado](#) foi pedido o seguinte:

Fazer um programa em C que peça dois caracteres ao usuário e os exiba.

Porém, há um problema ao se fazer isso, que é o que vai ser explicado nesse artigo de nossa **apostila de C**.

O problema de usar scanf, getchar, getc e fgetc para receber caracteres em C

"Ora, é só declarar duas variáveis char e usar a scanf duas vezes, uma para cada variável", é que você deve ter pensado para criar o tal programa.

Vamos fazer isso então, e ver o resultado:

```
#include <stdio.h>

int main()
{
    char letra1, letra2;

    printf("Insira um caractere: ");
    scanf("%c", &letra1);

    printf("Insira outro caractere: ");
    scanf("%c", &letra2);

    printf("Você digitou: '%c' e '%c'", letra1, letra2);
}
```

Eu digitei 'C' (a melhor letra do alfabeto), dei enter, e antes de digitar a próxima letra o programa terminou, exibindo a seguinte mensagem:

```
"Você digitou: 'C' e '
"
```

Nossa! Estranho, não? Será que hackeamos a linguagem C e descobrimos uma falha? Não ;)

Note que digitei 'C' e enter...mas enter também é uma tecla, e é representada por '\n', lembra? Ou seja, o C entendeu que nossa segunda letra era o enter!

A solução para isso é bem simples.

Na função scanf, dê um espaço entre a aspa " e o símbolo %c.

Nosso código fica assim:

```
#include <stdio.h>

int main()
{
    char letra1, letra2;

    printf("Insira um caractere: ");
    scanf("%c", &letra1);

    printf("Insira outro caractere: ");
    scanf(" %c", &letra2);

    printf("Você digitou: '%c' e '%c'", letra1, letra2);
}
```

Pronto! Agora funciona perfeitamente! Pois esse simples espaço é um comando para o C desconsiderar o enter, tab ou espaço em branco.

Ok, mas só é possível fazer isso na scanf().
E na getchar(), getc() e fgetc()?

Limpando o buffer em C: fflush e __fpurge

Ainda no primeiro exemplo desse artigo (o que dá problema), digitamos a letra 'C', que é armazenada na variável 'letra1' e em seguida apertamos enter.

Esse caractere (enter), ficará armazenado no buffer do teclado (um memória temporária).

Em seguida, nosso programa em C pede para que algo seja armazenado na variável 'letra2'.

Porém, antes do C receber um novo dado do usuário, ele checa se não tem mais alguma coisa armazenada no teclado (ele sempre faz isso...fez antes, para pegar a letra 'C'). E lá tem um caractere sim, o enter.

Então o programa pega esse caractere e o coloca na variável letra2, e é por isso que aparece uma quebra de linha em nosso programa.

Portanto, uma alternativa, caso não queria usar o espaço entre " e o %c na scanf, é **limpar o buffer** após cada scanf(), getchar(), getc() ou fgetc().

Para limpar o buffer em Windows, use: **fflush(stdin)**

Para limpar o buffer em Linux, use: **__fpurge(stdin)**

Veja como fica nosso programa original, funcionando do jeito que queríamos:

```
#include <stdio.h>

int main()
{
    char letra1, letra2;

    printf("Insira um caractere: ");
    scanf("%c", &letra1);

    fflush(stdin);
    __fpurge(stdin);
```



```
printf("Insira outro caractere: ");  
scanf("%c", &letra2);  
  
printf("Você digitou: '%c' e '%c'", letra1, letra2);  
}
```

Fica ao seu dispor escolher como vai ser.

E se habitue com essas coisas, em C há várias maneiras de se fazer várias coisas. Muitas vezes, porém, a solução que usamos não é muito segura ou portátil, mas é a mais simples.

Limpar o buffer, por exemplo, nem sempre é algo desejável, e para programação mais profissional e segura não é recomendado que se use **fflush** por exemplo.

Mas para quem está começando, não há problema algum ficar limpando o buffer após cada scanf, e o `scanf` (embora seja arriscado e não indicado em alguns casos) é o mais usado.

Por isso, usaremos bastante o `scanf` ao longo de nosso [curso de C](#).