
C Starter
www.t2ti.com

Apresentação

O Curso C Starter foi projetado com o objetivo de ajudar àquelas pessoas que têm uma base de lógica de programação e desejam aprender a linguagem C.

A estrutura do curso é formada por módulos em PDF e por mini-cursos em vídeo. O aluno deve baixar esse material e estudá-lo. Deve realizar os exercícios propostos. Todas as dúvidas devem ser enviadas para a lista de discussão que está disponível para inscrição na página do Curso C Starter no site www.t2ti.com. As dúvidas serão respondidas pelos instrutores Albert Eije, Cláudio de Barros e Miguel Kojiio, além dos demais participantes da lista.

Nosso objetivo é que após o estudo do Curso C Starter o aluno tenha as noções fundamentais da linguagem C e consiga, a partir deste momento, aprofundar-se no assunto de forma autônoma e de acordo com as suas expectativas.

Albert Eije trabalha com informática desde 1993. Durante esse período já trabalhou com várias linguagens de programação: Clipper, PHP, Delphi, C, Java, etc. Atualmente mantém o site www.alberteije.com e trabalha como analista de sistemas do Banco do Brasil.

Cláudio de Barros é Tecnólogo em Processamento de Dados e analista de sistemas do Banco do Brasil.

Miguel Kojiio é bacharel em Sistemas de Informação, profissional certificado Java e também trabalha como analista de sistemas do Banco do Brasil.

Módulo 01

Conhecendo a Linguagem C Parte I

A Linguagem C

A linguagem C foi criada em 1972 por Dennis M. Ritchie e Ken Thompson no laboratório Bell. Ela foi baseada na linguagem B que já era uma evolução da linguagem BCPL. É estruturada, imperativa, procedural, de baixo nível e padronizada.

Criadores da linguagem C – Ken Thompson e Dennis Ritchie

Características

Pela definição C é uma linguagem de baixo nível, no entanto a mesma alia características de linguagens de alto nível (como Pascal) e outras de baixo nível como assembly. Isso significa que a linguagem C junta flexibilidade, praticidade e poder de manipulação da máquina diretamente. Dessa forma não tem as limitações que tem o Pascal. C permite liberdade total ao programador e este é responsável por tudo que acontece.

C é uma linguagem estruturada, ou seja, estrutura o programa em blocos para resolver os problemas. A filosofia básica de uma linguagem estruturada é dividir para trabalhar, você divide um problema em pequenas partes que sejam possíveis de serem feitas. Neste contexto C é igual a Pascal.

C é uma linguagem muito poderosa que é utilizada para criar programas diversos como processadores de texto, planilhas eletrônicas, sistemas operacionais, programas de comunicação, programas para a automação industrial, gerenciadores de bancos de dados, programas de projeto assistido por computador, programas para a solução de problemas da Engenharia, Física, Química e outras ciências.

Compiladores e Interpretadores

Para que um programa seja executado em um computador o mesmo precisa estar em linguagem de máquina (zeros e uns). Quando digitamos nosso programa, seja em C, Pascal, Java, etc o mesmo ainda não está na linguagem de máquina. Para isso existem os interpretadores e os compiladores. Ambos convertem um programa de uma linguagem qualquer para a linguagem de máquina.

No caso do C existem tanto interpretadores quanto compiladores. Mas, qual seria a diferença? O interpretador lê as instruções do programa linha a linha e vai checando os erros e convertendo para a linguagem de máquina. Após a conversão a linha é executada. Se a linguagem for apenas interpretada o interpretador sempre deverá estar presente para que o programa funcione. Já o compilador lê o programa linha a linha, checa se existem erros e converte a linha para a linguagem de máquina. Mas o compilador não executa a linha. Ele converte todas as linhas para a linguagem de máquina e ao chegar ao final do arquivo do programa sem nenhum erro ele gera um arquivo com a extensão **.OBJ**. Esse arquivo não pode ser executado. Nessa ocasião entra em cena um programa linkeditor que vai agregar algumas rotinas ao arquivo **.OBJ**. Após isso será gerado um arquivo **.EXE**, que é um programa executável. Após a criação do **.EXE** o compilador não é mais necessário. Na realidade quando pedimos para gerar um **.EXE** o compilador e o linkeditor atuam de tal maneira que, para o programador, é como se apenas um passo estivesse sendo executado.

No nosso curso utilizaremos o compilador do programa gráfico DEV-C++. Esse programa é Open Source e serve para escrever programas tanto em C quanto em C++. Assista ao mini-curso que mostra como instalar esse programa e como iniciar e executar os seus programas em C.

O C é Case Sensitive

Queremos dizer com isso que *maiúsculas e minúsculas fazem diferença*. Se você declarar uma variável com o nome **teste** ela será diferente de **Teste**, **TESTE**, **TestE** ou **tESTe**. Da mesma maneira, os comandos do C **if** e **for**, por exemplo, só podem ser escritos em minúsculas, caso contrário o compilador irá interpretá-los como variáveis.

Estrutura Básica de um Programa em C

O menor programa escrito em C seria o seguinte:

```
main()
{
}
```

Main() - Esse é o nome da função, que por convenção vem com os parênteses. A função Main precisa estar presente no seu projeto C. Ela é a função principal do programa. É através dessa função que o programa será executado. As chaves de abertura e fechamento significam abertura e fechamento de bloco. Seriam o equivalente ao Begin e End do pascal. O programa acima não faz nada. Vamos criar um pequeno programa que imprime algo na tela:

```
main()
{
      printf("Meu primeiro programa C");
}
```

O fato de o C possuir diversos compiladores pode causar uma certa confusão. O programa acima poderia ser compilado e executado sem problemas em alguns compiladores. Outros iriam reclamar. No nosso caso, no programa Dev-C não haverá problema nenhum. O Dev-C utiliza o compilador GCC. No entanto, como o Dev-C é um programa gráfico e vai chamar o CMD para mostrar o resultado do nosso programa, nós vamos inserir mais uma linha para que haja uma pausa na tela antes do CMD ser fechado. Nosso programa ficará assim:

```
main()
{
      printf("Meu primeiro programa C");
      system("pause");
}
```

Com o comando system("pause") haverá uma pausa na tela e poderemos ver o resultado do nosso programa, que simplesmente imprime uma linha na tela. Observe que as duas instruções terminam com ponto-e-vírgula (;). Toda instrução em C deve terminar com ponto-e-vírgula.

A função printf()

Essa é uma função do tipo E/S (entrada/saída), ou I/O (input/output) em inglês. O uso dos parênteses indica que printf é realmente uma função. Dentro dos parênteses nós digitamos o argumento dessa função. No nosso caso, a frase que foi impressa na tela.

Sintaxe:

```
printf("expr. de controle", lista de argumentos)
```

Vamos a outro exemplo:

```
main()
{
      printf("Este eh o numero dois: %d",2);
      system("pause");
}
```

O que será impresso pelo programa acima é o seguinte:

```
Este eh o numero dois: 2
```

Observe que no programa acima estamos passando dois argumentos para a função printf(). Primeiro a frase e depois o número 2. Os argumentos são separados por vírgula. A expressão de controle, que é o primeiro argumento entre aspas, pode conter caracteres que serão exibidos na tela, como as frases que já utilizamos, e códigos de formatação que indicam o formato em que os argumentos devem ser impressos. No exemplo anterior utilizamos o **%d** que indica que o segundo argumento deve ser impresso em formato decimal.

Mas esse código para o formato decimal não é o único que podemos utilizar na

função printf(). Observe a tabela abaixo onde constam os códigos que podemos utilizar com esta função:

Código de Formatação	Formato
%c	Caracteres simples
%d	Decimal – números inteiros
%e	Notação científica
%f	Ponto flutuante
%g	%e ou %f (o mais curto)
%o	Octal
%s	Cadeia de caracteres
%u	Decimal sem sinal
%x	Hexadecimal
%ld	Decimal longo
%lf	Ponto flutuante longo (double)

Podemos utilizar os códigos acima conforme os exemplos abaixo:

```
main()
{
	printf("Este eh o numero dois: %d",2);
	printf("%s fica a %d kilometros daqui","Fortaleza",1200);
	printf("A letra %c pronuncia-se %s",'W',"dábiliu");
	system("pause");
}
```

Se você já está testando os exemplos acima deve ter observado que sai tudo impresso numa linha só. Como fazer para "quebrar" a linha? No C existe o que chamamos de caracteres de escape. Observe o exemplo abaixo:

```
main()
{
	printf("Este eh o numero dois: %d \n",2);
	printf("%s fica a %d kilometros daqui \n","Fortaleza",1200);
	printf("A letra %c pronuncia-se %s \n",'W',"dábiliu");
	system("pause");
}
```

Observou algo novo? Sim. Existe um **\n** em cada uma das linhas. Esse é um caractere de escape. Quando a função printf() encontra um **\n** na expressão de controle ela sabe que precisar simular um [enter]. Observe a tabela de caracteres de escape abaixo:

Caracteres de Escape	Significado
\n	Nova linha – simula o [enter]
\r	Retorno de cursor
\t	Tab
\b	Retrocesso
\”	Aspas
\\	Barra
\f	Salta página de formulário
\0	Nulo

Faça alguns exemplos utilizando os caracteres acima e tente compreender a função de cada um deles.

A função printf() faz parte de uma biblioteca que vem com o C. Conforme já mencionado essa é uma função de entrada/saida. Logo, a biblioteca que a contém é a **stdio.h**. O h significa header (cabeçalho). Isso significa que esse é um arquivo-cabeçalho. Como assim? A função printf() está contida dentro desse arquivo. Para usarmos essa função é necessário informarmos ao nosso programa onde essa função se encontra. Fazemos isso por inserir uma instrução no *cabeçalho* do nosso programa. Ué, então porque conseguimos utilizar essa função sem mencionar nada nos exemplos acima? É porque o Dev-C já trata essa função como algo padrão. Mas o mesmo código acima poderia causar problemas se fosse compilado diretamente na linha de comando.

O nome do arquivo, stdio, também tem um significado:

std – standard (padrão)
io – input/output (entrada/saída)

Ou seja, esse arquivo contém as funções de entrada/saída. Logo, a função printf() teria que estar nesse arquivo. E como faço para *incluir* esse arquivo no cabeçalho do meu programa? O raciocínio é esse mesmo: incluir. Pense na palavra incluir em inglês! Veja o exemplo abaixo:

```
#include <stdio.h>
void main ()
```

```
{
      /* esse programa pede ao usuario para entrar com um
         valor inteiro que chama de dias, depois ele divide
         esse numero por 365,25, ou seja, converte os dias
         para anos e informa ao usuario quantos anos equivalem
         aos dias digitados
      */
      int Dias;                              // Declaracao de Variaveis
      float Anos;
      printf ("Entre com o número de dias: "); // Entrada de Dados
      scanf ("%d",&Dias);
      Anos=Dias/365.25;                        // Conversao Dias->Anos
      printf ("\n\n%d dias equivalem a %f anos.\n",Dias,Anos);
      system("pause");
}
```

Observe a primeira linha do nosso programa: #include <stdio.h>. É assim que incluímos um arquivo que tem as funções que queremos utilizar em nosso programa. Mas espere. Esse exemplo tem bastante coisa nova. Para sermos mais precisos, esse exemplo traz 3 novos conceitos que vamos começar a estudar agora.

Comentários em C

No exemplo anterior utilizamos dois estilos de comentários:

/* comentário 1 comentário 2 */	Esse estilo serve para comentários longos que ocuparão mais de uma linha. Você abre o comentário com /* e fecha o mesmo com */.
// comentário	Esse estilo serve para comentários breves, de uma linha só. Tudo que está a frente das duas barras não será levado em conta pelo compilador.

Variáveis em C

As variáveis são o aspecto fundamental em qualquer linguagem de computador. Uma variável em C é um espaço de memória reservado para armazenar um certo tipo de dado e tem um nome para referenciar o seu conteúdo. Esse espaço de memória pode conter, a cada tempo, valores diferentes.

No exemplo anterior utilizamos duas variáveis: Dias e Anos. Observe que antes do nome das variável temos as palavras int para Dias e float para Anos. O que significam? São o tipo dessas variáveis. Nós precisamos informar ao C de que tipo são os valores que queremos trabalhar em memória.

Observe abaixo os tipos de dados que podemos utilizar no C:

Tipo	Tamanho-Bytes	Escala
char	1	-127 a 127
unsigned char	1	0 a 255
signed char	1	-127 a 127
int	4	-2.147.483.648 a 2.147.483.647
Tipo	**Tamanho-Bytes**	**Escala**
unsigned int	4	0 a 4.294.967.295
signed int	4	-2.147.483.648 a 2.147.483.647
short int	2	-32.768 a 32.767
unsigned short int	2	0 a 65.535
signed short int	2	-32.768 a 32.767
long int	4	-2.147.483.648 a 2.147.483.647
signed long int	4	-2.147.483.648 a 2.147.483.647
unsigned long int	4	0 a 4.294.967.295
float	4	Seis digitos de precisão
double	8	Dez digitos de precisão
long double	10	Dez digitos de precisão

A Função scanf()

O terceiro novo conceito abordado no nosso exemplo anterior foi a função scanf(). Já vimos que a função printf() serve para imprimir informações na tela. E quando precisarmos solicitar algo ao usuário? Utilizamos a scanf().

Sintaxe:

```
scanf("expressão de controle",lista de argumentos);
```

A função scanf() é muito semelhante à printf(). Também é uma função de entrada e saída. Vamos utilizar os mesmos códigos de formatação que usamos na printf(). Mas ela tem uma diferença fundamental: o operador de endereço (&).

A memória do computador é dividida em bytes, que são numerados de zero até o limite de memória da máquina (524.287 se a máquina tem 512K de memória). Estes números são chamados de endereços de bytes. Toda variável ocupa uma certa localização na memória e seu endereço é o primeiro byte ocupado por ela.

Um número inteiro ocupa 4 bytes. Se você declarou a variável **nota** como inteiro e atribuiu a ela o valor 5, quando **nota** for referenciada devolverá o valor 5. No entanto, se você referenciar **nota** precedida por **& (&nota)**, você receberá o endereço do primeiro byte onde **nota** está guardada.

Observe novamente o nosso exemplo:

```
#include <stdio.h>
void main ()
{
      /* esse programa pede ao usuario para entrar com um
         valor inteiro que chama de dias, depois ele divide
         esse numero por 365,25, ou seja, converte os dias
         para anos e informa ao usuario quantos anos equivalem
         aos dias digitados
      */
      int Dias;                              // Declaracao de Variaveis
      float Anos;
      printf ("Entre com o número de dias: "); // Entrada de Dados
      scanf ("%d",&Dias);
      Anos=Dias/365.25;                        // Conversao Dias->Anos
      printf ("\n\n%d dias equivalem a %f anos.\n",Dias,Anos);
      system("pause");
}
```

Quando solicitamos ao usuário que ele entre com o número de dias utilizamos o código de formatação **%d**, pois vamos armazenar um inteiro. Após isso, no argumento, utilizamos a variável **Dias** precedida por **& (&Dias)**. Isso faz com que o valor digitado pelo usuário seja armazenado num endereço de memória.

Palavras Reservadas

Toda linguagem possuem palavras que não podem ser utilizadas como nomes de variáveis. Essas palavras são reservadas para o trabalho do compilador em cima delas. Por exemplo, a palavra **for** não poderia ser utilizada como um nome de

variável, pois o compilador sabe que essa palavra inicia um laço, que veremos mais a frente.

Veja agora a lista de palavras reservadas do C:

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Operadores Aritméticos

O C é uma linguagem com muitos operadores, cerca de 40. C oferece 6 operadores aritméticos binários (que operam sobre dois operandos) e um operador unário (que opera sobre um operando).

São eles:

Binários:

= Atribuição
+ Soma
- Subtração
* Multiplicação
/ Divisão
% Módulo (resto da divisão inteira)

Unário:

- Menos unário

Não existe muito o que falar sobre cada operador. Vejamos na prática o que

cada um deles realmente faz. Execute o seguinte exemplo no Dev-C:

```
#include <stdio.h>
void main ()
{
	int a,b,c;
	float resultado = 0.0;
	printf ("Digite tres numeros: ");
	scanf ("%d %d %d",&a, &b, &c);
	resultado = ((a + b) * c) / (c-a);
	printf("O resultado de ((a + b) * c) / (c-a) eh %f \n", resultado);
	printf ("O resto da divisao entre a e b eh %d \n", a%b);
	printf ("E se multiplicarmos c por -1 o resultado sera %d \n", -c);
	system("pause");
}
```

Faça alguns testes com o programa acima. Tente compreender o seu funcionamento. Faça os cálculos numa calculadora e observe se o resultado está correto. Comente sobre esse programa na lista de discussão.

No programa anterior utilizamos todos os operadores aritméticos. Vamos observar agora um programa que utiliza alguns operadores e tem uma finalidade mais precisa:

```
#include <stdio.h>
void main ()
{
	/*
	para converter de Fahrenheit para Celsius usa-se a formula:
	c = (f-32) / 1.8
	para converter de Celsius para Fahrenheit usa-se a formula:
	f = c * 1.8 + 32
	*/
	float f;
	printf("Entre com a temperatura em Fahrenheit\n\n");
	scanf("%f",&f);
	printf("%.f em Fahrenheit equivale a %.f em Celsius",f,(f-32)/1.8);
	printf("\n");
	system("pause");
}
```

O programa acima solicita ao usuário que entre com uma temperatura em Fahrenheit e devolve o resultado da mesma em Celsius.

Operadores de Incremento e Decremento

Observe o seguinte trecho de programa em Pascal:

```
var
      a, b, soma: integer;
begin
      soma := 0;
      a := 5;
      b := 6;
      soma := a + b;
      soma := soma + 1;
end;
```

O que nos interessa mesmo no algoritmo acima é a última linha onde somamos o resultado de soma com 1 e atribuímos à própria soma. O C possui u operador de incremento que faz isso de uma forma mais interessante. Observe:

```
void main ()
{
      int a, b, soma;
      a = 5;
      b = 6;
      soma = a + b;
      soma++;
      printf("%d",soma);
      system("pause");
}
```

Note a diferença. Basta utilizar o operador de incremento (++). O mesmo já atribui 1 à variável soma. Da mesma forma o operador de decremento (--) faria uma subtração de 1 e atribuiria o novo valor a variável.

É importante saber que os operadores podem ser utilizados antes ou após a variável. No entanto, existe uma diferença fundamental. Observe:

```
void main ()
{
      int n = 5, x;
      x = n++;
      printf("x=%d  n=%d",x,n);
      system("pause");
}
```

A saída será x=5 e n=6. Vamos fazer uma pequena alteração:

```
void main ()
{
      int n = 5, x;
      x = ++n;
      printf("x=%d  n=%d",x,n);
      system("pause");
}
```

A saída será x=6 e n=6. Percebeu a diferença? Quando o operador vem após a variável, primeiro o valor da variável é atribuído à outra variável, depois é que ela é incrementada. Quando o operador vem antes da variável, primeiro é feito o incremento e depois a atribuição. Faça mais alguns testes atribuindo valores entre variáveis e utilizando os operadores de incremento e decremento para entender o seu funcionamento.

Operadores Aritméticos de Atribuição

Existem ainda os operadores aritméticos de atribuição (+=; -=; *=; /=; %=). Vejamos como eles funcionam:

Exemplo	Equivale a
i += 2;	i = i + 2;
x *= y+1;	x = x * (y+1);
t /= 2.5;	t = t / 2.5;
p %= 5;	p = p % 5;
d -= 3;	d = d - 3;

Operadores Relacionais

Esses operadores são utilizados para fazer comparações. São os seguintes:

>	Maior
>=	Maior ou igual
<	Menor
<=	Menor ou igual
==	Igualdade
!=	Diferente

Mais adiantes utilizaremos bastante esses operadores quando trabalharmos com controle de fluxo e laços.

Comando de Decisões (Controle de Fluxo)

Toda linguagem de computador precisa oferecer um mínimo de três formas básicas de controle:

1 – executar uma série de instruções;
2 – praticar testes para decidir entre ações alternativas.
3 – repetir uma seqüência de instruções até que uma certa condição seja encontrada;

Vamos conhecer o segundo item a partir de agora e o terceiro será explorado mais tarde.

O C oferece 4 principais estrutura de decisão: if, if-else, switch e o operador condicional ternário.

O Comando if

Esse comando instrui o computador a tomar uma decisão simples.

Sintaxe:

```
if (expressão de teste)
      instrução;
```

Observe o exemplo:

```
void main ()
{
      int n;
      printf("Digite um valor, por favor!\n");
      scanf("%d",&n);
      if (n > 5)
            printf("o valor de n eh %d e eh maior que dois\n",n);
      system("pause");
}
```

Se o usuário digitar um número maior que cinco ele verá uma mensagem na tela. Caso digite um número menor ou igual a 5 o programa não faz nada. Mas e se eu quisesse inserir mais um comando dentro desse if? Caso se deseje inserir mais que uma linha de comando em um if será necessário abrir um bloco com chaves e fechá-lo após o último comando. Observe:

```
void main ()
{
	int n;
	printf("Digite um valor, por favor!\n");
	scanf("%d",&n);
	if (n > 5)
	{
		printf("o valor de n eh %d e eh maior que cinco\n",n);
		printf("esse if tem dois comandos, por isso utilizamos a chave\n");
		printf("de abertura e a chave de fechamento de bloco\n");
	}
	system("pause");
}
```

No entanto, esse programa não faz nada se o usuário digitar um número menor ou igual a cinco. Vamos mudar isso. Utilizemos o else. Vejamos:

```
void main ()
{
	int n;
	printf("Digite um valor, por favor!\n");
	scanf("%d",&n);
	if (n > 5)
	{
		printf("o valor de n eh %d e eh maior que cinco\n",n);
		printf("esse if tem dois comandos, por isso utilizamos a chave\n");
		printf("de abertura e a chave de fechamento de bloco\n");
	}
	else
		printf("o valor de n eh %d e eh menor ou igual a cinco\n",n);

	system("pause");
}
```

Nesse primeiro momento utilizamos apenas uma instrução no else, por esta razão não foi necessário abrir um bloco. Mas o princípio é o mesmo: se fosse necessário inserir mais comandos abaixo do else e que pertencessem ao mesmo, seria necessário abrir e fechar um bloco.

Comandos if Aninhados

Quando um if está dentro de outro dizemos que o if interno está aninhado. Veja o exemplo:

```
void main ()
{
	int n;
	printf("Digite um valor, por favor!\n");
	scanf("%d",&n);
	if (n > 5)
```

```
            if (n < 10)
                  printf("n eh %d e eh maior que cinco e menor que 10 \n",n);

      system("pause");
}
```

Operadores Lógicos

Está na hora de conhecermos mais 3 operadores:

&&	Lógico E	Seria o and no Pascal
||	Lógico Ou	Seria o or no Pascal
!	Lógico de negação	Seria o not no Pascal

Exemplo:

```
void main ()
{
      int n;
      printf("Digite um valor, por favor!\n");
      scanf("%d",&n);
      if (n>5 && n<10)
            printf("n eh %d e eh maior que cinco e menor que 10 \n",n);

      system("pause");
}
```

Precedência dos Operadores

Precedência quer dizer qual dos operadores será executado primeiro numa instrução quando eles aparecerem juntos. Observe a tabela abaixo, pois ela mostra a precedência dos operadores de cima para baixo e da esquerda para a direita.

Operadores	Tipos
! - ++ --	Unários; não lógico e menos aritméticos
* / %	Aritméticos
+ -	Aritméticos
< > <= >=	Relacionais
== !=	Relacionais
&&	Lógico E

||	Lógico OU
= += -= *= /= %=	Aritméticos de atribuição

Ou seja, o operador **! (not)** é o de maior precedência e o **%=** é o de menor.

O Comando switch

Quando temos uma ou duas escolhas a fazer usar o if pode ser o ideal. Mas às vezes é necessário escolher entre uma de muitas opções. Quando isso ocorre, podemos utilizar o comando switch.

Sintaxe

```
switch(expressão constante) {
      case constante 1:
            instruções;
            break;
      case constante 2:
            instruções;
            break;
      default:
            instruções;
}
```

Vamos construir um pequeno menu e quando o usuário escolher uma das opções imprimiremos qual delas ele escolheu. Observe:

```
void main ()
{
     printf("------------------------------\n");
     printf("| PROGRAMA BANCO DE DADOS    |\n");
     printf("------------------------------\n");
     printf("| 1 - INCLUSAO               |\n");
     printf("| 2 - EXCLUSAO               |\n");
     printf("| 3 - ALTERACAO              |\n");
     printf("------------------------------\n");
     printf("| 4 - SAIR                   |\n");
     printf("------------------------------\n");

     int op;

     printf("Selecione uma das opcoes acima: ");
     scanf("%d",&op);

     switch(op)
     {
        case 1:
           printf("escolheu a opcao INCLUSAO");
           break;
        case 2:
           printf("escolheu a opcao EXCLUSAO");
```

```
            break;
         case 3:
            printf("escolheu a opcao ALTERACAO");
            break;
         case 4:
            printf("escolheu a opcao SAIR");
            break;
         default:
            printf("digitou outra coisa...");
      }
      printf("\n");

      system("pause");
}
```

Veja que após cada instrução é necessário inserir a instrução break. Esse comando "quebra" o switch, ou seja sai do switch. Utilizaremos esse comando novamente quando estivermos trabalhando com laços.

Operador Condicional Ternário ?

Conheceremos agora um if-else um pouco estranho. Quando estivermos numa situação de uma simples condição podemos utilizar essa estrutura.

Sintaxe:

```
condição ? expressão1 : expressão2
```

Traduzindo: se a condição for verdadeira execute a expressão1, senão execute a expressão2.

Exemplo:

```
max = (num1 > num2) ? num1 : num2;
```

A instrução acima equivale a:

```
if (num1 > num2)
      max = num1;
else
      max = num2;
```

Laços

Em C existem 3 estruturas principais de laços: for, while e do-while.

O Laço for

O laço for engloba 3 expressões numa única e é útil principalmente quando queremos repetir algo um número fixo de vezes.

O exemplo seguinte imprime os número 0 a 9 utilizando um laço for:

```
void main ()
{
	int conta;
	for(conta=0;conta<10;conta++)
		printf("Conta=%d\n",conta);
	system("pause");
}
```

Sintaxe:

```
for(inicialização;teste;incremento)
	instrução;
```

A expressão de inicialização é uma atribuição e é executada uma única vez antes do laço ser iniciado.

O teste é uma instrução de condição que controla o laço. Essa expressão é avaliada a cada passo do laço. Quando a mesma torna-se falsa o laço é terminado.

A expressão de incremento define a maneira como a variável de controle do laço será alterada a cada vez que o laço é repetido.

Vamos modificar o programa anterior para imprimir apenas os números ímpares de 1 a 9.

```
void main ()
{
	int conta;
	for(conta=1;conta<10;conta+=2)
		printf("Conta=%d\n",conta);
	system("pause");
}
```

O laço for é uma estrutura bem flexível. Na verdade você poderia informar mais do que um parâmetro em cada uma das expressões do laço ou até mesmo não

fornecer parâmetro algum. Observe:

```
void main ()
{
	int x,y;
	for(x=0, y=0; x+y<100; x=x+1, y=y+1)
		printf("%d\n",x+y);
	system("pause");
}
```

Execute o programa acima e descubra o que o mesmo faz. Analise o código e veja que estamos utilizando mais do que um parâmetro na inicialização e no incremento.

O Laço while

O while utiliza os mesmos elementos do for, só que distribuídos de forma diferente.

Sintaxe:

```
inicialização;
while(expressão de teste){
	instrução;
	incremento;
}
```

Vamos fazer o mesmo programa para contar de 0 a 9 com o while.

```
void main ()
{
	int conta;
	conta = 0;
	while(conta < 10) {
		printf("Conta=%d\n",conta);
		conta++;
	}
	system("pause");
}
```

O Laço do-while

O laço while executa os comandos enquanto sua expressão de teste for verdadeira. Já o do-while cria um ciclo de repetição até que a expressão de teste seja falsa. Outra diferença é que devido a estrutura, o laço do-while é executado pelo

menos uma vez.

Sintaxe:

```
do {
      instruções;
} while (expressão de teste);
```

Vamos criar um menu dinâmico com o do-while. O usuário só sairá do nosso programa quando selecionar a opção sair. Vamos utilizar o mesmo menu criado anteriormente.

```
void main ()
{
 int op;
     do
     {
        system("cls");

        printf("------------------------------\n");
        printf("| PROGRAMA BANCO DE DADOS    |\n");
        printf("------------------------------\n");
        printf("| 1 - INCLUSAO               |\n");
        printf("| 2 - EXCLUSAO               |\n");
        printf("| 3 - ALTERACAO              |\n");
        printf("------------------------------\n");
        printf("| 4 - SAIR                   |\n");
        printf("------------------------------\n");

         scanf("%d",&op);
         switch(op)
         {
            case 1:
               printf("escolheu a opcao INCLUSAO \n");
               system("pause");
               break;
            case 2:
               printf("escolheu a opcao EXCLUSAO \n");
               system("pause");
               break;
            case 3:
               printf("escolheu a opcao ALTERACAO \n");
               system("pause");
               break;
            case 4:
               printf("escolheu a opcao SAIR \n");
               break;
            default:
               printf("digitou outro numero... \n");
               system("pause");
         }
     } while (op != 4);

      system("pause");
}
```

Os Comandos break e continue

Dentro dos laços você poderá utilizar os comandos break e continue. Como já foi explicado o comando break vai interromper o laço. O comando continue vai pular uma iteração do laço e passar para a próxima. Vamos modificar nosso programar que conta de 0 a 9. Se o número for ímpar nós vamos chamar o comando continue. Vejamos o que ocorre:

```
void main ()
{
	int conta;
	conta = 0;
	while(conta < 10) {
		if(conta%2 == 1){
			conta++;
			continue;
		}
		printf("Conta=%d\n",conta);
		conta++;
	}
	system("pause");
}
```

O comando continue passa o fluxo do programa para o início do laço novamente. Dessa forma apenas os números pares serão impressos.

Explorando a Função printf()

Podemos definir o tamanho mínimo para a impressão de um campo. Observe:

```
void main ()
{
	printf("Os empregados sao %2d \n",350);
	printf("Os empregados sao %4d \n",350);
	printf("Os empregados sao %5d \n",350);
	system("pause");
}
```

Execute o programa acima e observe o que ocorre. O tamanho do campo é definido pelo número que colocamos antes da letra d. A esquerda o C preenche com espaços em branco. É interessante quando queremos alinhar um resultado e mostrar para o usuário.

Podemos também controlar quantas casas decimais serão mostradas num número de ponto flutuante. Veja:

```
void main ()
{
	printf("%.2f \n",123456.789);
	printf("%4.2f \n",123456.789);
	printf("%3.2f \n",123456.789);
	printf("%3.1f \n",123456.789);
	printf("%10.3f \n",123456.789);
	system("pause");
}
```

O número que vem antes do ponto serve para definir o tamanho do campo, como já vimos anteriormente. Já o número após o ponto e antes do f serve para dizer ao C quantas casas decimais queremos que ele apresente no resultado. Veja que o C arredonda os valores quando pedimos um número de casas menor que o número a ser exibido. É Utilizado o arredondamento simétrico.

Imprimindo Caracteres

Em C um caractere pode ser representado de diversas maneiras: o próprio caractere entre aspas simples ou sua representação decimal, hexadecimal ou octal segundo a tabela ASCII. Veja o programa abaixo:

```
void main ()
{
	printf("%d %c %x %o \n",'A', 'A', 'A', 'A');
	system("pause");
}
```

A saída será: **65 A 41 101**.

A Tabela ASCII (American Standard Code for Information Interchange) é usada pela maior parte da industria de computadores para a troca de informações. Cada caracter é representado por um código de 8 bits (um byte). Abaixo mostramos a tabela ASCII de 7 bits. Existe uma tabela estendida para 8 bits que inclui os caracteres acentuados.

Caracter	Decimal	Hexa	Binário	Comentário
NUL	00	00	0000 0000	Caracter Nulo
SOH	01	01	0000 0001	Começo de cabeçalho de transmissão

STX	02	02	0000 0010	Começo de texto
ETX	03	03	0000 0011	Fim de texto
EOT	04	04	0000 0100	Fim de transmissão
ENQ	05	05	0000 0101	Interroga
ACK	06	06	0000 0110	Confirmação
BEL	07	07	0000 0111	Sinal sonoro
BS	08	08	0000 0100	Volta um caracter
HT	09	09	0000 1001	Tabulação Horizontal
LF	10	0A	0000 1010	Próxima linha
VT	11	0B	0000 1011	Tabulação Vertical
FF	12	0C	0000 1100	Próxima Página
CR	13	0D	0000 1101	Início da Linha
SO	14	0E	0000 1110	Shift-out
SI	15	0F	0000 1111	Shift-in
DLE	16	10	0001 0000	Data link escape
D1	17	11	0001 0001	Controle de dispositivo
D2	18	12	0001 0010	Controle de dispositivo
D3	19	13	0001 0011	Controle de dispositivo
D4	20	14	0001 0100	Controle de dispositivo
NAK	21	15	0001 0101	Negativa de Confirmação
Caracter	**Decimal**	**Hexa**	**Binário**	**Comentário**
SYN	22	16	0001 0110	Synchronous idle
ETB	23	17	0001 0111	Fim de transmissão de bloco
CAN	24	18	0001 1000	Cancela
EM	25	19	0001 1001	Fim de meio de transmissão
SUB	26	1A	0001 1010	Substitui
ESC	27	1B	0001 1011	Escape
FS	28	1C	0001 1100	Separador de Arquivo
GS	29	1D	0001 1101	Separador de Grupo
RS	30	1E	0001 1110	Separador de registro
US	31	1F	0001 1111	Separador de Unidade
Espaço	32	20	0010 0000	
!	33	21	0010 0001	
"	34	22	0010 0010	
#	35	23	0010 0011	
$	36	24	0010 0100	
%	37	25	0010 0101	

&	38	26	0010 0110	
'	39	27	0010 0111	
(	40	28	0010 1000	
)	41	29	0010 1001	
*	42	2A	0010 1010	
+	43	2B	0010 1011	
,	44	2C	0010 1100	
-	45	2D	0010 1101	
.	46	2E	0010 1110	
/	47	2F	0010 1111	
0	48	30	0011 0000	
1	49	31	0011 0001	
2	50	32	0011 0010	
3	51	33	0011 0011	
4	52	34	0011 0100	
5	53	35	0011 0101	
6	54	36	0011 0110	
7	55	37	0011 0111	
8	56	38	0011 1000	
9	57	39	0011 1001	
Caracter	**Decimal**	**Hexa**	**Binário**	**Comentário**
:	58	3A	0011 1010	
;	59	3B	0011 1011	
<	60	3C	0011 1100	
=	61	3D	0011 1101	
>	62	3E	0011 1110	
?	63	3F	0011 1111	
@	64	40	0100 0000	
A	65	41	0100 0001	
B	66	42	0100 0010	
C	67	43	0100 0011	
D	68	44	0100 0100	
E	69	45	0100 0101	
F	70	46	0100 0110	
G	71	47	0100 0111	
H	72	48	0100 1000	
I	73	49	0100 1001	

J	74	4A	0100 1010	
K	75	4B	0100 1011	
L	76	4C	0100 1100	
M	77	4D	0100 1101	
N	78	4E	0100 1110	
O	79	4F	0100 1111	
P	80	50	0101 0000	
Q	81	51	0101 0001	
R	82	52	0101 0010	
S	83	53	0101 0011	
T	84	54	0101 0100	
U	85	55	0101 0101	
V	86	56	0101 0110	
W	87	57	0101 0111	
X	88	58	0101 1000	
Y	89	59	0101 1001	
Z	90	5A	0101 1010	
[	91	5B	0101 1011	
\	92	5C	0101 1100	
]	93	5D	0101 1101	
Caracter	**Decimal**	**Hexa**	**Binário**	**Comentário**
^	94	5E	0101 1110	
_	95	5F	0101 1111	
`	96	60	0110 0000	
a	97	61	0110 0001	
b	98	62	0110 0010	
c	99	63	0110 0011	
d	100	64	0110 0100	
e	101	65	0110 0101	
f	102	66	0110 0110	
g	103	67	0110 0111	
h	104	68	0110 1000	
i	105	69	0110 1001	
j	106	6A	0110 1010	
k	107	6B	0110 1011	
l	108	6C	0110 1100	
m	109	6D	0110 1101	

n	110	6E	0110 1110	
o	111	6F	0110 1111	
p	112	70	0111 0000	
q	113	71	0111 0001	
r	114	72	0111 0010	
s	115	73	0111 0011	
t	116	74	0111 0100	
u	117	75	0111 0101	
v	118	76	0111 0110	
w	119	77	0111 0111	
x	120	78	0111 1000	
y	121	79	0111 1001	
z	122	7A	0111 1010	
{	123	7B	0111 1011	
|	124	7C	0111 1100	
}	125	7D	0111 1101	
~	126	7E	0111 1110	
DELETE	127	7F	0111 1111	

Outras Funções Importantes

Existem situações em que scanf() não serve porque você precisa pressionar [enter] para que a mesma termine sua leitura. O C oferece funções que lêem o caractere no momento em que ele é digitado.

A função getche() lê o caractere e permite que seja impresso na tela. Esta função não aceita argumentos e devolve o caractere lido para a função que a chamou. Vamos a um exemplo:

```
void main ()
{
	char caractere;
	printf("Digite um caractere: ");
	caractere = getche();
	printf("\nA tecla pressionada foi %c \n", caractere);
	system("pause");
}
```

Observe que a função getche() permite que o caractere digitado pelo usuário

seja impresso na tela. Já a função getch() não permite. Vamos ao mesmo exemplo com a função getch().

```
void main ()
{
      char caractere;
      printf("Digite um caractere: ");
      caractere = getch();
      printf("\nA tecla pressionada foi %c \n", caractere);
      system("pause");
}
```

As funções acima são interessantes, mas capturam apenas um caractere. E se quiséssemos capturar uma frase? Para isso vamos utilizar a gets(). Veja o exemplo abaixo:

```
void main ()
{
     char frase[100];
     printf("Digite uma frase e eu direi quantos caracteres ela possui \n\n");
     gets(frase);

     int anda = 0;
     //o caractere '\0' indica que se chegou ao final da cadeia de caracteres
     //eh conhecido como caractere nulo
     while (frase[anda] != '\0')
     {
        anda++;
     }
     printf("A frase tem %d caracteres, contando com os espacos",anda);
     printf("\n");

      system("pause");
}
```

Procedimentos e Funções

Até o momento falamos muito de funções já prontas que o C disponibiliza para nós através de suas bibliotecas. E se quiséssemos elaborar nossas próprias funções? Note que Main é em si uma função. É a função principal de um programa. Observe o exemplo abaixo:

```
#include <stdio.h>

main ()
{
      void repete(char a, int vezes); //main sabe que existe um procedimento
      repete('-',30);
      printf("| PROGRAMA BANCO DE DADOS    |\n");
```

```
	repete('-',30);
	printf("| 1 - INCLUSAO                |\n");
	printf("| 2 - EXCLUSAO                |\n");
	printf("| 3 - ALTERACAO               |\n");
	repete('-',30);
	printf("| 4 - SAIR                    |\n");
	repete('-',30);

	system("pause");
}

void repete(char a, int vezes)
{
	int i=0;
	for (i=0; i<vezes; i++)
	{
		printf("%c",a);
	}
	printf("\n");
}
```

Eu criei um procedimento chamado repete que recebe dois parâmetros: um char e um int. O objetivo desse procedimento é repetir determinado caractere um número qualquer de vezes. No caso anterior eu usei para repetir o hífen (-) 30 vezes quando desenhei o meu menu. Mas porque estou chamando o repete de procedimento e não de função? Existe uma diferença básica entre procedimento e função. A função sempre retorna um valor. O procedimento não. E como sei que o procedimento não está retornando um valor? Observe a assinatura do procedimento, ou seja, a declaração do mesmo:

void repete(char a, int vezes)

Em primeiro lugar precisamos definir o tipo de retorno. No caso acima o retorno é do tipo void, ou seja, sem valor. Isso indica que não haverá retorno. Após o tipo de retorno indicamos o nome de nosso procedimento ou função. No nosso caso o procedimento chama-se repete. Depois, dentro dos parênteses, informamos os nossos parâmetros. É bom frisar que não é obrigatória a presença de parâmetros.

Para entendermos a diferença entre procedimentos e funções vamos criar agora uma função bem simples:

```
main ()
{
	int num1, num2;
	printf("Digite dois numeros para a realizacao da soma: \n");
	scanf("%d %d",&num1, &num2);
	printf("a soma dos numeros: %d\n",soma(num1,num2));
	system("pause");
```

```
}

int soma(int a, int b)
{
      return (a+b);
}
```

Observe que o tipo de retorno da função é diferente de void. No caso da função soma o retorno é do tipo int. Além disso a função deve retornar um valor com a utilização do comando return.

A Biblioteca math.h

Para finalizarmos a primeira parte do nosso curso vamos analisar a biblioteca math.h que já traz prontas diversas funções matemáticas.

Para a utilização dessa biblioteca basta incluí-la no cabeçalho de nosso programa e depois chamar suas funções. Observe o exemplo:

```
#include <math.h>
main ()
{
      int num1, num2;
      printf("Digite dois numeros: \n");
      scanf("%d %d",&num1, &num2);
      printf("a raiz quadrada de %d eh %f\n",num1,sqrt(num1));
      printf("%d ao quadrado eh %f\n",num2,pow(num2,2));

      system("pause");
}
```

Agora observe a lista das funções contidas em math.h:

Função	Descrição
fabs(double)	Absoluto. abs (inteiro), cabs (complexo), fabs(real), labs (longo).
floor(double)	Trunca número real.
ceil(double)	Arredonda número real
modf(double, double *)	Quebra um real em uma parte inteira e outra fracionária.
sqrt(double)	Raiz quadrada.
atof(char *)	Converte string para real.
sin(double)	Seno.
cos(double)	Cosseno.

Função	Descrição
tan(double)	Tangente.
asin(double)	Arco seno.
acos(double)	Arco cosseno.
atan(double)	Arco tangente.
atan2(double, double)	Arco tangente.
frexp(double, int *)	Quebra número real em mantissa e expoente.
ldexp(double, int)	Equivale a (double * 2^int)
log(double)	Logaritmo natural.
log10(double)	Logaritmo na base 10.
pow(double, double)	x ^ y. Ex: pow(2,4) = 16.
exp(double)	e ^ double
sinh(double)	Seno hiperbólico.
cosh(double)	Cosseno hiperbólico.
tanh(double)	Tangente hiperbólico.

Agora você deve tentar resolver a lista de exercícios abaixo. Qualquer dificuldade envie para a lista de discussão:

Exercícios

01 - Elaborar um programa que leia dois números reais e apresente como resultado a soma e o produto entre eles.

02 - Elaborar um programa que leia o nome e as 4 notas de um aluno e apresente como resultado o nome e a média aritmética das notas.

03 - Elaborar um programa que receba um número positivo e maior que zero, calcule e mostre o número digitado ao quadrado; o número digitado ao cubo; a raiz quadrada do número digitado e a raiz cúbica do número digitado.

04 - Elaborar um programa em C que a partir da idade do usuário decida se o mesmo é jovem (idade < 21), adulto (21 < idade < 70) ou idoso (idade > 70).

05 - Elaborar um programa que dados 3 números apresente-os ordenados na

tela.

06 - Elaborar um programa em C que decida qual o maior entre três números digitados por um usuário e o apresente na tela.

07 - Elaborar um programa em C que receba 3 números em qualquer ordem e apresente como resultado o maior e o menor entre eles.

08 - Elaborar um programa em C que resolva uma equação do 2^o grau ($ax^2 + bx + c = 0$).

09 - A nota final de um estudante é calculada a partir de três notas atribuídas respectivamente a um trabalho de laboratório, a uma avaliação semestral e a um exame final. A média das três notas mencionadas obedece aos pesos 2, 3 e 5, respectivamente. Elaborar um programa em C que receba as três notas, calcule e mostre a média ponderada e o conceito (A – 10 a 8; B – 8 a 7; C – 7 a 6; D – 6 a 5; E – 5 a 0).

10 - Elaborar um programa que realize a contagem de 1 a 96, ordenados de 8 em 8.

11 - Elaborar um programa que imprima a tabuada multiplicativa de 1 a 10 de um número N fornecido pelo usuário.

12 - Elaborar um programa que simule um jogo de adivinhação, onde o usuário deve informar um valor aleatório e o programa deve testar se o resultado está certo ou errado, calculando o número de vezes que o usuário necessitou para acertar. Em cada iteração o programa deve fornecer uma pista.

13 - Elaborar um programa que decida se um número informado pelo usuário é o número 1, é primo (possui apenas 2 divisores naturais) ou composto (possui mais de 2 divisores).

14 - Elaborar um programa que apresente em ordem decrescente os números

de 100 até 11.

15 - Elaborar um programa que passa dois argumentos (base e altura) para a função retang( ), cujo propósito é desenhar retângulos de vários tamanhos na tela, correspondentes a cômodos de uma casa.

16 - Elaborar um programa em C que peça ao usuário que digite 3 números reais, e calcule o módulo destes números e devolva o maior valor em módulo e o menor valor real, utilizando função.

17 - Elaborar um programa em C que calcula o fatorial de um número informado pelo usuário, utilizando função.

18 - Faça um programa que calcule o imposto de renda de 10 contribuintes considerando que o número de dependentes e renda mensal de cada contribuinte são valores fornecidos pelo usuário. Para cada contribuinte será feito um desconto de 5% de salário mínimo por dependente. Os valores da alíquota para cálculo do imposto são:

Até 2 salários mínimos	Isento
2..3 (inclusive)	5%
3..5 (inclusive)	10%
5..7 (inclusive)	15%
Acima de 7	20%

19 - Faca um programa para verificar se um número inteiro lido é quadrado perfeito. Para que um número seja quadrado perfeito a soma dos seus divisores deve resultar num valor igual ao número.

Ex: Os divisores de 6 são: 1, 2, 3 que somados totalizam 6, logo 6 é quadrado perfeito.

20 – Faça um programa que contenha um menu com 4 opções:

1 – calcular o fatorial de um número dado

2 – calcular a raiz quadrada de 3 números dados

3 – imprimir a tabuada completa de 1 a 10

4 – sair do programa

Cada um dos 3 itens acima deve ser elaborado em forma de função ou procedimento.