A ESTRUTURA BÁSICA DE UM PROGRAMA EM C

Um programa C consiste em uma ou várias "funções". Os nomes programa e função se confundem em C.

FORMA GERAL DAS FUNÇÕES C

Vamos começar pelo menor programa possível em C.

```
main()
{
}
```

Este programa compõe-se de uma única função chamada **main**.

`main()`	———	primeira função a ser executada
{	———	inicia o corpo da função
}	———	termina a função

Os parênteses após o nome indicam que esta é uma função. O nome de uma função C pode ser qualquer um com exceção de "main", reservado para a função que inicia a execução do programa.

Toda função C deve ser iniciada por uma chave de abertura, {, e encerrada por uma chave de fechamento, }.

O nome da função, os parênteses e as chaves são os únicos elementos obrigatórios de uma função.

Você pode colocar espaços, caracteres de tabulação e pular linhas à vontade em seu programa, pois o compilador ignora estes caracteres. Em C não há um estilo obrigatório.

Os exemplos a seguir mostram como o estilo de programação pode causar problemas de legibilidade aos programas C.

`main () {}`	`main ()` `{ }`

`main`
`(`
`)`
`{`
`}`
	`main`
	`() {}`

Todos estes exemplos são de programas que não fazem nada, ou seja, programas vazios.

A FUNÇÃO main()

A função **main()** deve existir em algum lugar de seu programa e marca o ponto de início da execução do programa. Se um programa for constituído de uma única função esta será **main()**.

INSTRUÇÕES DE PROGRAMA

Vamos adicionar uma instrução em nosso programa.

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

Observação: a linguagem C não possui muitos comandos definidos. A maioria de seus comandos vem de funções externas que estão armazenados em bibliotecas (arquivos com terminação **.h**). Para os comandos de entrada e saída de dados, precisamos de uma biblioteca que tenha essas funções, que é a biblioteca stdio.h. Portanto insira na primeira linha de seu programa a instrução:

#include <stdio.h>

Existem diversas bibliotecas para os mais diversos fins, como recursos gráficos (graphics.h), instruções de tela (conio.h), instruções matemáticas (math.h), instruções de strings (string.h), instruções diversas (stdlib.h). Ao logo do curso iremos vendo e utilizando cada uma delas, de acordo com sua necessidade.

Todas as instruções devem estar dentro das chaves que iniciam e terminam a função e são executadas na ordem em que as escrevemos.

As instruções C são sempre encerradas por um ponto-e-vírgula (;). O ponto-e-vírgula é parte da instrução e não um simples separador.

Esta instrução é uma chamada à função **printf()**, os parênteses nos certificam disso e o ponto-e-vírgula indica que esta é uma instrução.

Como em C não existe um estilo obrigatório, vamos reescrever o programa anterior de forma que a sua execução será exatamente como a do anterior:

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

A FUNÇÃO printf()

A função **printf()** é uma das funções de E/S (entrada e saída) que podem ser usadas em C. Ela não faz parte da definição de C mas todos os sistemas têm uma versão de **printf()** implementada.

Os parênteses indicam que estamos, realmente, procedendo com uma função. No interior dos parênteses estão as informações passadas pelo programa **main()** a função **printf()**, isto é "primeiro programa". Esta informação é chamada de argumento de **printf()**.

Quando o programa encontra esta linha, passa o controle para a função **printf()** que imprime na tela do seu computador

primeiro programa

e, quando encerra a execução desta, o controle é transferido novamente para o programa.

Sintaxe:

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

Outro exemplo:

```
main()
{
   printf("Este e' o numero dois: %d",2);
}
```

Este programa imprimirá na tela do seu computador:

Este e' o numero dois: 2

A função **printf()** pode ter um ou vários argumentos. No primeiro exemplo nós colocamos um único argumento: "primeiro programa". Agora, entretanto, colocamos dois: "Este e' o numero: **%d**" que está à esquerda e o valor 2 à direita. Estes dois argumentos são separados por uma vírgula.

A expressão de controle pode conter caracteres que serão exibidos na tela e códigos de formatação que indicam o formato em que os argumentos devem ser impressos. No nosso exemplo o código de formatação **%d** solicita a **printf()** imprimir o segundo argumento em formato decimal.

printf() é uma função da biblioteca padrão de C e pode receber um número variável de argumentos. Isto é, a cadeia de caracteres de controle e mais tantos argumentos quantas especificações de formato a cadeia de controle contiver.

Cada argumento deve ser separado por uma vírgula.

IMPRIMINDO CADEIA DE CARACTERES

Além do código de formatação decimal **(%d)**, **printf()** aceita vários outros. O próximo exemplo mostra o uso do código **%s** para imprimir uma cadeia de caracteres.

```
main()
{
   printf("%s esta a %d milhoes de milhas\ndo sol","Venus",67);
}
```

A saída será:

Venus esta a 67 milhoes de milhas
do sol

Aqui, além do código de formatação, a expressão de controle de **printf()** contém um conjunto de caracteres estranho: **\n**.

O **\n** é um código especial que informa a **printf()** que o restante da impressão deve ser feito em nova linha. A combinação de caracteres **\n** representa, na verdade, um único caractere em C, chamado de nova-linha. Em outras palavras, este caractere desempenha a mesma função que a executada quando pressionamos a tecla **[enter]** do teclado. Por que, então, não usar a tecla **[enter]** ? Porque, quando pressionamos a tecla **[enter]**, o editor de textos que estamos usando para editar nosso programa deixa a linha atual onde estamos trabalhando e passa para outra linha, deixando a linha anterior inacabada, e a função **printf()** não o tomará para a impressão.

Os caracteres que não podem ser obtidos diretamente do teclado para dentro do programa (como a mudança de linha) são escritos em C, como a combinação do sinal \ (barra invertida) com outros caracteres. Por exemplo, **\n** representa a mudança de linha.

Vamos agora escrever um programa com mais de uma instrução:

```
main()
{
   printf("A letra %c ",'j');
   printf("pronuncia-se %s.","jota");
}
```

A saída será:

A letra j pronuncia-se jota.

'j' é um caractere e "jota" é uma cadeia de caracteres.

Note que 'j' é delimitado por aspas simples enquanto que "jota" é delimitado por aspas duplas. Isto indica ao compilador como diferenciar um caractere de uma cadeia de caracteres. Observe também que a saída é feita em duas linhas de programa, o que não constitui duas linhas impressas de texto. A função **printf()** não imprime automaticamente um caractere de nova linha; se você desejar, deve inserir um explicitamente.

A tabela seguinte mostra os códigos de C para caracteres que não podem ser inseridos diretamente do teclado. A função **printf()** aceita todos eles.

CÓDIGOS ESPECIAIS	SIGNIFICADO
\n	NOVA LINHA
\r	RETORNO DO CURSOR
\t	TAB
\b	RETROCESSO
\"	ASPAS
\\	BARRA
\f	SALTA PÁGINA DE FORMULÁRIO
\0	NULO

A próxima tabela mostra os códigos para impressão formatada de **printf()**.

CÓDIGO printf()	FORMATO
%c	CARACTERE SIMPLES
%d	DECIMAL
%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)

CONSTANTES E VARIÁVEIS

Uma constante tem valor fixo e inalterável.

Nos exemplos anteriores, mostramos o uso de constantes numéricas, cadeias de caracteres e caracteres em **printf()**.

Em C uma constante caractere é escrita entre aspas simples, uma constante cadeia de caracteres entre aspas duplas e constantes numéricas como o número propriamente dito.

Exemplos de constantes:

'c'
"primeiro programa"
8

O programa

```
main()
{
   printf("Este e' o numero dois: %d",2);
}
```

imprime a constante 2, no formato especificado, %d.

Certamente esta não é a maneira mais simples de obter o mesmo resultado:

```
main()
{
   printf("Este e' o numero dois: 2");
}
```

As **variáveis** são o aspecto fundamental de qualquer linguagem de computador. Uma variável em C é um espaço de memória reservado para armazenar um certo tipo de dado e tendo um nome para referenciar o seu conteúdo.

O espaço de memória de uma variável pode ser compartilhado por diferentes valores segundo certas circunstâncias. Em outras palavras, uma variável é um espaço de memória que pode conter, a cada tempo, valores diferentes.

Para explicar o uso de variáveis vamos reescrever o programa anterior usando uma variável ao invés de uma constante:

```
main()
{
   int num;
   num=2;
   printf("Este e' o numero dois: %d",num);
}
```

A execução deste programa é exatamente a mesma que a da versão anterior porém ele cria a variável **num**, atribui a ela o valor 2 e imprime o valor contido nela.

A primeira instrução,

```
int num;
```

é um exemplo de declaração de variável, isto é, apresenta um tipo, **int**, e um nome, **num**.

A segunda instrução,

```
num=2;
```

atribui um valor à variável e este valor será acessado através de seu nome. Usamos o operador de atribuição (=) para este fim.

A terceira instrução chama a função **printf()** mandando o nome da variável como argumento, substituindo a constante 2 usada anteriormente.

DECLARAÇÕES DE VARIÁVEIS

Uma declaração de variável é uma instrução para reservar uma quantidade de memória apropriada para armazenar o tipo especificado, neste caso **int**, e indicar que o seu conteúdo será referenciado pelo nome dado, neste caso **num**.

```
int num;
```

UMA DECLARAÇÃO DE VARIÁVEL CONSISTE NO NOME DE UM TIPO, SEGUIDO DO NOME DA VARIÁVEL.

Em C todas as variáveis devem ser declaradas.

Se você tiver mais de uma variável do mesmo tipo, poderá declará-las de uma única vez, separando seus nomes por vírgulas.

```
int aviao, foguete, helicoptero;
```

POR QUE DECLARAR VARIÁVEIS

- Reunir variáveis em um mesmo lugar, dando a elas nomes significativos, facilita ao leitor entender o que o programa faz.
- Uma seção de declarações de variáveis encoraja o planejamento do programa antes de começar a escrevê-lo. Isto é, planejar as informações que devem ser dadas ao programa e quais as que o programa deverá nos fornecer.
- Declarar variáveis ajuda a prevenir erros. Por exemplo, se escrevermos 0 (zero) em vez de O:

```
int BOBO;
B0B0 = 5;
```

  o compilador acusará o erro.

- Se os motivos anteriores não convenceram você, este deverá convencê-lo:

C NÃO TRABALHA SE VOCÊ NÃO DECLARAR SUAS VARIÁVEIS.

TIPOS DE VARIÁVEIS

O tipo de uma variável informa a quantidade de memória, em bytes, que esta irá ocupar e a forma como o seu conteúdo será armazenado.

Em C existem 5 tipos de variáveis básicas. Nos computadores da linha IBM-PC a tabela seguinte é válida:

TIPO	BIT	BYTES	ESCALA
char	8	1	-128 a 127
int	16	2	-32768 a 32767
float	32	4	3.4E-38 a 3.4E+38
double	64	8	1.7E-308 a 1.7E+308
void	0	0	sem valor

Com exceção de **void**, os tipos de dados básicos podem estar acompanhados por modificadores na declaração de variáveis. Os modificadores de tipos oferecidos por C são:

long ou long int (4 bytes)
unsigned char (de 0 a 255)
unsigned int (de 0 a 65535)
unsigned long
short (2 bytes no IBM-PC)

O tipo **short** tem tamanho diferente do tipo **int** em outros computadores (no computador IBM-370, por exemplo, o tipo **short** tem a metade do tamanho de um inteiro).

O tipo **int** tem sempre o tamanho da palavra da máquina, isto é, em computadores de 16 bits ele terá 16 bits de tamanho.

Vamos examinar um programa que usa variáveis caractere, ponto flutuante e inteiras. Chamaremos este programa de *evento.c*.

```
main()
{
   int evento;
   char corrida;
   float tempo;
   evento=5;
   corrida='C';
   tempo=27.25;
   printf("O tempo vitorioso na eliminatoria %c",corrida);
   printf("\nda competicao %d foi %f.",evento,tempo);
}
```

A saída será:

O tempo vitorioso na eliminatoria C
da competicao 5 foi 27.25.

Este programa usa os 3 tipos de variáveis mais comuns: **int**, **char** e **float**.

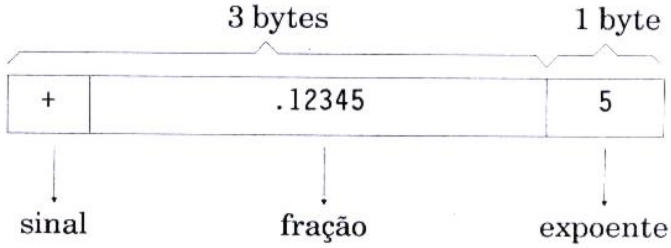
VARIÁVEIS PONTO FLUTUANTE

Números em ponto flutuante correspondem mais ou menos aos que os matemáticos chamam de "números reais".

Existem várias maneiras de escrever números em ponto flutuante. A notação "3.16e7" é um meio de indicar que 3.16 será multiplicado por 10 elevado a potência 7, isto é, 31600000. Esta indicação chama-se **notação científica** e é usada para armazenar números em ponto flutuante na memória do computador.

Assim, números em ponto flutuante são guardados na memória em duas partes. Estas duas partes são chamadas de **mantissa** e **expoente**. A mantissa é o valor do número e o expoente é a potência que irá aumentá-lo.

Por exemplo, 12345 é representado por .12345e5 onde o número que segue o (e) é o expoente, e .12345 é o valor do número.



Variáveis tipo **float** são guardadas em 4 bytes; um para o expoente e 3 para o valor do número e o sinal.

INICIALIZANDO VARIÁVEIS

É possível combinar uma declaração de variável com o operador de atribuição para que a variável tenha um valor ao mesmo tempo de sua declaração; é o que chamaremos de inicialização de variável. Como exemplo reescreveremos o programa *evento.c* inicializando as variáveis na sua declaração.

```
main()
{
   int evento=5;
   char corrida='C';
   float tempo=27.25;
   printf("O tempo vitorioso na eliminatoria %c",corrida);
   printf("\nda competicao %d foi %f.",evento,tempo);
}
```

A execução do programa será exatamente a mesma do da versão anterior.

INTEIROS COM E SEM SINAL

O modificador de tipo **unsigned** indica que o tipo associado deve ter seu bit de ordem superior interpretado de maneira diferente. Observe o programa a seguir:

```
main()
{
   unsigned int j=65000;
   int i=j;
   printf("%d %u \n", i, j);
}
```

O resultado será:

-536 65000

A razão disto está na maneira como o computador interpreta o bit de ordem superior do inteiro, ou seja, o bit 15.

Na forma binária o bit 15 de um inteiro positivo é sempre 0 e o de um inteiro negativo é sempre 1. Se usarmos o modificador **unsigned** em nossos programas, o computador irá ignorar o bit de sinal tratando-o como um bit a mais para números positivos.

Os números negativos são conhecidos como **complemento de dois** dos números positivos, pois a conversão de um número positivo para o seu negativo é feita por um processo de duas etapas. No momento você não precisa entender bem como é feita esta conversão. No segundo volume deste livro voltaremos a este assunto.

No nosso exemplo usamos o número 65000 que é maior que o maior inteiro interpretado com sinal, o que deixa evidente que o seu bit 15 é 1.

NOMES DE VARIÁVEIS

A escolha de nomes significativos para suas variáveis pode ajudá-lo a entender o que o programa faz e prevenir erros.

Você pode usar quantos caracteres quiser para um nome de variável com o primeiro sendo obrigatoriamente uma letra ou o caractere de sublinhar e os demais podendo ser letras, números ou caracteres de sublinhar.

Uma variável não pode ter o mesmo nome de uma palavra-chave de C.

Em C letras minúsculas e maiúsculas são diferentes.

PESO
Peso
peso
peSo

Em C os 32 primeiros caracteres são significativos, mas o sistema operacional MS-DOS limita os nomes em 8 caracteres significativos, isto é:

Altura_de_Maria
Altura_de_Pedro

são considerados o mesmo nome em programas processados sob o sistema operacional MS-DOS.

PALAVRAS-CHAVES EM 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

EXPLORANDO A FUNÇÃO printf()

Como já vimos, a função **printf()** usa o caractere % seguido de uma letra para identificar o formato de impressão. O problema surge quando queremos imprimir o caractere %. Se usarmos simplesmente % na expressão de controle de **printf()**, alguns compiladores acharão que não especificamos o formato corretamente e acusarão um erro. O caminho é usar dois símbolos %.

```
main()
{
   int reajuste = 10;
   printf("O reajuste foi de %d%%.\n",reajuste);
}
```

A saída será:

O reajuste foi de 10%.

TAMANHO DE CAMPOS NA IMPRESSÃO

Em **printf()** é possível estabelecer o tamanho mínimo para a impressão de um campo.

```
main()
{
   printf("Os alunos sao %2d.\n",350);
   printf("Os alunos sao %4d.\n",350);
   printf("Os alunos sao %5d.\n",350);
}
```

A saída será:

Os alunos sao 350.
Os alunos sao 350.
Os alunos sao   350.

Pode-se usar tamanho de campos com números em ponto flutuante para obter precisão e arredondamento.

```
main()
{
   printf("%4.2f\n",3456.78);
   printf("%3.2f\n",3456.78);
   printf("%3.1f\n",3456.78);
   printf("%10.3f\n",3456.78);
}
```

A saída será:

```
3456.78
3456.78
3456.8
  3456.780
```

Os tamanhos de campos podem ser usados para alinhamento à direita ou à esquerda. Exemplos:

```
main()
{
   printf("%.2f %.2f %.2f\n",8.0,15.3,584.13);
   printf("%.2f %.2f %.2f\n",834.0,1500.55,4890.21);
}
```

```
8.00 15.30 584.13
834.00 1500.55 4890.21
```

```
main()
{
   printf("%10.2f %10.2f %10.2f\n",8.0,15.3,584.13);
   printf("%10.2f %10.2f %10.2f\n",834.0,1500.55,4890.21);
}
```

```
      8.00      15.30     584.13
    834.00    1500.55    4890.21
```

O sinal de menos (–) precedendo a especificação do tamanho do campo justifica os campos à esquerda, como mostra o próximo programa:

```
main()
{
   printf("%-10.2f %-10.2f %-10.2f\n",8.0,15.3,584.13);
```

```
   printf("%-10.2f %-10.2f %-10.2f\n",834.0,1500.55,4890.21);
}
```

8.00 15.30 584.13
834.00 1500.55 4890.21

Este formato é muito útil em certas circunstâncias, especialmente quando imprimimos cadeia de caracteres.

COMPLEMENTANDO COM ZEROS À ESQUERDA

Além de especificar o tamanho do campo, podemos complementar o campo todo ou parte dele com zeros à esquerda. Observe o exemplo a seguir.

```
main()
{
   printf("\n%04d",21);
   printf("\n%06d",21);
   printf("\n%6.4d",21);
   printf("\n%6.0d",21);
}
```

A saída será:

0021
000021
0021
21

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.

Por exemplo, a instrução:

```
printf("%d %c %x %o \n",'A','A','A','A');
```

imprime

65 A 41 101

e a instrução

```
printf("%c %c %c %c \n",'A',65,0x41,0101);
```

imprime

A A A A

Um octal em C sempre é iniciado por 0 e um hexadecimal por 0x para que o compilador saiba diferenciá-los de números decimais.

A tabela ASCII tem 256 códigos decimais numerados de 0 a 255. Se imprimirmos em formato caractere um número maior que 255, o computador imprimirá o equivalente ao resto da divisão do número por 256, ou seja, se o número for 3393 ele será impresso como 'A' pois o resto da divisão de 3393 por 256 é 65.

```
printf("%d %c \n",3393,3393);
```

3393 A

IMPRIMINDO CARACTERES GRÁFICOS

Como você provavelmente sabe, todo caractere (letra, dígito, caractere de pontuação etc...) é representado, no computador, por um número. O código ASCII dispõe de números de 0 a 127 (decimal) cobrindo letras, dígitos de 0 a 9, caracteres de pontuação e caracteres de controle como salto de linha, tabulação etc...

Os computadores IBM usam 128 caracteres adicionais com códigos de 128 a 255 que consistem em símbolos de línguas estrangeiras e caracteres gráficos.

Já mostramos como imprimir caracteres ASCII usando a função **printf()**. Os caracteres gráficos e outros não standard requerem uma

outra maneira de escrita para serem impressos. A forma de se representar um caractere de código acima de 127 decimal é:

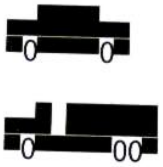
\xdd

onde **dd** representa o código do caractere em notação hexadecimal. Observe que **\xdd** é um caractere e pode ser usado na expressão de controle de **printf()** como qualquer outro caractere.

Usaremos este formato para impressão de qualquer caractere gráfico.

O programa a seguir imprime um carro e uma caminhonete usando caracteres gráficos:

```
main()
{
   printf("\n\n\n");
   printf("\n \xDC\xDC\xDB\xDB\xDB\xDB\xDC\xDC");
   printf("\n \xDF0\xDF\xDF\xDF\xDF0\xDF");
   printf("\n\n\n");
   printf("\n \xDC\xDC\xDB \xDB\xDB\xDB\xDB\xDB\xDB");
   printf("\n \xDF0\xDF\xDF\xDF\xDF\xDF00\xDF");
   printf("\n\n\n");
}
```



O próximo exemplo será denominado *box.c* e imprimirá uma moldura na tela.

```
main()
{
   printf("\xC9\xCD\xBB\n");
   printf("\xBA \xBA\n");
   printf("\xC8\xCD\xBC\n");
}
```


A saída será:

```
   C9  CD  BB
BA                 BA
   C8  CD  BC
```

EXERCÍCIOS

1. Um dos alunos preparou o seguinte programa e o apresentou para ser avaliado. Ajude-o.

```
main{}
(
   printf(Existem %d semanas no ano.,56);
)
```

2. O programa seguinte tem vários erros em tempo de compilação. Execute-o e observe as mensagens apresentadas por seu compilador.

```
Main()
{
   int a=1; b=2, c=3;
   printf("Os numeros sao: %d %d %d\n,a,b,c,d)
}
```

3. Qual será a saída do programa abaixo:

```
main()
{
   printf("%s\n%s\n%s","um","dois","tres");
}
```

4. Qual será a impressão obtida por cada uma destas instruções? Assuma que fazem parte de um programa completo.

a) printf("Bom Dia ! Shirley.");
printf("Voce ja tomou cafe ?\n");

b) printf("A solucao nao existe!\nNao insista");

c) printf("Duas linhas de saida\nou uma ?");

5. Identifique o tipo das seguintes constantes:

a)	'\r'	*b)*	2130	*c)*	-123
d)	33.28	*e)*	0x42	*f)*	0101
g)	2.0e30	*h)*	'\xDC'	*i)*	'\''

j) '\\' *k)* 'F' *l)* 0
m) '\0'

6. O que é uma variável em C ?

7. Quais os 5 tipos básicos de variáveis em C ?

8. Quais dos seguintes nomes são válidos para variáveis em C?

a) 3ab	*b)* _sim	*c)* n_a_o
d) 00FIM	*e)* int	*f)* A123
g) x**x	*h)* ___A	*i)* y-2
j) OOFIM	*k)* \meu	*l)* *y2

9. Quais das seguintes instruções são corretas ?

a) int a;
b) float b;
c) double float c;
d) unsigned char d;
e) long float e;

10. O tipo **float** ocupa o mesmo espaço que _____ variáveis do tipo char.

11. *Verdadeiro ou Falso*: tipos de variáveis **long int** podem conceber números não maiores que o dobro da maior variável do tipo **int**.

12. Escreva um programa que contenha uma única instrução e imprima na tela:

Esta e' a linha um.
Esta e' a linha dois.

13. Escreva um programa que imprima na tela:

um
dois
tres

14. Escreva um programa que declare 3 variáveis inteiras e atribua os valores 1,2 e 3 a elas; 3 variáveis caracteres e atribua a elas as letras a, b, e c; finalmente imprima na tela:

As variaveis inteiras contem os numeros 1, 2, e 3.
As variaveis caracteres contem os valores a, b, e c.

15. Reescreva o programa *box.c* para que desenhe uma moldura similar, mas que tenha 4 caracteres de largura e 4 caracteres de altura. Use o caractere ‖, de código BA hexa, para complementar a moldura.

OPERADORES ARITMÉTICOS

C é uma linguagem rica em operadores, em torno de 40. Alguns são mais usados que outros, como é o caso dos operadores aritméticos que executam operações aritméticas.

C oferece 6 operadores aritméticos binários (operam sobre dois operandos) e um operador aritmético unário (opera sobre um operando). São eles:

Binários

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

Unário

–	Menos unário

OPERADOR DE ATRIBUIÇÃO: =

Em C, o sinal de igual não tem a interpretação dada em matemática. Representa a atribuição da expressão à direita ao nome da variável à esquerda. Por exemplo:

```
num = 2000;
```

atribui o valor 2000 à variável *num*. A ação é executada da direita para a esquerda deste operador.

Observe que:

```
2000 = num
```

é uma igualdade válida em matemática mas que não tem sentido em C pois não podemos atribuir um valor a uma constante.

C aceita várias atribuições numa mesma instrução:

```
laranjas=cenouras=abacates=80;
```

OPERADORES : + - / *

Estes operadores representam as operações aritméticas básicas de soma, subtração, divisão e multiplicação.

A seguir está um programa que usa vários operadores aritméticos e converte temperatura Fahrenheit em seus correspondentes graus Celsius.

```
main()
{
   int ftemp,ctemp;
   printf("Digite temperatura em graus Fahrenheit: ");
   scanf("%d",&ftemp);
   ctemp=(ftemp-32) * 5/9;
   printf("Temperatura em graus Celsius e' %d",ctemp);
}
```

Eis um exemplo:

```
C>ftemp
Digite temperatura em graus Fahrenheit: 32
Temperatura em graus Celsius e' 0.
C>ftemp
Digite temperatura em graus Fahrenheit: 70
Temperatura em graus Celsius e' 21.
```

Vamos analisar a instrução:

```
ctemp=(ftemp-32) * 5/9;
```

Note que colocamos parênteses em **ftemp-32**. Se você lembra um pouco de álgebra, a razão estará clara. Nós queremos que 32 seja subtraído de ftemp antes de multiplicarmos por 5 e dividirmos por 9. A multiplicação e a divisão são feitas antes da soma ou subtração.

O programa a seguir usa um algoritmo interessante para adivinhar a soma de 5 números. O usuário digita um número qualquer e o computador informa o resultado da soma dos 5 números dos quais o primeiro o usuário já forneceu. O usuário digita o segundo número e o computador mostra o terceiro. O usuário digita o quarto número e o computador mostra o quinto.

Eis a listagem.

```
main()
{
   int x,r;
   printf("Digite um numero de ate 4 algarismos\n");
   scanf("%d",&x);
   r = 19998 + x;
   printf("O resultado da nossa conta sera: %d\n",r);
   printf("Digite o segundo numero (4 algarismos)\n");
   scanf("%d",&x);
   printf("O meu numero e': %d\n",9999-x);
   printf("Digite o quarto numero (4 algarismos)\n");
   scanf("%d",&x);
   printf("O meu numero e': %d\n",9999-x);
}
```

Eis um exemplo de sua execução:

Digite um numero de ate 4 algarismos
198
O resultado da nossa conta sera: 20196
Digite o segundo numero (4 algarismos)
1234
O meu numero e': 8765
Digite o quarto numero (4 algarismos)
2233
O meu numero e': 7766

OPERADOR MENOS UNÁRIO: -

O operador menos unário é usado somente para indicar a troca do sinal algébrico do valor. Pode também ser pensado como o operador que multiplica seu operando por -1. Por exemplo:

num = -8;
num1 = -num;

Depois destas duas instruções, o conteúdo de **num1** será 8.

OPERADOR MÓDULO: %

O operador módulo aceita somente operandos inteiros. Resulta o resto da divisão do inteiro à sua esquerda pelo inteiro à sua direita.

Por exemplo, 17%5 tem o valor 2 pois quando dividimos 17 por 5 teremos resto 2.

É também possível incluir expressões envolvendo operadores aritméticos (e outros operadores) diretamente em **printf()**.

Na verdade, uma expressão completa pode ser usada em quase todos os lugares onde uma variável pode ser usada. Como exemplo, vamos modificar o programa anterior usando uma expressão em **printf()** ao invés da variável.

```
main()
{
   int ftemp;
   printf("Digite temperatura em graus Fahrenheit: ");
   scanf("%d",&ftemp);
   printf("Temper. em graus Celsius e' %d",(ftemp-32)* 5/9);
}
```

A FUNÇÃO scanf()

A função **scanf()** é outra das funções de E/S implementadas em todos os compiladores C. Ela é o complemento de **printf()** e nos permite ler dados formatados da entrada padrão (teclado).

Sua sintaxe é similar à de **printf()**, isto é, uma expressão de controle seguida por uma lista de argumentos separados por vírgulas.

A principal diferença está na lista de argumentos. Os argumentos de **scanf()** devem ser endereços de variáveis.

Sintaxe:

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

A expressão de controle pode conter códigos de formatação, precedidos por um sinal % ou ainda o caractere * colocado após o % que avisa à função que deve ser lido um valor do tipo indicado pela especificação, mas não deve ser atribuído a nenhuma variável (não deve ter parâmetros na lista de argumentos para estas especificações).

A lista de argumentos deve consistir nos endereços das variáveis. C oferece um operador para tipos básicos chamado **operador de endereço** e referenciado pelo símbolo **&** que resulta o endereço do operando.

O OPERADOR DE ENDEREÇO (&)

A memória de seu computador é dividida em bytes, e estes bytes são numerados de 0 até o limite de memória de sua máquina (524.287 se você tem 512K de memória). Estes números são chamados de "endereços" de bytes. Um endereço é o nome que o computador usa para identificar a variável.

Toda variável ocupa uma certa localização na memória, e seu endereço é o do primeiro byte ocupado por ela. Um inteiro ocupa 2 bytes. Se você declarou a variável **n** como inteira e atribuiu a ela o valor 2, quando **n** for referenciada devolverá 2. Entretanto, se você referenciar **n** precedido de **& (&n)** devolverá o endereço do primeiro byte onde **n** está guardada.

O programa seguinte imprime o valor e o endereço de **n**:

```
main()
{
   int num;
   num=2;
   printf("Valor=%d, endereco=%u",num,&num);
}
```

Um endereço de memória é visto como um número inteiro sem sinal, por isso usamos **%u**.

A saída deste programa varia conforme a máquina e a memória do equipamento, um exemplo é:

Valor=2, endereco=1370

CÓDIGO DE FORMATAÇÃO DA FUNÇÃO scanf()

CÓDIGO	FUNÇÃO
%c	Leia um único caractere
%d	Leia um inteiro decimal
%e	Leia um número em notação científica
%f	Leia um número em ponto flutuante
%o	Leia um inteiro octal
%s	Leia uma série de caracteres
%x	Leia um número hexadecimal
%u	Leia um decimal sem sinal
%l	Leia um inteiro longo
%lf	Leia um double

Vamos escrever um programa para exemplificar a função **scanf()**, que chamaremos de *idade.c*.

```
main()
{
   float anos,dias;
   printf("Digite sua idade em anos: ");
   scanf("%f",&anos);
   dias = anos*365;
   printf("Sua idade em dias e' %.0f.\n",dias);
}
```

Eis uma execução do programa:

```
C>idade
Digite sua idade em anos: 4
Sua idade em dias e' 1460.
```

Visto que usamos variáveis float podemos entrar com frações decimais:

```
C>idade
Digite sua idade em anos: 12.5
Sua idade em dias e' 4562.
```

Neste programa utilizamos variável **float** ao invés de **int** para entrar com frações decimais em anos e para obter números maiores para dias.

O próximo programa, além de mostrar um outro uso de **scanf()**, mostra também a saída formatada **printf()**.

```
main()
{
   char a;
   printf("Digite um caractere e veja-o em decimal,");
   printf(" octal e hexadecimal.\n");
   scanf("%c",&a);
   printf("\n%c=%d dec.,%o oct. e %x hex.\n",a,a,a,a);
}
```

Eis uma execução do programa:

```
C>codchar
Digite um caractere e veja-o em decimal, octal e hexadecimal.
m
m=109 dec.,155 oct. e 6D hex.
```

AS FUNÇÕES getche() e getch()

Em algumas situações, a função **scanf()** não se adapta perfeitamente pois você precisa pressionar **[enter]** depois da sua entrada para que **scanf()** termine a leitura.

A biblioteca de C oferece funções que lêem um caractere no instante em que é datilografado, sem esperar **[enter]**.

A função **getche()** lê o caractere do teclado e permite que seja impresso na tela. Esta função não aceita argumentos e devolve o caractere lido para a função que a chamou.

O programa seguinte chama a função **getche()** e atribui o caractere que ela devolve à variável *ch* para depois imprimi-lo com **printf()**.

```
main()
{
   char ch;
   printf("Digite algum caractere: ");
   ch=getche();
   printf("\n A tecla que voce pressionou e' %c.",ch);
}
```

Eis a execução:

```
C>Ecoa
Digite algum caractere: a
A tecla que voce pressionou e' a.
```

A função **getch()** lê o caractere do teclado e não permite que seja impresso na tela. Como **getche()**, esta função não aceita argumentos e devolve o caractere lido para a função que a chamou.

O programa seguinte chama a função **getch()** e atribui o caractere que ela devolve à variável *ch* para depois imprimir o caractere e o seu sucessor na tabela ASCII com **printf()**.

```
main()
{
   char ch;
   printf("Digite algum caractere: ");
   ch=getch();
   printf("\n A tecla que voce pressionou e' %c",ch);
   printf(" e a sua sucessora ASCII e' %c.",ch+1);
}
```

Eis a execução:

```
C>necoa
Digite algum caractere:
A tecla que voce pressionou e' a e a sua sucessora ASCII e' b.
```

A biblioteca padrão provê outras funções para a leitura e escrita de um caractere por vez.

A FUNÇÃO getchar()

A função **getchar()** está definida no arquivo *stdio.h*, que acompanha seu compilador. Obtém o próximo caractere da entrada cada vez que é chamada, só terminando a leitura quando é pressionada a tecla **[enter]**, e retorna o caractere como seu valor. A função **getchar()** não aceita argumentos. Isto é, após

```
c = getchar();
```

a variável **c** contém o próximo caractere da entrada. Caso encontre a indicação do fim de arquivo **getchar()** retorna -1.

A FUNÇÃO putchar()

A função **putchar()** é o complemento de **getchar()** e também está definida no arquivo *stdio.h*. A função **putchar()** aceita um argumento cujo valor será impresso. Os comandos a seguir mostram como ler um caractere da entrada, atribuir seu valor à variável **c** e imprimir o conteúdo da variável **c** na saída padrão.

```
c = getchar();
putchar(c);
```

Um argumento de uma função pode ser outra função. Por exemplo, as linhas anteriores podem ser escritas como:

```
putchar(getchar());
```

Exercícios:

01 – Escreva um programa em C que leia dois números inteiros e mostre-os em ordem inversa no final.

02 – Escreva um programa em C que leia um número real e um número inteiro, faça a divisão do número real pelo inteiro e mostre o resultado.

03 – Escreva um programa C que leia 4 caracteres e mostre-os em seqüência como se fossem uma única palavra.

04 – Um carro percorreu 3 quilômetros em 90 segundos. Calcule a velocidade média do carro em Km/h nesse percurso, sabendo que 1 m/s corresponde a 3,6 Km/h e que Vm = E / t.

05 – Modifique o programa anterior para que aceite qualquer quilometragem e qualquer tempo gasto (em segundos) para calcular a velocidade média.

TREINAMENTO EM LINGUAGEM C – VIVIANE VICTORINE

FUNDAMENTOS DA PROGRAMAÇÃO DE COMPUTADORES – Ana Fernanda Gomes Ascencio

C – Completo e Total

DEV C++ – Vers. 5.0