

## 4. Entrada e Saída

Para que um programa torne-se minimamente funcional é preciso que ele receba dados do meio externo (teclado, mouse, portas de comunicação, drives de disco, etc.) e emita o resultado de seu processamento de volta para o meio externo (monitor, impressora, alto-falante, portas de comunicação, drives de disco, etc.). De outro modo: um programa deve trocar informações com o meio externo. Em C, existem muitas funções pré-definidas que tratam desta troca de informações. São as funções de **entrada e saída** do C. Nos exemplos mostrados nos capítulos anteriores foram vistas algumas funções de entrada (`scanf()`, `getch()`) e algumas funções de saída (`printf()`). Neste capítulo veremos, em detalhe, estas e outras funções de modo a permitir escrever um programa completo em C.

Mostraremos, nas duas seções iniciais as mais importantes funções de entrada e saída de dados em C: as funções `printf()` e `scanf()`. A partir do estudo destas funções é possível escrever um programa propriamente dito com entrada, processamento e saída de dados.

### 4.1 Saída formatada: `printf()`

**Biblioteca:** `stdio.h`

**Declaração:** `int printf (const char* st_cont r [, lista_arg]);`

**Propósito:** A função `printf()` (*print formatted*) imprime dados da lista de argumentos `lista_arg` formatados de acordo com a string de controle `st_cont r`. Esta função retorna um valor inteiro representando o número de caracteres impressos.

Esta função imprime dados numéricos, caracteres e *strings*. Esta função é dita de saída formatada pois os dados de saída podem ser formatados (alinhados, com número de dígitos variáveis, etc.).

**Sintaxe:** A *string de controle* é uma máscara que especifica (formata) o que será impresso e de que maneira será impresso.

**Exemplo:** Observe no exemplo abaixo as *instruções* de saída formatada e os respectivos resultados.

<b>Instrução</b>	<b>Saída</b>
<code>printf("Ola', Mundo!");</code>	Ola', Mundo!
<code>printf("linha 1 \nlinha 2 ");</code>	linha 1
	linha 2

Observe que na primeira instrução, a saída é exatamente igual a string de controle. Já na segunda instrução a impressão se deu em duas linhas. Isto se deve ao `\n` que representa o código ASCII para quebra de linha (veja seção 2.1.3).

Nesta mascara é possível reservar espaço para o **valor** de alguma variável usando *especificadores de formato*. Um especificador de formato marca o **lugar** e o **formato** de impressão das variáveis contidas na **lista variáveis**. Deve haver um especificador de formato para cada variável a ser impressa. Todos os especificadores de formato começam com um `%`.

**Exemplo:** Observe no exemplo abaixo as *instruções* de saída formatada e os respectivos resultados. Admita que `idade` seja uma variável `int` contendo o valor 29 e que `tot` e `din` sejam variáveis `float` cujo valores são, respectivamente, 12.3 e 15.0.

**Instrução:**

```
printf("Tenho %d anos de vida",idade);
```

**Saída:**

Tenho 29 anos de vida

**Instrução:**

```
printf("Total: %f.2 \nDinheiro: %f.2 \nTroco: %f.2",tot,din,din-tot);
```

**Saída:**

Total: 12.30

Dinheiro: 15.00

Troco: 2.70

Depois do sinal `%`, seguem-se alguns modificadores, cuja sintaxe é a seguinte:

```
% [flag] [tamanho] [.precisão] tipo
```

**[flag] justificação de saída: (Opcional)**

- justificação à esquerda.
- + conversão de sinal (saída sempre com sinal: + ou -)

<espaço> conversão de sinal (saídas negativas com sinal, positivas sem sinal)

**[ *t* *amanho* ] especificação de tamanho (Opcional)**

n pelo menos n dígitos serão impressos (dígitos faltantes serão completados por brancos).

0n pelo menos n dígitos serão impressos (dígitos faltantes serão completados por zeros).

**[ . *precisão* ] especificador de precisão, dígitos a direita do ponto decimal. (Opcional)**

(nada) padrão: 6 dígitos para reais.

.0 nenhum dígito decimal.

.n são impressos n dígitos decimais.

**Tipo caracter de conversão de tipo (Requerido)**

d inteiro decimal

o inteiro octal

x inteiro hexadecimal

f ponto flutuante: [-]dddd.dddd.

e ponto flutuante com expoente: [-]d.dddd[+/-]ddd

c caracter simples

s string

**Programa Exemplo:** O arquivo `e0401.cpp` contém um programa que ilustra o uso da função `printf()` usando várias combinações de *strings de controle* e *especificadores de formato*. Execute o programa passo-a-passo e verifique a saída dos dados.

## 4.2 Leitura formatada: `scanf()`

**Biblioteca:** `stdio.h`

**Declaração:** `int scanf(const char* st_contr [, end_var, ...]);`

**Propósito:** A função `scanf()` (*scan formatted*) permite a entrada de dados numéricos, caracteres e 'strings' e sua respectiva atribuição a variáveis cujos endereços são `end_var`. Esta função é dita de entrada formatada pois os dados de entrada são formatados pela *string de controle* `st_contr` a um determinado tipo de variável (`int`, `float`, `char`, ...).

**Sintaxe:** O uso da função `scanf()` é semelhante ao da função `printf()`. A função lê da entrada padrão (em geral, teclado) uma lista de valores que serão formatados pela string de controle e armazenados nos endereços das variáveis da lista. A string de controle é formada por um conjunto de especificadores de formato, cuja sintaxe é a seguinte:

`% [ * ] [ tamanho ] tipo`

**\* indicador de supressão (Opcional)**

<presente> Se o indicador de supressão estiver presente o campo não é lido. Este supressor é útil quando não queremos ler um campo de dado armazenado em arquivo.

<ausente> O campo é lido normalmente.

**Tamanho especificador de tamanho(Opcional)**

`n` Especifica `n` como o número máximo de caracteres para leitura do campo.

<ausente> Campo de qualquer tamanho.

**Tipo define o tipo de dado a ser lido (Requerido)**

`d` inteiro decimal (`int`)

`f` ponto flutuante (`float`)

`o` inteiro octal (`int`)

`x` inteiro hexadecimal (`int`)

`i` inteiro decimal de qualquer formato (`int`)

`u` inteiro decimal sem sinal (`unsigned int`)

`s` string (`char*`)

`c` caracter (`char`)

A lista de variáveis é o conjunto de (endereços) de variáveis para os quais serão passados os dados lidos. Variáveis simples devem ser precedidos pelo caracter `&`. Veja mais sobre endereços na seção ?? Variáveis string e vetores não são precedidos pelo caracter `&`. Veja mais sobre strings e vetores na seção ??

**Programa exemplo:** O arquivo `e0402.cpp` contém um programa que ilustra o uso da função `scanf()` na leitura de dados. Execute o programa passo-a-passo e verifique como os especificadores de formato agem sobre os dados digitados.

### 4.3 Entrada de caracter individual: `getchar()`

**Biblioteca:** `stdio.h`

**Declaração:** `int getchar(void);`

**Propósito:** A função `getchar()` (*get character*) lê um caracter individual da entrada padrão (em geral, o teclado). Se ocorrer um erro ou uma condição de *'fim-de-arquivo'* durante a leitura, a função retorna o valor da constante simbólica `EOF` (*end of file*) definida na biblioteca `stdio.h`. Esta função permite uma forma eficiente de detecção de finais de arquivos.

Esta função é dita *line buffered*, isto é, não retorna valores até que o caracter de controle *line feed* (`\n`) seja lido. Este caracter, normalmente, é enviado pelo teclado quando a tecla `[enter]` é pressionada. Se forem digitados vários caracteres, estes ficarão armazenados no *buffer* de entrada até que a tecla `[enter]` seja pressionada. Então, cada chamada da função `getchar()` lerá um caracter armazenado no *buffer*.

#### 4.4 Saída de caracter individual: `putchar()`

**Biblioteca:** `stdio.h`

**Declaração:** `int putchar(int c);`

**Propósito:** Esta função `putchar()` (*put character*) imprime um caracter individual `c` na saída padrão (em geral o monitor de vídeo).

**Programa Exemplo:** O programa `e0403.cpp` mostra o uso das funções `getchar()` e `putchar()` em um programa que lê caracteres do teclado e os reimprime convertidos para maiúsculos.

#### 4.5 Leitura de teclado: `getch()`, `getche()`

**Biblioteca:** `conio.h`

**Declaração:** `int getch(void);`

`int getche(void);`

**Propósito:** Estas funções fazem a leitura dos códigos de teclado. Estes códigos podem representar teclas de caracteres (A, y, \*, 8, etc.), teclas de comandos (`[enter]`, `[delete]`, `[Page Up]`, `[F1]`, etc.) ou combinação de teclas (`[Alt] + [A]`, `[Shift] + [F1]`, `[Ctrl] + [Page Down]`, etc.).

Ao ser executada, a função `getch()` (*get character*) aguarda que uma tecla (ou combinação de teclas) seja pressionada, recebe do teclado o código correspondente e retorna este valor. A função `getche()` (*get character and echo*) também escreve na tela, quando possível, o caracter correspondente.

**Código ASCII:** ao ser pressionada uma tecla correspondente a um caracter ASCII, o teclado envia um código ao 'buffer' de entrada do computador e este código é lido. Por exemplo, se a tecla A for pressionada o código 65 será armazenado no *buffer* e lido pela função.

**Código Especial:** ao serem pressionadas certas teclas (ou combinação de teclas) que não correspondem a um caracter ASCII, o teclado envia ao 'buffer' do computador **dois** códigos, sendo o primeiro **sempre** 0. Por exemplo, se a tecla [F1] for pressionada os valores 0 e 59 serão armazenados e a função deve ser chamada **duas vezes** para ler os dois códigos.

**Programa exemplo:** O arquivo `e0404.cpp` mostra um programa para a leitura de teclado. Este programa usa a função `getch()` para reconhecer teclas e combinação de teclas.

**Programa exemplo:** O arquivo `e0405.cpp` mostra um programa para a leitura de teclado usando a função `getche()`.

## 4.6 Escrita formatada em cores: `cprintf()`

**Biblioteca:** `conio.h`

**Declaração:** `int cprintf (const char* st_contr [, lista_arg]);`

**Propósito:** Esta função `cprintf()` (*color print formatted*) permite a saída de dados numéricos, caracteres e strings usando cores. O uso da função `cprintf()` é semelhante a `printf()` porém permite que a saída seja a cores. Para que a saída seja colorida é necessário definir as cores de fundo e de letra para a impressão antes do uso da função.

**Cores ( Modo Texto)**

Cor	Constante	Valor	Fundo	Letra
Preto	BLACK	0	ok	ok
Azul	BLUE	1	ok	ok
Verde	GREEN	2	ok	ok
Cian	CYAN	3	ok	ok
Vermelho	RED	4	ok	ok
Magenta	MAGENTA	5	ok	ok
Marrom	BROWN	6	ok	ok
Cinza Claro	LIGHTGRAY	7	ok	ok
Cinza Escuro	DARKGRAY	8	--	ok
Azul Claro	LIGHTBLUE	9	--	ok
Verde Claro	LIGHTGREEN	10	--	ok
Cian Claro	LIGHTCYAN	11	--	ok
Vermelho Claro	LIGHTRED	12	--	ok
Magenta Claro	LIGHTMAGENTA	13	--	ok
Amarelo	YELLOW	14	--	ok
Branco	WHITE	15	--	ok
Piscante	BLINK	128	--	ok

Estas definições são feitas pelas funções `textcolor()` e `textbackground()` cuja sintaxe é:

```
textcolor(cor_de_letra);
textbackground(cor_de_fundo);
```

onde *cor\_de\_letra* e *cor\_de\_fundo* são números inteiros referentes as cores da paleta padrão (16 cores, modo texto). Estes valores de cor são representadas por constantes simbólicas definidas na biblioteca `conio.h`. Para se usar uma letra piscante deve-se adicionar o valor 128 ao valor da cor de letra. Alguns valores de cor não podem ser usados como cor de fundo. A relação acima mostra as cores, suas constantes simbólicas e onde podem ser usadas:

**Exemplo:** O trecho de programa abaixo imprime uma mensagem de alerta em amarelo piscante sobre fundo vermelho.

```
#include <conio.h>
...
textbackground(RED);
textcolor(YELLOW + BLINK);
cprintf(" Alerta: Vírus Detectado! ");
...
```

**Programa Exemplo:** O programa do arquivo `e0406.cpp` mostra todas as combinações possíveis de impressão colorida em modo texto.

## 4.7 Saída sonora: `sound()`, `delay()`, `nosound()`

**Biblioteca:** `dos.h`

**Declarações:** `void sound(unsigned freq);`  
`void delay(unsigned tempo);`  
`void nosound(void);`

**Propósito:** A função `sound()` ativa o alto-falante do PC com uma frequência `freq` (Hz). A função `delay()` realiza uma pausa (aguarda intervalo de tempo) de duração `tempo` (milisegundos). A função `nosound()` desativa o alto-falante.

**Programa Exemplo:** O uso destas funções é muito simples mas produz resultados interessantes. No arquivo `e0407.cpp` temos um exemplo do uso de sons em programas.

## 4.8 Limpeza de tela: `clrscr()`, `clreol()`

**Biblioteca:** `conio.h`

**Declarações:** `void clrscr(void);`  
`void clreol(void);`

**Propósito:** A função `clrscr()` (*clear screen*) limpa a janela de tela e posiciona o cursor na primeira linha e primeira coluna da janela (canto superior esquerdo da janela). A função `clreol()` (*clear to end of line*) limpa uma linha desde a posição do cursor até o final da linha mas não modifica a posição do cursor. Ambas funções preenchem a tela com a cor de fundo definida pela função `textbackground()`.

## 4.9 Posicionamento do cursor: `gotoxy()`

**Biblioteca:** `conio.h`



**Declarações:** `void gotoxy(int pos_x, int pos_y);`

**Propósito:** Em modo texto padrão, a tela é dividida em uma janela de 25 linhas e 80 colunas. A função `gotoxy()` permite posicionar o cursor em qualquer posição (`pos_x, pos_y`) da tela. Sendo que a posição (1,1) corresponde ao canto superior esquerdo da tela e a posição (80,25) corresponde ao canto inferior direito. Como as funções `printf()` e `cprintf()` escrevem a partir da posição do cursor, podemos escrever em qualquer posição da tela.

#### 4.10 Redimensionamento de janela: `window()`

**Biblioteca:** `conio.h`

**Declarações:** `void window(int esq, int sup, int dir, int inf);`

**Propósito:** Esta função permite redefinir a janela de texto. As coordenadas `esq` e `sup` definem o canto superior esquerdo da nova janela, enquanto as coordenadas `inf` e `dir` definem o canto inferior direito da nova janela. Para reativar a janela padrão escreve-se a instrução `window(1,1,80,25)`. Quando uma janela é definida, o texto que ficar fora da janela fica congelado até que se redefina a janela original.

#### 4.11 Monitoração de teclado: `kbhit()`

**Biblioteca:** `conio.h`

**Declarações:** `int kbhit(void);`

**Propósito:** Esta função (*keyboard hitting*) permite verificar se uma tecla foi pressionada ou não. Esta função verifica se existe algum código no *buffer* de teclado. Se houver algum valor, ela retorna um número não nulo e o valor armazenado no *buffer* pode ser lido com as funções `getch()` ou `getche()`. Caso nenhuma tecla seja pressionada a função retorna 0. Observe que, ao contrário de `getch()`, esta função **não aguarda** que uma tecla seja pressionada.

**Programa Exemplo:** O arquivo `e0408.cpp` contém um programa para exemplificar o uso das funções `clrscr()`, `clreol()`, `gotoxy()`, `window()`, `kbhit()`.