COMANDOS ESPECIAIS E DE TELA

Entrada de caractere individual: `getchar()`

Biblioteca: `stdio.h`

Sintaxe: `int getchar(void);`

A função `getchar()` (*get character*) lê um caractere individual da entrada padrão (em geral, o teclado).

Esta função não retorna valor até que o caractere de controle *line feed* (`\n`) seja lido. Este caractere, 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 caractere armazenado no *buffer*.

Exemplo:

```
/* ... */
do{
  printf ("Sexo (m/f)? ");
  sexo = getchar();
}while(sexo!='m' && sexo!='M' && sexo!='f' && sexo!='F');
/* ... */
```

Saída de caractere individual: `putchar()`

Biblioteca: `stdio.h`

Declaração: `int putchar(int c);`

A função `putchar()` (*put character*) imprime um caractere individual `c` na saída padrão (em geral o monitor de vídeo).

Exemplo:

```
/* ... */
for (i = 0; i < strlen(nome); i++) {
  putchar (nome[i]);
  putchar (' ');
}
/* ... */
```

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

Biblioteca: `conio.h`

Declaração:
```
int getch(void);
int getche(void);
```

As funções `getch()` e `getche()` fazem a leitura dos códigos de teclado. Estes códigos podem representar teclas de caracteres (`a` , `Z` , `*` , `6` , 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 echoe*) também escreve na tela, quando possível, o caractere correspondente.

Código ASCII: ao ser pressionada uma tecla correspondente a um caractere ASCII, o teclado envia um código ao '*buffer*' de entrada do computador e este código é lido.

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 caractere ASCII, o teclado envia ao '*buffer*' do computador dois códigos, sendo o primeiro sempre 0.

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.

Exemplo:
```
#include<stdio.h>
#include<conio.h>
#define esc 27
int main() {
  char t;
  printf ("Digite um caractere (letra,tecla,etc.)”);
  printf (“ - <esc> para finalizar:\n\n");
  do {
    t = getch();
    if (t <= 0) {
      t = getch ();
      printf ("   ->> %d %d \n",0,t);
    }
    else
      printf ("%c   -> %d \n",t,t);
  } while (t != esc);
}
```

Monitoração de teclado: `kbhit()`

Biblioteca: `conio.h`

Declaração: `int kbhit(void);`

A função `kbhit()` (*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.

Exemplo:

```
#include <stdio.h>
#include <conio.h>
int main() {
  char t;
  while (!kbhit()) {
    printf ("ALGORITMOS I    ");
  }
  t = getche();
  printf (" %d ",t);
}
```

Função `gotoxy ()`

Biblioteca: `conio.h`

Declaração: `void gotoxy(int coluna, int linha);`

```
//Defines gotoxy() for ANSI C compilers
void gotoxy(short x, short y) {
  COORD pos = {x, y};
  SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE),
pos);
}
```

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 `(coluna,linha)` 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 a função `printf()` escreve a partir da posição do cursor, pode-se escrever em qualquer posição da tela.

Exemplo:

```
#include <stdio.h>
#include <windows.h>
#include <string.h>
#include <conio.h>
void gotoxy (int x, int y) {
  COORD pos = {x, y};
  SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), pos);
}
int main (){
  int col, lin = 1, n;
  char texto[] = "Calculadora";
  n = strlen(texto);
  col = (80 - n) / 2;
  // cálculo da centralização do título na linha
  gotoxy (col,lin);
  printf ("%s", texto);
  getch ();
}
```

Cores

Pode-se alterar as cores de fundo e de texto dos programas em linguagem C, existem mais de um comando para esta função. Um comando que pode ser usado para trocar a cor de fundo e de texto de um programa em C é o:

```
system (“color corfundocortexto”);
```

Esse comando altera para todo um programa em C, a cor de fundo da janela e do texto mostrado, essa alteração vale para toda a execução do comando.

Para usá-lo deve-se incluir a biblioteca `stdlib.h`.

Exemplo:

```
#include <stdio.h>
#include <stdlib.h>
int main () {
  // coloca a cor de fundo como vermelho e o texto em azul
  system ("color C1");
  printf ("Texto na cor azul no fundo vermelho.\n\n");
  system ("pause");
}
```

Cores disponíveis

0 = Preto	8 = Cinza
1 = Azul	9 = Azul claro
2 = Verde	10 = Verde claro
3 = Verde-água	11 = Verde-água claro
4 = Vermelho	12 = Vermelho claro
5 = Roxo	13 = Lilás

6 = Amarelo 14 = Amarelo claro
7 = Branco 15 = Branco brilhante

As cores devem ser definidas pelos seus números em hexadecimal. Por exemplo, vermelho = C.

Outros comandos disponíveis para trabalhar com cores estão na biblioteca chamada `conio.c`, nela tem-se os comandos:

```
textbackground(corfundo);
```
e
```
textcolor(cortexto);
```

A vantagem destes comandos em relação ao `system("color")` é que com eles pode-se mudar a cor do texto e de fundo várias vezes durante o mesmo programa. Aceitam as mesmas variações de cores, e elas podem ser definidas por números de 0 a 15.

Exemplo:

```
#include <stdlib.h>
#include <stdio.h>
#include <windows.h>
int SetColor (char color) {
  HANDLE h;
  h = GetStdHandle (STD_OUTPUT_HANDLE);
  return SetConsoleTextAttribute (h,color);
}
int main () {
  int i;
  for (i = 0; i <= 127; i++) {
    SetColor (i);
    printf (" Cor %d ",i);
  }
  system ("pause");
}
```

Limpar tela

Outro comando importante em se tratando de tela, é o comando para limpar a tela,
tela e, existem dois comandos que podem ser usados para tal.

```
system ("cls");     // biblioteca <stdlib.h>
clrscr();           // biblioteca <conio.c>
```

Biblioteca `ctype.h`

`ctype.h` contém protótipos de funções e macros que permitem verificar se um determinado caractere é ASCII, se é numérico, se é maiúscula, minúscula, etc.

Considere `c` um caractere no formato inteiro, algumas funções de `ctype.h`:

`int isalnum(int c);`
Verifica se o caractere é alfanumérico, devolve zero (falso) se não for caractere alfanumerico e um valor não-zero se for.

`int isalpha(int c);`
Verifica se o caractere é uma letra (A-Z ou a-z) ou não, devolve zero se não for uma letra e um valor não-zero se for.

`int isdigit(int c);`
Verifica se o caractere é uma algarismo (0-9) ou não, devolve zero se não for um algarismo e um valor não-zero se for.

`int islower(int c);`
Verifica se o caractere é uma letra minúscula, devolve zero se não for (0 - falso) e um valor não-zero se for (!0 - verdadeiro).

`int isupper(int c);`
Verifica se o caractere é uma letra maiúscula, devolve zero se não for (0 - falso) e um valor não-zero se for (!0 - verdadeiro).

`int tolower(int c);`
Converte uma letra para o formato minúsculo.

`int toupper(int c);`
Converte uma letra para o formato maiúsculo.