

## Algoritmos e Programação de Computadores

### Guia ao *kbhit* e *getch*

---

## A utilidade

Enquanto estamos desenvolvendo algo, frequentemente em um jogo ou fazer um login de um usuário no seu programa, queremos que o programa funcione enquanto o usuário não faz nada ou mesmo que capture a entrada do usuário sem apertar enter. Para isso que servem as duas funções:

- *kbhit* - Essa função indica se existe uma entrada no buffer
- *getch* - Essa função captura um caracter do buffer sem que seja necessário apertar enter

### Login de usuário

Quando se faz o login do usuário, é interessante esconder a senha à medida que o usuário digite. Os programas feitos até aqui sempre que você digita qualquer caracter, ele fica exposto até você apertar enter (enviar para processamento). Logo, queremos que quando o usuário digite um **char**, o programa capture esse caracter e escreva o '\*' no seu lugar.

### Jogo 2048

No [Jogo 2048](#), quando se toma uma decisão (enviar as peças para algum lugar), se for necessário apertar **enter** para cada comando, o jogo fica extremamente ruim.

### Jogo Snake

No [Jogo Snake](#), enquanto o usuário não digita algo, a serpente se move até o usuário decidir fazer algo.

## Os requisitos

Ambas as funções foram testadas utilizando o compilador [gcc](#) versão 5.4.0 utilizando [Ubuntu](#) 16.04.4. Versões similares do gcc e na maioria das distribuições linux são capazes de compilar e executar, além de ter as bibliotecas ([headers](#)) requeridas.

#### **getch**

Para a função *getch* é necessário três *headers*:

- 1) stdio
- 2) termios
- 3)unistd

#### **kbhit**

Para a função *kbhit* é necessário quatro *headers*:

- 1) stdio
- 2) termios
- 3) unistd
- 4) fcntl

**Obs:** Não foram testadas estas funções para windows.

# O código

Por questões de complexidade de código, o código não será explicado. Mas como não são funções padrões então é necessário que tenhamos o código. O código é dado tanto pelo arquivo [kbhitgetch.h](#) quanto é mostrado abaixo.

## getch

```
1 char getch()
2 {
3     int ch;
4     struct termios oldt;
5     struct termios newt;
6     tcgetattr(STDIN_FILENO, &oldt);
7     newt = oldt;
8     newt.c_lflag &= ~(ICANON | ECHO);
9     tcsetattr(STDIN_FILENO, TCSANOW, &newt);
10    ch = getchar();
11    tcsetattr(STDIN_FILENO, TCSANOW, &oldt);
12    return ch;
13 }
```

## kbhit

```
1 int kbhit(void)
2 {
3     struct termios oldt, newt;
4     int ch;
5     int oldf;
6
7     tcgetattr(STDIN_FILENO, &oldt);
8     newt = oldt;
9     newt.c_lflag &= ~(ICANON | ECHO);
10    tcsetattr(STDIN_FILENO, TCSANOW, &newt);
11    oldf = fcntl(STDIN_FILENO, F_GETFL, 0);
12    fcntl(STDIN_FILENO, F_SETFL, oldf | O_NONBLOCK);
13
14    ch = getchar();
15
16    tcsetattr(STDIN_FILENO, TCSANOW, &oldt);
17    fcntl(STDIN_FILENO, F_SETFL, oldf);
18
19    if(ch != EOF)
20    {
21        ungetc(ch, stdin);
22        return 1;
23    }
24
25    return 0;
26 }
```

# O funcionamento

Agora explicaremos como funcionam as funções *getch* e *kbhit*

## getch

Essa função é semelhante ao [getchar](#). A diferença de uso de ambas as funções é que a *getch* não precisa esperar o usuário terminar de digitar e depois apertar enter.

Um exemplo de implementação é mostrado pelo arquivo [1-getch\\_example1.c](#), em que cada letra que é digitada é mostrada na tela até que aperte enter('\n').

```
1 #include <stdio.h>
2 #include "kbhitgetch.h"
3
4 int main()
5 {
6     char c;
7     do{
8         c = getch();
9         putchar(c);
10    }while(c != '\n');
11    return 0;
12 }
```

Bem, se você for bastante esperto, notará a diferença entre antes e depois. Caso ainda não tenha notado, primeiro digite alguns caracteres(no terminal) e tente apagá-los. Caso ainda não tenha notado a diferença, apague a linha 9 do código acima e teste.

Outra diferença que pode-se notar é que quando você digita um caracter quando há o *getch* é que ele não parece na tela. No exemplo acima, só apareceram os caracteres devido à linha 9 que contém um *putchar*.

Agora, outro exemplo([2-getch\\_example2.c](#)) para o caso do **login** que foi mostrado no início, é trocar na linha 9 o argumento de *c* para *''*. E então mostrará escondida a senha e informará ao usuário quantos caracteres. Mais abaixo iremos retomar este exemplo e como podemos melhorar.

## kbhit

Essa função verifica no *buffer* se existe alguma entrada. Caso haja alguma entrada, então a função retorna 1 e caso contrário retorna 0.

Um exemplo de implementação é mostrado pelo arquivo [3-kbhit\\_example1.c](#), em que imprime a mensagem **"Aperte uma tecla para parar"** até que se aperte uma tecla.

```
1 #include <stdio.h>
2 #include "kbhitgetch.h"
3
4 int main()
5 {
6     while(!kbhit()) /* Enquanto nao tem entrada */
7     {
8         printf("Aperte uma tecla para parar\n");
9     }
10    printf("Voce parou o processo apertando alguma tecla!\n\n");
11    return 0;
12 }
```

O importante é que essa função não captura o caracter digitado. Ela simplesmente verifica se tem entrada ou não. Essa entrada, assim como no *getch* reconhece antes de apertar enter.

Neste caso, se tiver entrada, podemos capturar essa entrada utilizando uma entrada de dados que pode ser *getchar* ou a função que acabamos de aprender que é *getch*.

Assim, no código abaixo(4-kbhit\_example2.c) está um uso das duas funções juntas que semelhantemente ao código acima, pede para parar o processo apertando uma tecla. A diferença é que agora iremos informar qual foi o caracter digitado.

```
1 #include <stdio.h>
2 #include "kbhitgetch.h"
3
4 int main()
5 {
6     while(!kbhit()) /* Enquanto nao tem entrada */
7     {
8         printf("Aperte uma tecla para parar\n");
9     }
10    printf("Voce parou o processo apertando a tecla: [%c]!\n\n", getch
        ());
11    return 0;
12 }
```

Agora, faremos um número começar do zero e para cada aperto em uma tecla, informará um número correspondente. O código é 5-kbhit\_example3.c

```
1 #include <stdio.h>
2 #include "kbhitgetch.h"
3
4 int main()
5 {
6     long long int contador = 0;
7     char c = 'a';
8     while(c != '\n') /* Enquanto o digitado nao for digitado o Enter */
9     {
10        contador += 1;
11        if(kbhit()) /* Se existe entrada */
12        {
13            c = getch(); /* Entao captura e grava na variavel c */
14            printf("contador = %d\n", contador); /* E imprime o numero
                correspondente */
15        }
16    }
17    return 0;
18 }
```

Neste código, quanto maior o intervalo entre uma tecla e outra, mais distante um número estará do outro.

## Aplicações com getch e kbhit

Como de costume, sempre que aprendemos algo novo queremos implementar. Você tem capacidade de implementar algumas coisas interessantes já, como por exemplo o [Jogo Snake](#). Mas tem algumas coisas interessantes que ainda não vimos...

### Teclas especiais

Como que captura uma seta para cima ou a tecla F2?

Vá no código [2-getch\\_example2.c](#) e digite a tecla [BACKSPACE](#) e veja o que acontece. Se você digitar, aparecerá um caracter que segundo o código [getch\\_example1.c](#) não existe. Bem, na verdade existe, a diferença é que quando você tenta imprimi-lo, imprime nada.

Agora, nesse mesmo código, digite a seta para cima ([as direcionais](#)) e verá que aparecerá três simbolos `''`. Ora, aprendemos algo inusitado, mas de princípio não sabemos o que é. Para auxiliar na nossa análise do que é, vamos em vez de imprimir `''`, iremos imprimir o próprio caracter. Contudo, como no caso do `BACKSPACE`, imprimindo o próprio caracter não é muito inteligente e neste caso imprimiremos o número correspondente. Para imprimir o número faremos o uso da tabela ASCII. Alguns caracteres e seus respectivos números no padrão ASCII estão no quadro:

Dec	Char	Dec	Char	Dec	Char	Dec	Char
48	0	58	:	65	A	97	a
49	1	59	;	66	B	98	b
50	2	60	<	67	C	99	c
51	3	61	=	68	D	100	d
52	4	62	>	69	E	101	e
53	5	63	?	86	V	118	v
54	6	64	@	87	W	119	w
55	7	42	*	88	X	120	x
56	8	43	+	89	Y	121	y
57	9	45	-	90	Z	122	z

Assim, um código que lê um char e escreve o número correspondente é dado por [6-aplicacao.c](#) que é mostrado abaixo:

```
1 #include <stdio.h>
2 #include "kbhitgetch.h"
3
4 int main()
5 {
6     char c;
7     do{
8         c = getch();
9         printf("%d\n", c);
10    }while(c != '\n');
11    return 0;
12 }
```

Assim você consegue descobrir os números correspondente segundo a tabela ASCII. Agora, o que acontece se apertar a seta para cima? Irá imprimir três números consecutivos que são 27, 91 e 65. Assim que funcionam as teclas que não são "simples", elas são decompostas em uma sequência de caracteres que no caso da seta para cima são respectivamente 27, 91 e 65. Para cada tecla, é uma outra sequência correspondente. No caso do F2, é a sequência 27, 79 e 81. Para a tecla "Page Up" é a sequência 27, 91, 53 e 126.

Quando você tenta aperta uma determinada tecla, uma sequência de comandos irá para o buffer e não importa quão veloz você seja, nunca irá apertar duas teclas ao mesmo tempo e isso é uma vantagem para nós, pois iremos utilizar justamente isso para diferenciar as teclas `'ESC'`(27), `'^'`(94) e `'A'`(65) da `UP_KEY`(27, 94, 65).

Para arrumar isso, vamos verificar se existe entrada após digitar uma tecla. Caso exista ele vai armazenar novamente em outra variável e verifica se existe novamente entrada e repete o processo até que acabe. Assim, se apertarmos a sequência `'ESC'+'^'+ 'A'` haverá uma pausa entre as teclas enquanto que digitar `UP_KEY` todas são enviadas de uma vez.

O código [7-get\\_keys.c](#) exemplifica esse processo. Caso seja pegos todo de uma vez, então imprime todos em uma linha. Teste o código e veja como funciona. Para sair, basta apertar enter ou mesmo Ctrl+C.

```

1 char c;
2 do{
3     do{
4         c = getch();
5         printf("%d;", c);
6     }while(kbhit()); /* Enquanto houver entrada, ler e imprimir */
7     printf("\n"); /* Imprimir a quebra de linha para diferenciar
8         comandos */
9 }while(c != '\n'); /* Pode haver confusao se o ultimo caracter da
    sequencia tambem tiver numero 65, mas muito raro */
10 return 0;

```

A partir daí, podemos fazer uma função que verifica isso e retorna um número correspondente à tecla, mas como é um conjunto de números(lembrando-se, 27 94 65), então é necessário criar nosso próprio padrão. Um padrão bom para isso é mostrado no código [funcoes\\_extras.c](#), em que se utiliza uma função que retorna inteiro e dependendo da entrada, consegue retornar uma tecla de até 8 chars.

```

1 unsigned long int key()
2 {
3     int j, i = 0;
4     unsigned long int retorno = 0;
5     char c[8];
6     do{
7         c[i] = getch();
8         i++;
9     }while(kbhit() && i < 8);
10    for(j = 0; j < i; j++)
11    {
12        retorno *= 256;
13        if(c[j] < 0)
14            retorno += 256;
15        retorno += c[j];
16    }
17    return retorno;
18 }

```

Assim, tem-se que para um número, sempre há somente uma combinação devido à base numérica utilizada. Tal função utiliza somente uma variável, contudo há possibilidade de retornar uma estrutura ou gravar os chars dentro de um vetor utilizando ponteiros.

Entendendo o funcionamento da função(*key*) pode-se criar diversas outras que sejam mais fáceis de utilizar. Algumas chaves estão armazenadas no arquivo [keys.txt](#) e diversas outras podem ser descobertas fazendo combinações. Um exemplo de combinação é utilizar *Alt + alguma\_tecla*, que no caso *Alt + q* = 27 113 enquanto *q* = 113.

## Outras funções

Outras funções estão no arquivo [funcoes\\_extras.c](#), documentadas e explicando o que cada uma faz além de serem fáceis de entender sabendo o que as funções *kbhit* e *getch* fazem.

## Contato

Para críticas, sugestões, dúvidas e/ou contribuições, envie e-mail para:  
*carlos.adir.leite@gmail.com*.