# Erros comuns ao utilizarmos a função ***scanf()***

A função ***scanf()***, cujo protótipo está declarado em ***stdio.h***, serve para efetuarmos a entrada (do teclado) formatada de dados nos nossos programas, daí o nome ***scan** formated*. É uma função muito poderosa, porém muito irritante e problemática para alguns iniciantes.

Podemos classificar os problemas básicos do uso de ***scanf()*** em 3 tipos:

- Deixar sujeira no *buffer* do teclado
- Ler *strings* com espaço
- Mau uso do ***fflush()***

Antes de explicar o que é o *buffer* do teclado também chamado de *buffer* de entrada, vamos dar olhada num código bem simples:

Código:
```
#include <stdio.h>

char caractere;

int main()
{
     printf("Escreva algum caractere\n");
     scanf()("%c", &caractere);

     printf("Mais um caractere\n");
     scanf()("%c", &caractere);
}
```

Antes de compilar e rodar o código acima, vamos tentar raciocinar de forma lógica como o programa acima deverá supostamente funcionar. Declaramos uma variável caractere do tipo *char*, que claramente usaremos para armazenar um (e só um) caractere.

O primeiro ***scanf()*** vai tentar ler o primeiro caractere que você digitar e vai guardar na variável. O segundo ***scanf()*** vai fazer exatamente o mesmo. Isso é só uma dedução lógica, agora copie o código e tente compilar, você irá de certa maneira ficar surpreso com o comportamento do programa.

O primeiro ***scanf()*** vai funcionar normalmente, já o segundo não vai funcionar e o programa vai terminar, será que fiquei louco ou o ***scanf()*** é que está louco?

Agora chegou o momento de explicar porque é que isso ocorre, e o que é o *buffer* de teclado (também chamado *buffer* de entrada ou *stdin*).

Quando você invoca funções de entrada como ***scanf()***, *getchar*, etc, tais funções pausam a execução e esperam que você digite qualquer caractere terminado por ENTER. Quando você digita esses dados, eles não são ainda enviados para a variável que irá guardar os mesmos. Os dados são enviados para um espaço de memória chamado *buffer* de teclado, isso explica o porquê de ser possível deletar os caracteres com a tecla BACKSPACE, antes de teclar ENTER.

Os dados só são realmente guardados na variável pretendida, quando você pressiona a tecla ENTER, pois essa tecla indica ao ***scanf()*** que terminamos o ato de entrada de dados. É importante saber que ENTER na memória é um caractere como qualquer outro, ele é o caractere LF (*Line Feed*) cujo código na tabela ASCII é igual à 10 (decimal).

http://pt.wikipedia.org/wiki/Tabela_ASCII

O que acontece basicamente no primeiro ***scanf()*** é:

- Você digita um caractere e pressiona ENTER;
- Como só precisamos armazenar um caractere na variável, o caractere é armazenado na variável e o ENTER permanece no *buffer* de entrada;

Já o segundo ***scanf()***:

- Como o *buffer* de entrada não está vazio (tem o ENTER lá), ele lê aquele ENTER, e como ENTER serve para terminar a entrada de dados pelo teclado, o ***scanf()*** prossegue.

Vamos supor que no primeiro ***scanf()*** você digite **'A'**(ASCII 97) e pressione ENTER, o ***scanf()*** vai armazenar os dois caracteres (97,10) no *buffer* de teclado:


Como o ENTER serve para sinalizar o fim da entrada de dados pelo teclado, o caractere **'A'**(ASCII 97) vai ser tirado do *buffer* e armazenado na variável. Quando você invocar o segundo ***scanf()***, ele vai ler o próximo caractere do *buffer* de entrada, que é o ENTER. Mais uma vez repito: o ENTER sinaliza o fim de entrada de dados, portanto o segundo ***scanf()*** vai prosseguir sem dar chance de digitar-se algo no console.

Modifique o programa para a versão que se segue e verifique seu comportamento e os valores (tabela ASCII) dos caracteres apresentados pelo programa:

Código:

```
#include <stdio.h>

int main(){
     char caracter;
     printf("Digite algum caracter\n");
     scanf("%c",&caracter);

     printf("O primeiro caracter foi [%c], seu numero na tabela ASCII e [%d]", caracter, caracter);
     printf("\nDigite o segundo caracter:\n");
     scanf("%c",&caracter);
     printf("O segundo caracter foi [%c], seu numero na tabela ASCII e %d", caracter, caracter);
}
```

Há uma solução simples para esse problema. Basta usar o *scanset* **" * "** (asterisco) no ***scanf()***. Tal operador diz para o ***scanf()*** ignorar qualquer coisa por exemplo:

Código:

```
scanf()("%c %*c", &caractere);
```

No código acima o ***scanf()*** irá ler um primeiro caractere e depois vai suprimir o seguinte (que normalmente é o ENTER), o que quer dizer que tal caractere não será armazenado no *buffer* de entrada. O operador de supressão " * " pode ser normalmente usado com outros especificadores de formato, por exemplo:

Código:

```
%*i // ignora um inteiro.
%*f // ignora um float.
```

É recomendável também usar o *scanset* ou operador de supressão " * " em leituras de *string* via ***scanf()***, exemplo:

Código:

```
char str[32];
scanf()("%s %*c", str);
```

O código acima irá ler uma *string* e armazenar em *str*, o ENTER será ignorado pelo ***scanf()***, ou seja, não será armazenado no *buffer* de entrada.

Correção para o código problemático:

Código:

```
#include <stdio.h>

int main(){
    char caracter;
    printf("Escreva algum caractere\n");
    scanf()("%c %*c",&caractere);

    printf("Mais um caractere\n");
    scanf()("%c %*c",&caractere);
}
```

O segundo problema associado ao uso do ***scanf()*** é a leitura de *strings* com espaço em branco, repare o código abaixo:

Código:

```
#include <stdio.h>

int main()
{
    char str[100];

    printf("Digite uma string com espacos\n");
    scanf()("%s %*c",str);

    printf("Você digitou: %s",str);
    getchar();
}
```

Compile e rode o programa, digite uma *string* com espaços em branco, por exemplo **"Ola Mundo"**.

Repare que o ***scanf()*** só irá ler a primeira palavra da *string* ("Ola") e irá ignorar o resto. Resolver este problema é relativamente fácil, basta usar o *scanset* **[^caractere]** que diz ao ***scanf()*** ler todos os elementos de uma *string* (até espaços) **delimitados** por um caractere especificado no *scanset*.

O código abaixo por exemplo, faz o ***scanf()*** ler toda a *string* e ignorar os caracteres após o caractere 'a'.

Código:

```
#include <stdio.h>

int main()
{
    char str[100];

    printf("Digite uma string que tenha o caractere 'a'\n");
    scanf()("%[^'a']s%*c",str);

    printf("Você digitou: %s",str);
    getchar();
}
```

Tente digitar por exemplo **"Eu idolatro C"**, o ***scanf()*** só vai ler **"Eu idol"**, pois o ***scanf()*** só leu parte da *string* que é delimitada pelo caractere 'a'.

Agora voltando ao cerne do problema inicial, nós podemos ler *strings* inteiras com espaços, se especificarmos ao ***scanf()*** o caractere " **\n** " (que representa o ENTER ou quebra de linha) como delimitador da *string* que pretendemos ler.

Código:

```
#include <stdio.h>

int main()
{
    char str[100];

    printf("Digite uma string\n");
    scanf()("%[^\n]s%*c",str);

    printf("Você digitou: %s",str);
    getchar();
}
```

Resolvido, recomendo ir ao final da página para dar uma olhada nos links referentes ao ***scanf()***.

O terceiro problema associado ao uso do ***scanf()***, é usar a função ***fflush()*** para limpar o *buffer* de entrada. Essa é uma alternativa ao especificador de supressão " ***** **"**, pois elimina o ENTER ou qualquer outro caractere do *buffer* de entrada.

Código:

```
#include <stdio.h>

char caractere;

int main()
{
    printf("Escreva algum caractere\n");
    scanf()("%c",&caractere);

     fflush(stdin);

    printf("Mais um caractere\n");
    scanf()("%c",&caractere);
}
```

Mas apesar do ***fflush()*** resolver o problema, não é recomendável o seu uso em *buffer*s de entrada (*stdin* é o *buffer* do teclado), pois segundo a documentação o seu efeito em *buffer*s de entrada é indefinido, hora pode funcionar hora pode não funcionar, ou pode acontecer algo mais grave.

Além dos três problemas mais comuns apresentados anteriormente, acredito que existe mais um que acontece com a maioria dos iniciantes na linguagem C em sua fase inicial: esquecer-se de incluir o operador " **&** " antes do nome da variável que receberá o conteúdo lido pela função ***scanf()***.

É preciso entender a importância desse operador para tentarmos minimizar o problema do "esquecimento" de sua inclusão. Como foi dito no início desse texto, ***scanf()*** pertence ao conjunto de funcionalidades presentes na biblioteca *stdio.h*, que engloba todas as rotinas de

entrada e saída como leitura do teclado, leitura e escrita em arquivos, saída de dados para monitores, saída de dados para impressoras, etc.

Uma vez que essa biblioteca não foi escrita pelo programador que a está utilizando incluida em seu programa, qualquer função pertencente a ela, como por exemplo o ***scanf()*** ou o ***printf()***, não faz parte do **escopo** deste programa, ou seja, qualquer função que seja utilizada pertencente a qualquer biblioteca incluída (#include) em seu programa, não pertencem a este, por esse motivo **NÃO TÊM ACESSO DIRETO** às variáveis criadas dentro do escopo deste mesmo programa. Digamos que essas funções, que são **externas**, não "enxergam" as variáveis **internas** ao seu programa, e não as "conhecem" pelo nome dado a elas por você, pois estas funções estão fora do escopo (o arquivo ou aquele par de chaves " { } ") do seu programa.

Então como fazer com que o ***scanf()*** encontre a variável correta para armazenar o dado que você quer coletar do teclado?

Simples. Usamos o endereço de memória em que a variável está alocada, ou em outras palavras, a posição ou lugar que ela ocupa **naquele** momento na memória. Fazemos isso com o operador " **&** ", quando utilizamos o comando:

```
scanf(" %d", &x);
```

A linha de código acima diz o seguinte ao compilador: "Faça uma leitura (*scan*) formatada (*f*) do teclado, leia um número inteiro (*%d*), e guarde o valor lido no endereço (*&*) da variável (*x*).

Isso é realizado dessa forma porque a função ***scanf()*** tem que ser genérica, e funcionar para qualquer tipo de dado lido, a qualquer momento que seja invocada. Digo a qualquer momento, pois por exemplo, em duas execuções seguidas de seu programa (executou, finalizou e executou novamente), não há garantias de que o programa sempre será executado no mesmo trecho de memória disponível, portanto muito provavelmente ocupará posições de memórias diferentes em execuções diferentes, quem define isso é o Sistema Operacional, e nós programadores não temos nenhum poder de decisão sobre isso. Mas as posições de memória (seus endereços), são **físicos** dentro do computador, e não mudam. Sempre serão os mesmos e estarão sempre no mesmo lugar. Portanto, suas variáveis podem ocupar qualquer posição, mas passando a posição em que ela está naquele momento (como em *&x*), a função ***scanf()*** será capaz de encontrá-la corretamente através de seu endereço atual.

Documentação do ***scanf()*** no cplusplus.com

http://cplusplus.com/reference/cstdio/scanf/

Ler espaços de uma *string* com ***scanf()***

http://gpraveenkumar.wordpress.com/2009/06/10/how-to-use-scanf-to-read-string-with-space/