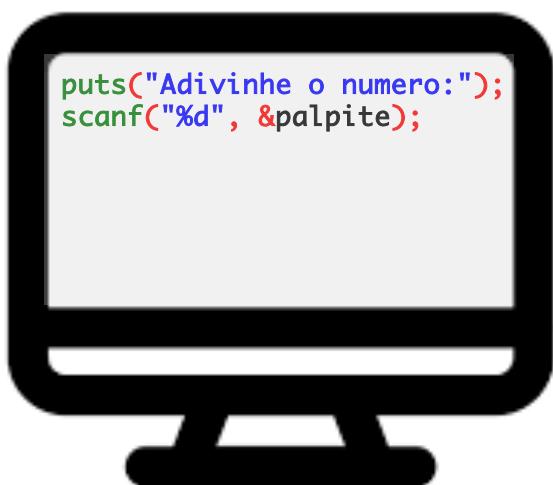
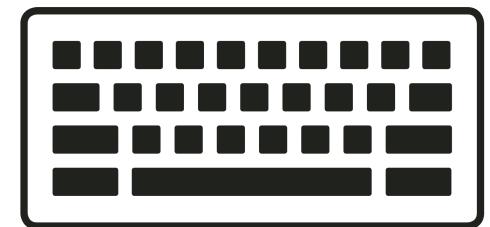


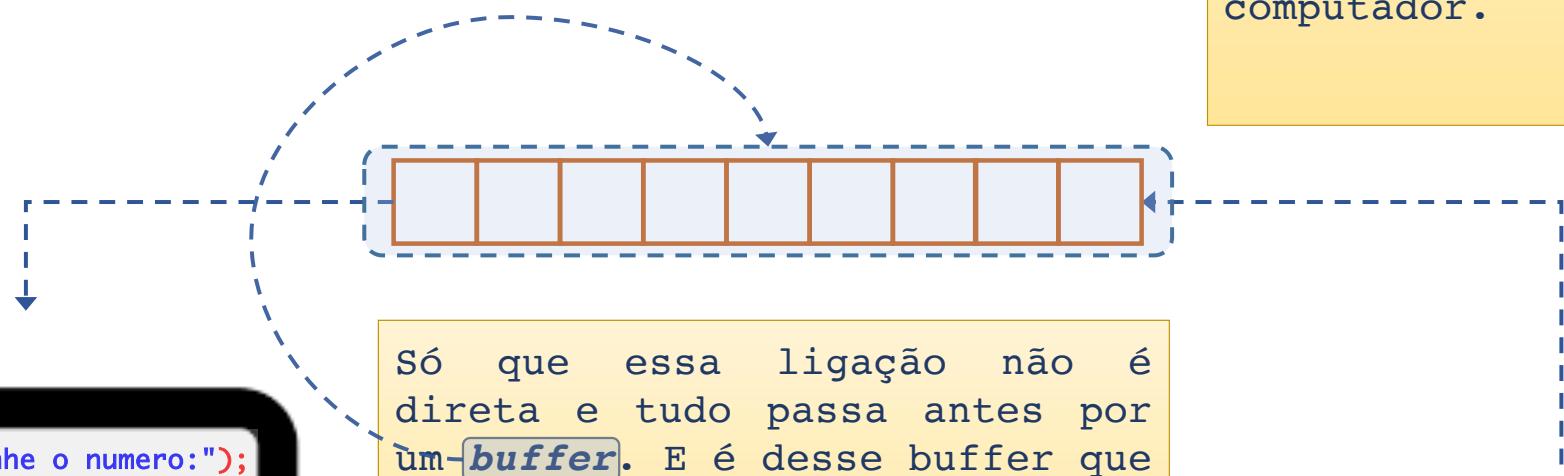
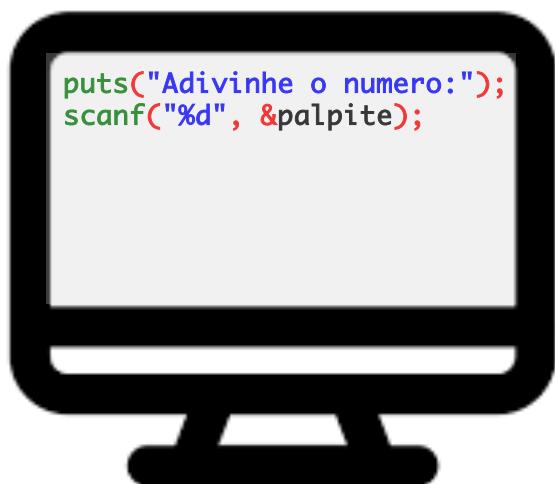
O buffer de entrada e o scanf

Já sabemos que o ***scanf*** é a função utilizada para capturar a entrada do teclado e transformar em dados úteis para o programa na memória do computador.



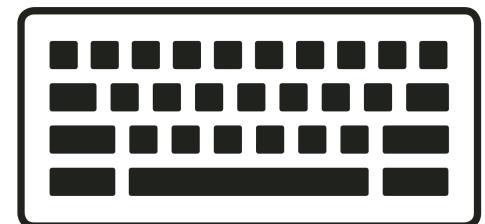
```
puts("Adivinhe o numero:");
scanf("%d", &palpite);
```





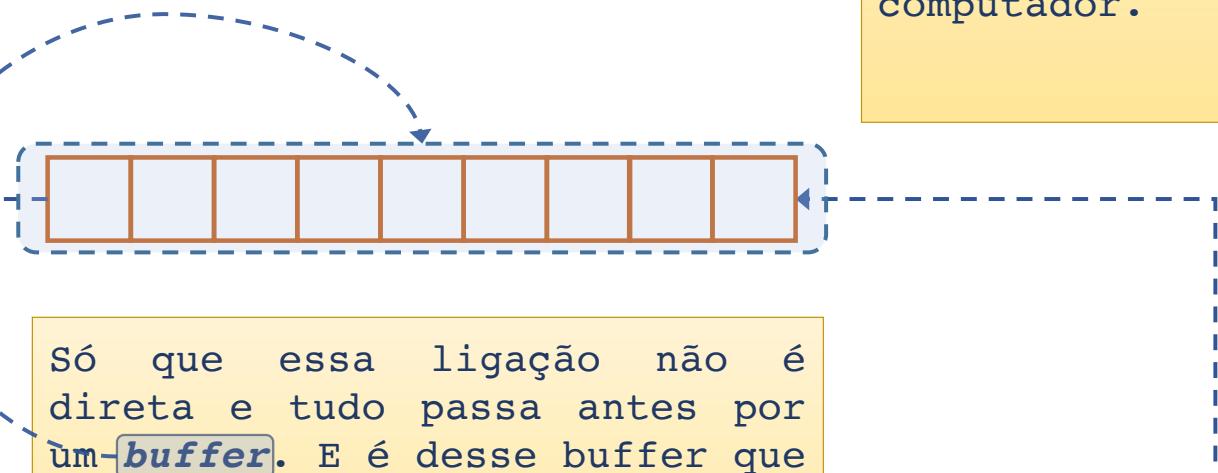
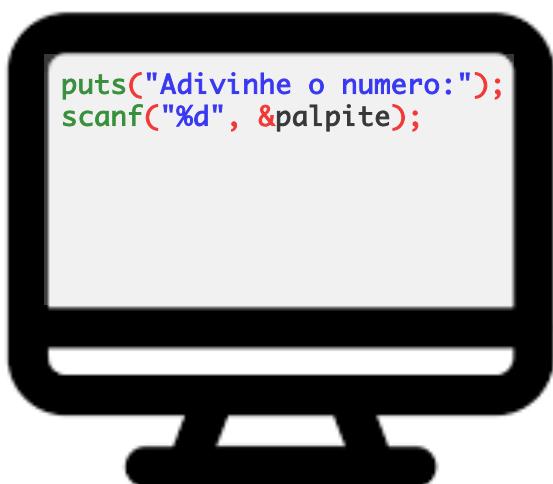
Só que essa ligação não é direta e tudo passa antes por um **buffer**. E é desse buffer que o **scanf** (como qualquer programa no computador) faz a leitura.

Já sabemos que o **scanf** é a função utilizada para capturar a entrada do teclado e transformar em dados úteis para o programa na memória do computador.

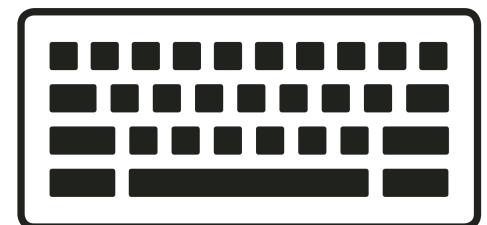


Esse **buffer** é uma região da memória que armazena tudo que digitamos no teclado e vai passando pro programa quando solicitado por ele – por exemplo, quando existe um **scanf**.

Já sabemos que o **scanf** é a função utilizada para capturar a entrada do teclado e transformar em dados úteis para o programa na memória do computador.



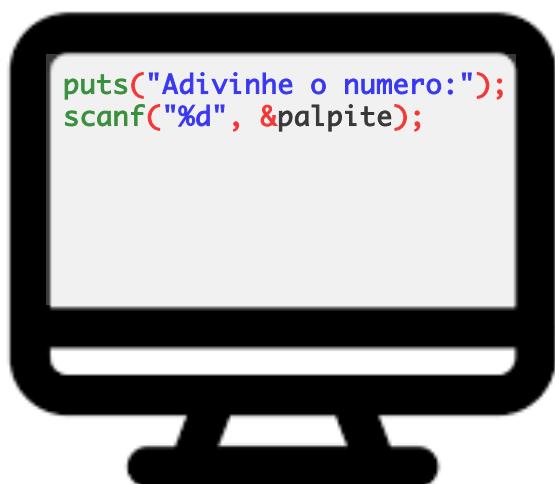
Só que essa ligação não é direta e tudo passa antes por um-**buffer**. E é desse buffer que o **scanf** (como qualquer programa no computador) faz a leitura.



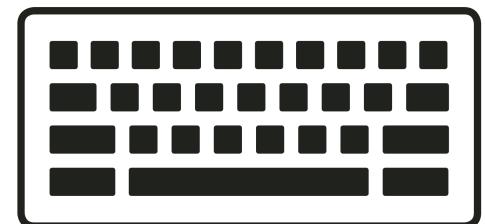
Esse **buffer** é uma região da memória que armazena tudo que digitamos no teclado e vai passando pro programa quando solicitado por ele – por exemplo, quando existe um **scanf**.

Já aconteceu de seu computador ficar meio lento e você digitar algumas coisas que não aparecem imediatamente mas tudo de uma vez quando o computador volta a funcionar bem? Pois é, essas coisas estavam no **buffer** esperando pelo programa.

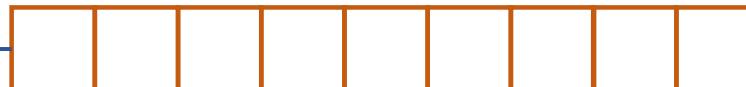
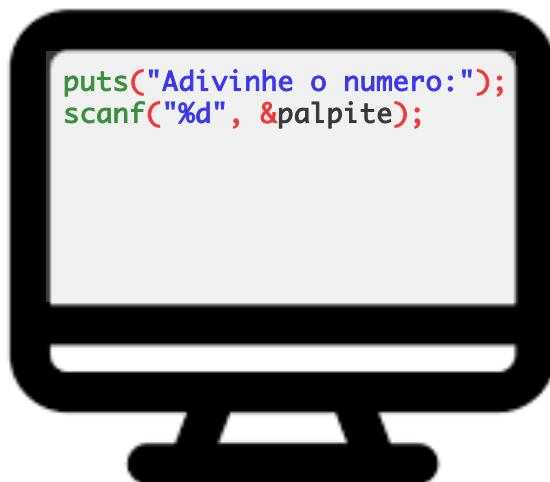
Já sabemos que o **scanf** é a função utilizada para capturar a entrada do teclado e transformar em dados úteis para o programa na memória do computador.



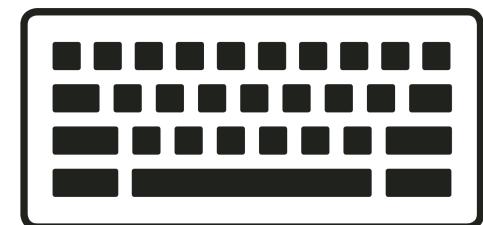
Só que essa ligação não é direta e tudo passa antes por um-**buffer**. E é desse buffer que o **scanf** (como qualquer programa no computador) faz a leitura.



Esse programa faz a leitura de um número inteiro.



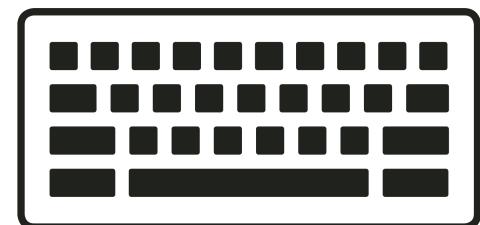
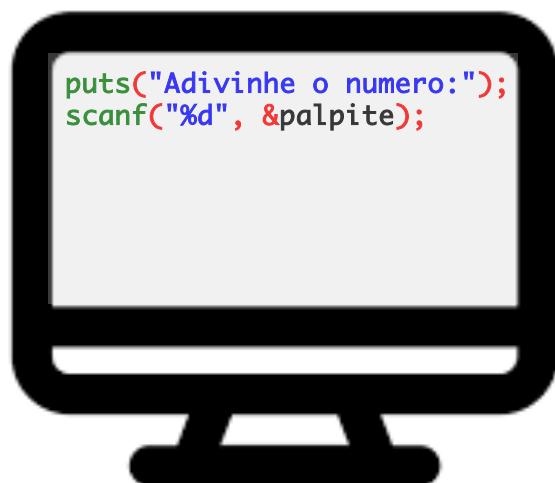
Já sabemos que o **scanf** é a função utilizada para capturar a entrada do teclado e transformar em dados úteis para o programa na memória do computador.



Esse programa faz a leitura de um número inteiro.

Quando a pessoa digitar **14** e apertar **enter**, o **buffer** vai ficar assim.

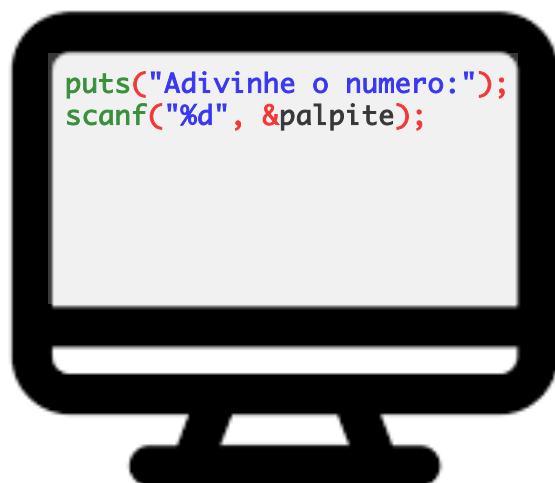
Já sabemos que o **scanf** é a função utilizada para capturar a entrada do teclado e transformar em dados úteis para o programa na memória do computador.



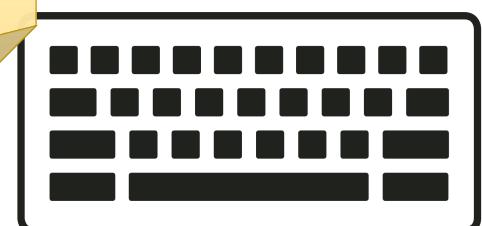
Esse programa faz a leitura de um número inteiro.

Quando a pessoa digitar **14** e apertar **enter**, o **buffer** vai ficar assim.

Já sabemos que o **scanf** é a função utilizada para capturar a entrada do teclado e transformar em dados úteis para o programa na memória do computador.



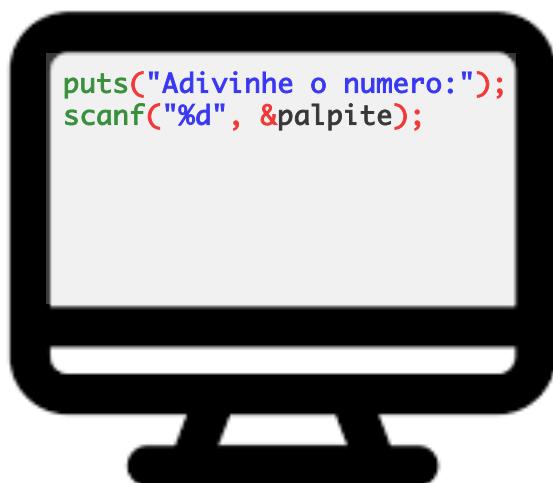
Então, a partir da primeira posição do **buffer** o **scanf** se pergunta: é possível formar um número com esse caractere?



Esse programa faz a leitura de um número inteiro.

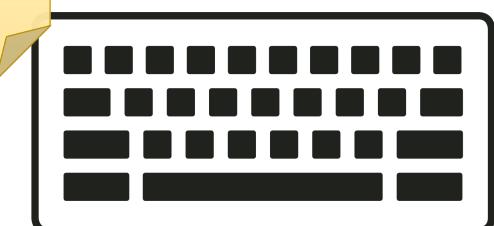
Quando a pessoa digitar **14** e apertar **enter**, o **buffer** vai ficar assim.

Já sabemos que o **scanf** é a função utilizada para capturar a entrada do teclado e transformar em dados úteis para o programa na memória do computador.



Então, a partir da primeira posição do **buffer** o **scanf** se pergunta: é possível formar um número com esse caractere?

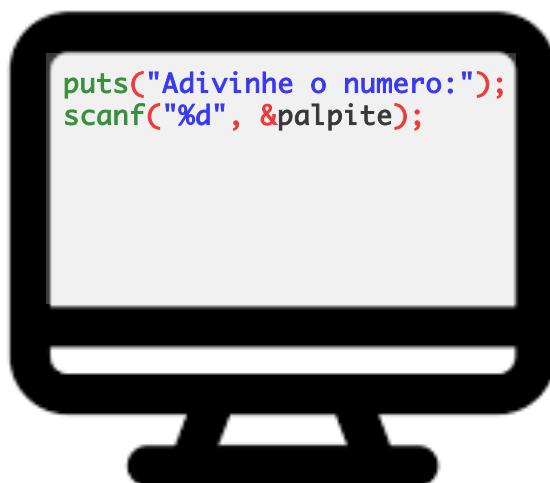
A resposta é **sim** e ele faz a mesma pergunta na próxima posição...



Esse programa faz a leitura de um número inteiro.

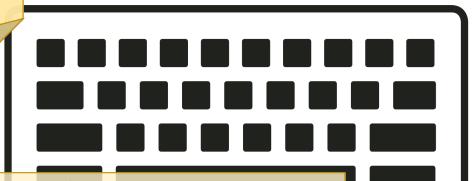
Quando a pessoa digitar 14 e apertar **enter**, o **buffer** vai ficar assim.

Já sabemos que o **scanf** é a função utilizada para capturar a entrada do teclado e transformar em dados úteis para o programa na memória do computador.



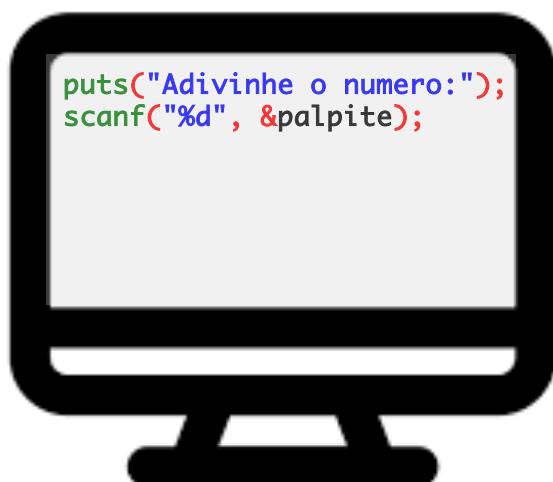
Então, a partir da primeira posição do **buffer** o **scanf** se pergunta: é possível formar um número com esse caractere?

A resposta é **sim** e ele faz a mesma pergunta na próxima posição...



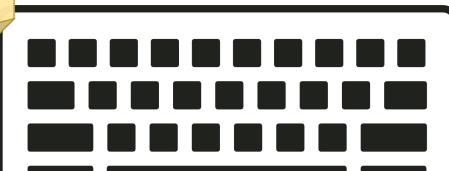
Mais uma vez é possível e o **scanf** chega no **\n**, que não é mais um dígito e não pode fazer parte do inteiro.

Nessa situação o **scanf** entende que o número já foi lido, então ele armazenará o número inteiro **14** na variável e excluirá os caracteres lidos do **buffer**...



Então, a partir da primeira posição do **buffer** o **scanf** se pergunta: é possível formar um número com esse caractere?

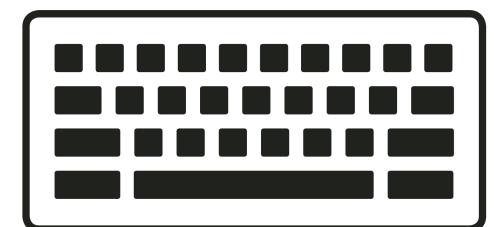
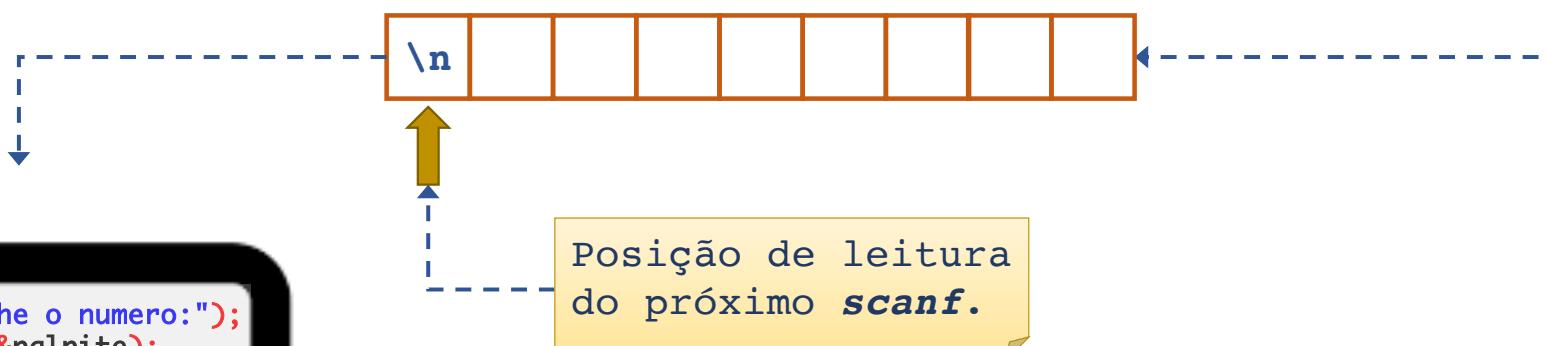
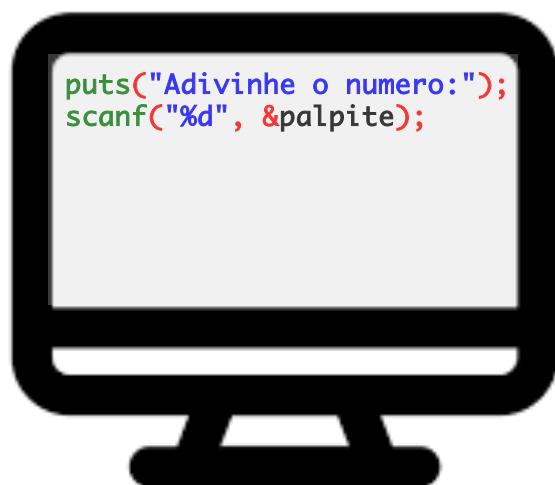
A resposta é **sim** e ele faz a mesma pergunta na próxima posição...



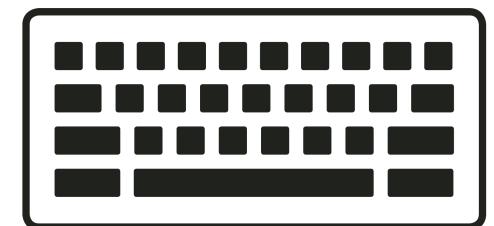
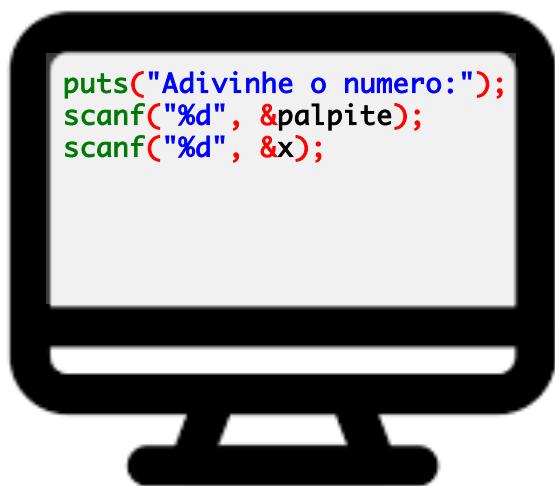
Mais uma vez é possível e o **scanf** chega no **\n**, que não é mais um dígito e não pode fazer parte do inteiro.

Nessa situação o **scanf** entende que o número já foi lido, então ele armazenará o número inteiro **14** na variável e excluirá os caracteres lidos do **buffer**...

O **buffer** agora ficará assim.

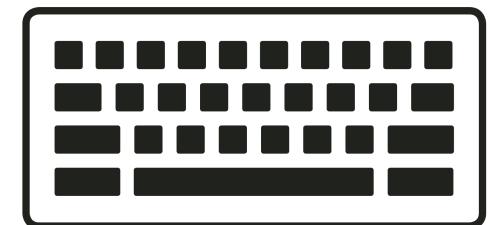
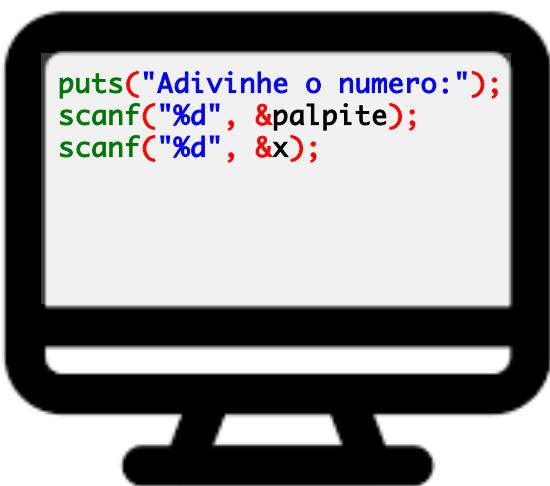
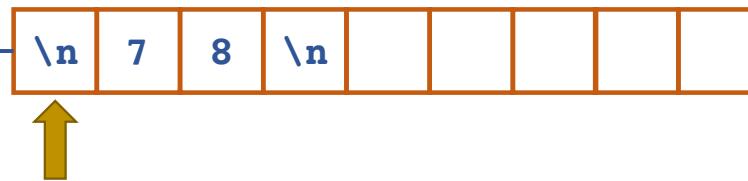


Suponha a leitura de um novo inteiro.



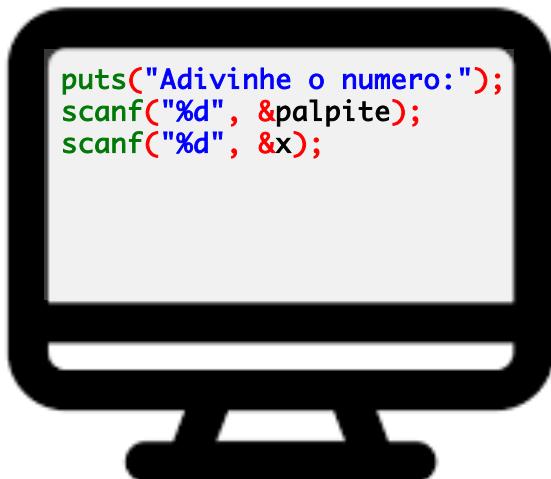
A pessoa digita **78** e
o **buffer** fica assim.

Suponha a
leitura de um
novo inteiro.

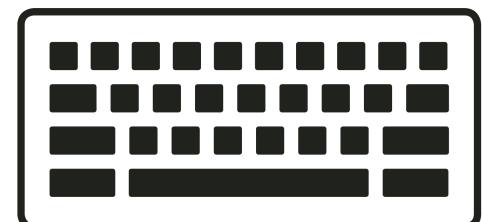


Suponha a leitura de um novo inteiro.

A pessoa digita **78** e o **buffer** fica assim.

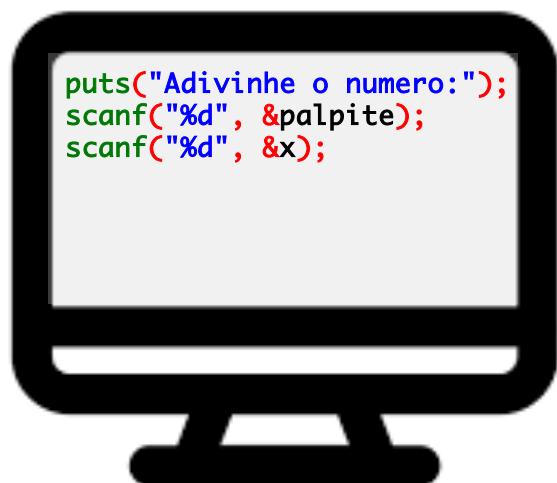


O **\n** não pode integrar um número inteiro, mas, apenas quando a leitura é de um tipo numérico (int, float, double...), os caracteres brancos do início do buffer são ignorados.



Suponha a leitura de um novo inteiro.

A pessoa digita **78** e o **buffer** fica assim.

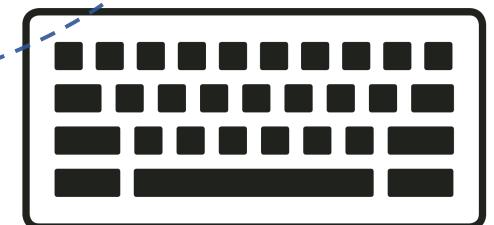


\n 7 8 \n



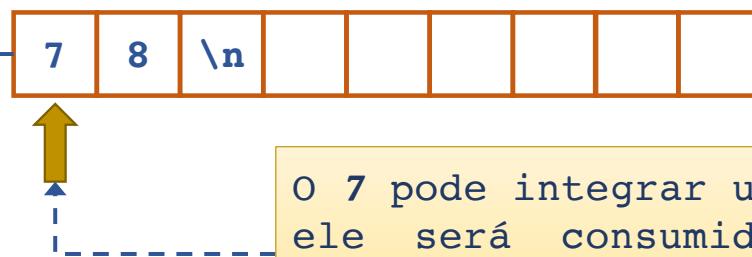
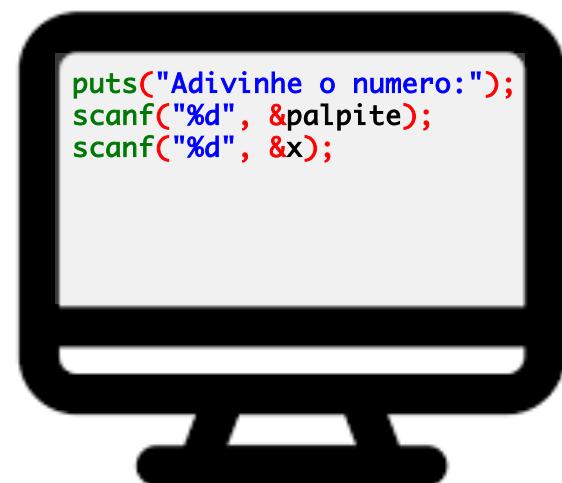
Os caracteres brancos são aqueles sem representação gráfica, como o \n, \t, espaço e outros. Eles ficam no início da tabela **ASCII**.

O \n não pode integrar um número inteiro, mas, apenas quando a leitura é de um tipo numérico (int, float, double...), os **caracteres brancos** do índice do buffer são ignorados.

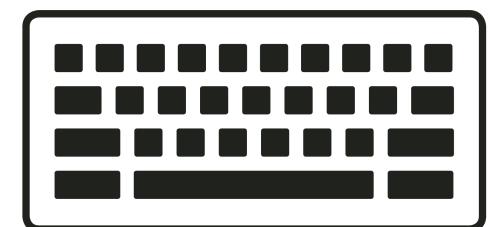


Suponha a leitura de um novo inteiro.

Depois de ignorar os brancos do início do **buffer**, ele fica assim.

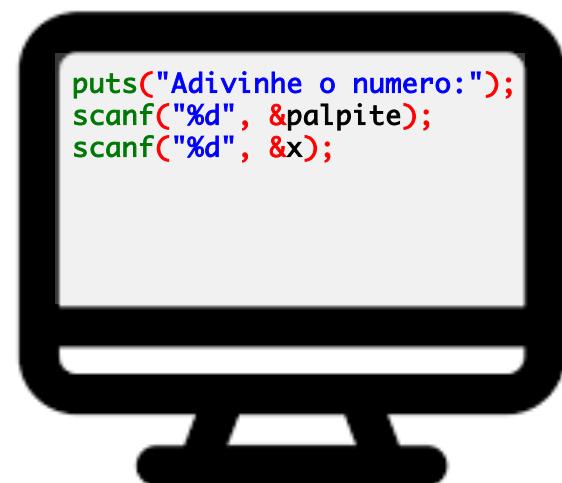


O 7 pode integrar um inteiro, então ele será consumido e a leitura passará pra próxima posição.



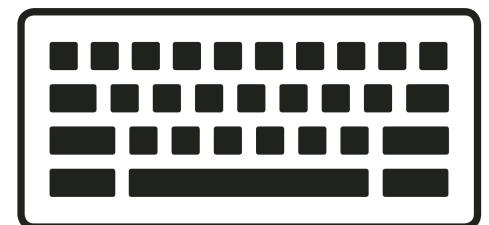
Suponha a leitura de um novo inteiro.

Depois de ignorar os brancos do início do **buffer**, ele fica assim.



O 7 pode integrar um inteiro, então ele será consumido e a leitura passará pra próxima posição.

Ocorrerá o mesmo com o 8.



Suponha a leitura de um novo inteiro.



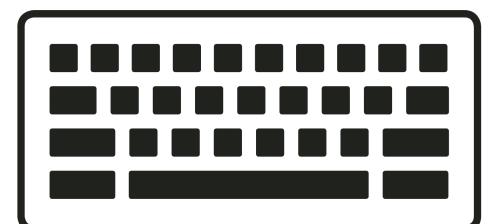
Depois de ignorar os brancos do início do **buffer**, ele fica assim.



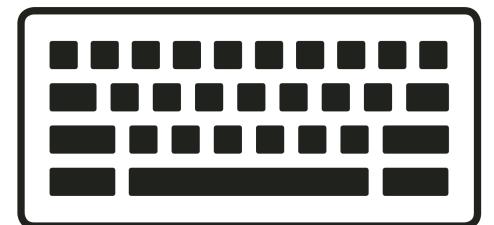
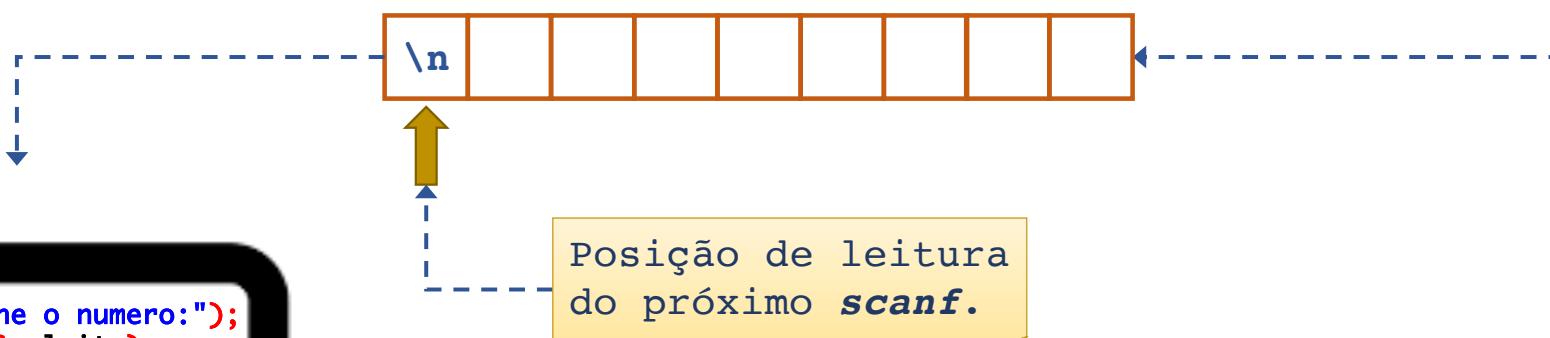
O 7 pode integrar um inteiro, então ele será consumido e a leitura passará pra próxima posição.

Ocorrerá o mesmo com o 8.

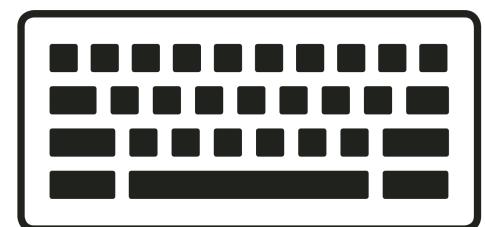
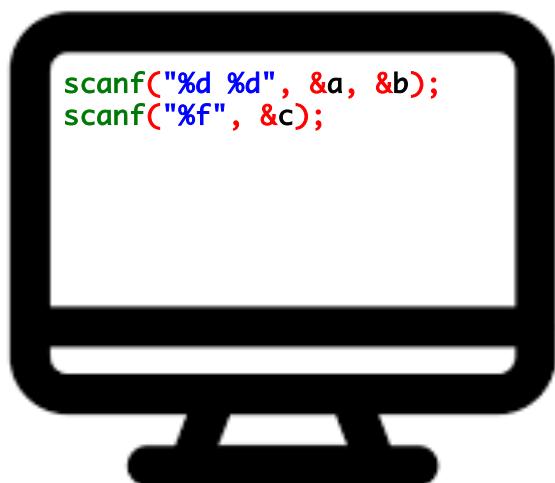
O \n não é um dígito, portanto é encerrada a leitura e o inteiro 78 é armazenado na variável.



E o **buffer** volta a ficar assim.



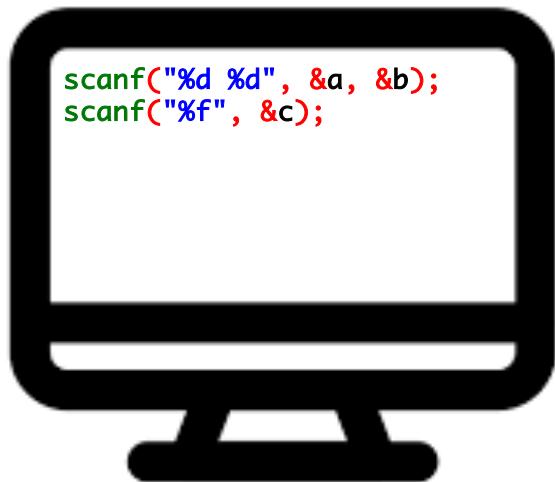
Suponha agora
essa sequência
de **scans**.



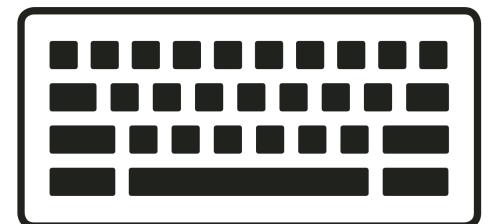
E que a pessoa tenha preenchido o **buffer** dessa forma.

Suponha agora essa sequência de **scanf**s.

```
scanf("%d %d", &a, &b);
scanf("%f", &c);
```



Posição de leitura do próximo **scanf**.



E que a pessoa tenha preenchido o **buffer** dessa forma.

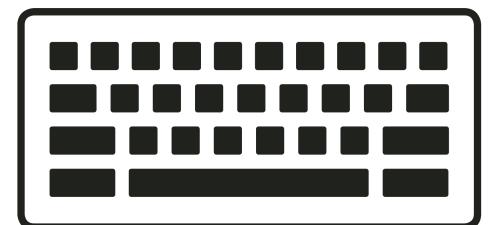
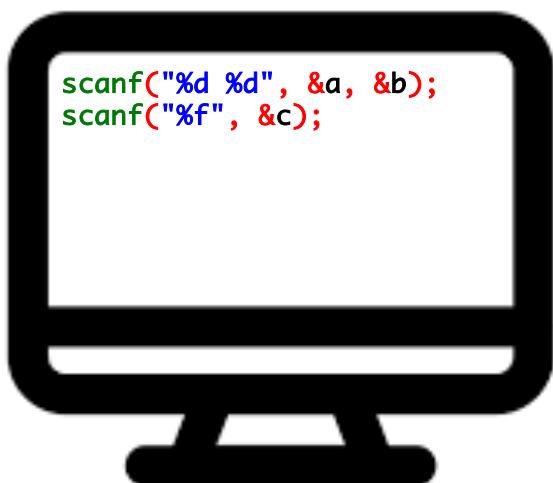
Suponha agora essa sequência de **scanf**s.

Seguindo o mesmo algoritmo, as variáveis **a**, **b** e **c** receberão os valores **2**, **3** e **1.56**, respectivamente.

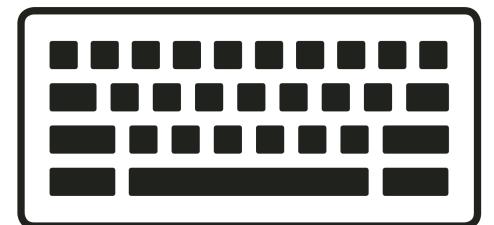
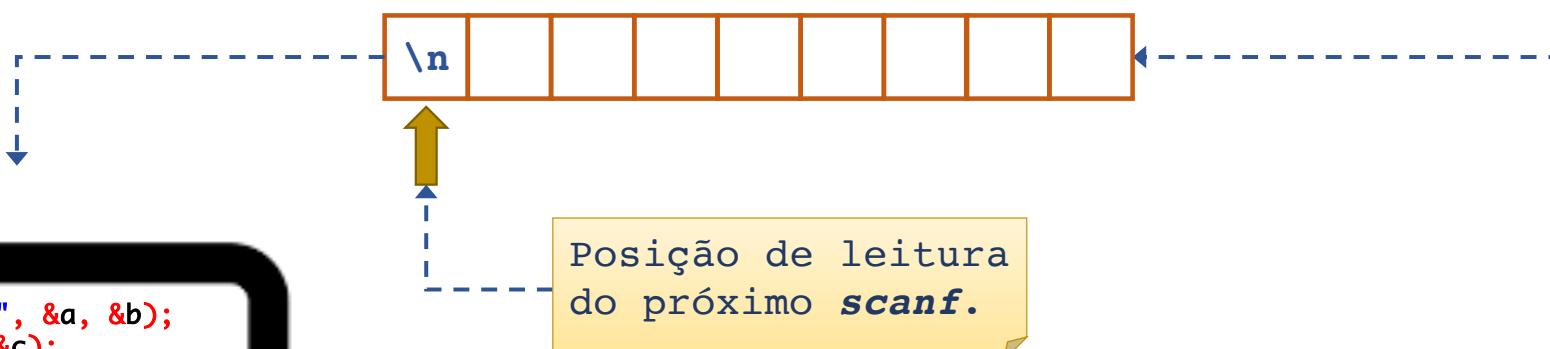
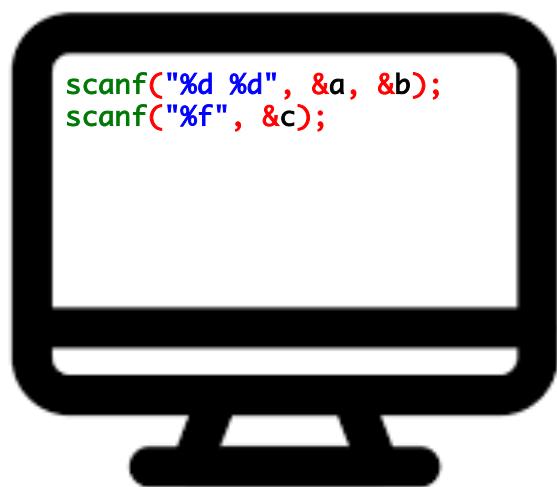


Posição de leitura
do próximo **scanf**.

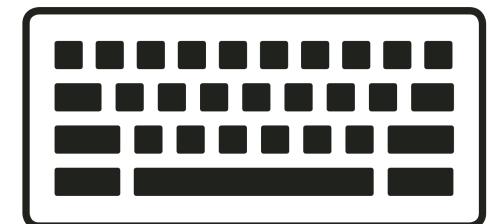
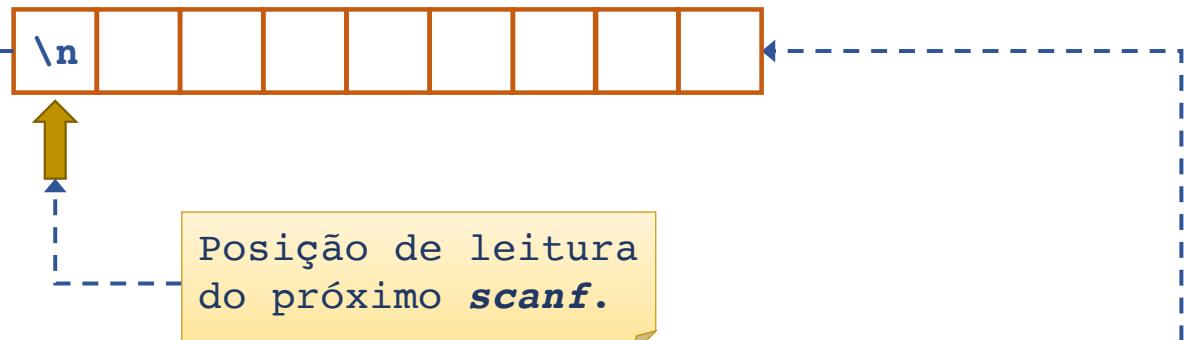
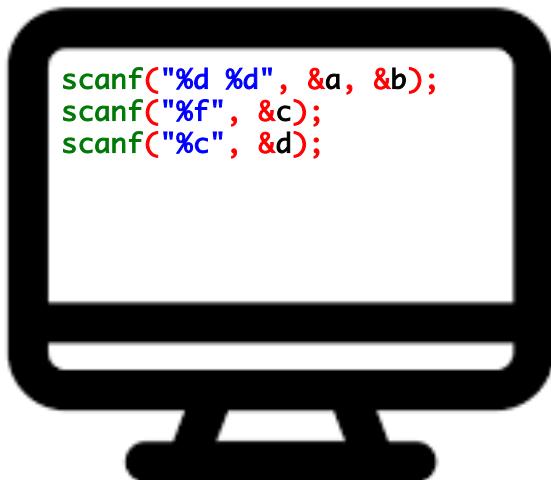
```
scanf("%d %d", &a, &b);
scanf("%f", &c);
```



E o **buffer** volta a ficar assim.

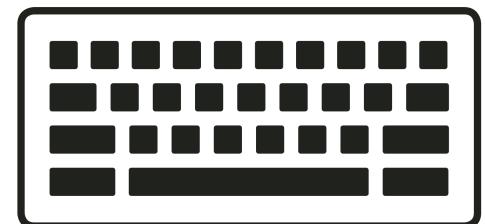
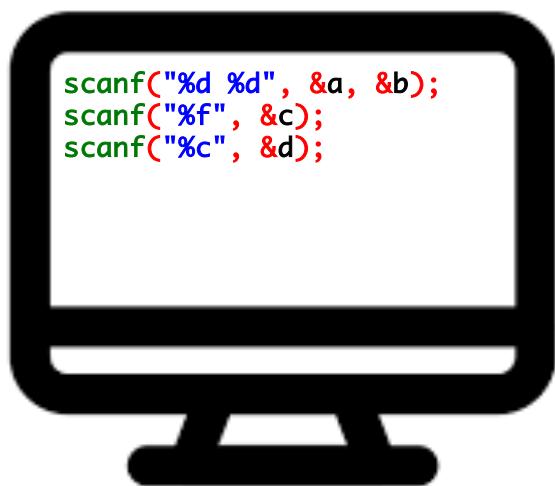
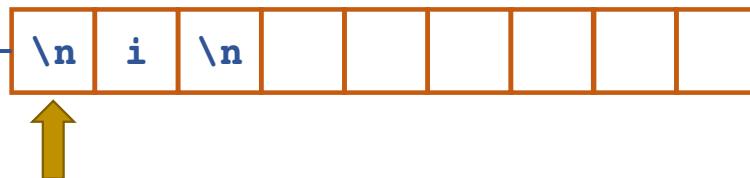


Um problema ocorre quando o próximo **scanf** for de um **%c**.



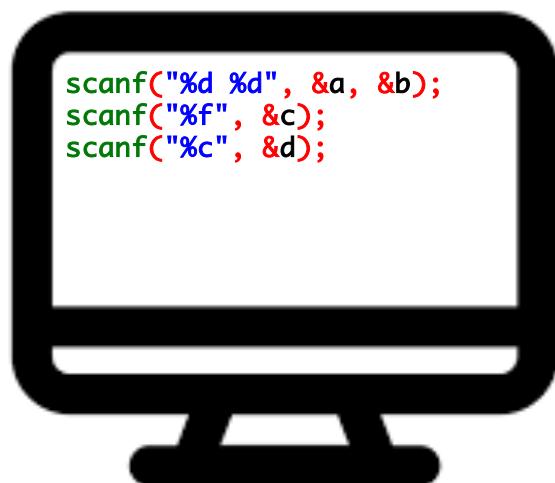
Um problema ocorre quando o próximo **scanf** for de um **%c**.

Se a pessoa digitar o caractere 'i' e apertar o **enter**, o **buffer** ficará assim.



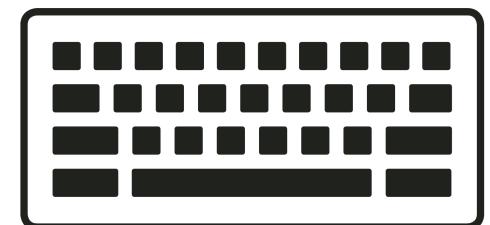
Um problema ocorre quando o próximo **scanf** for de um **%c**.

Se a pessoa digitar o caractere 'i' e apertar o **enter**, o **buffer** ficará assim.



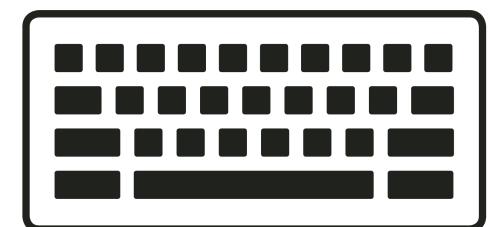
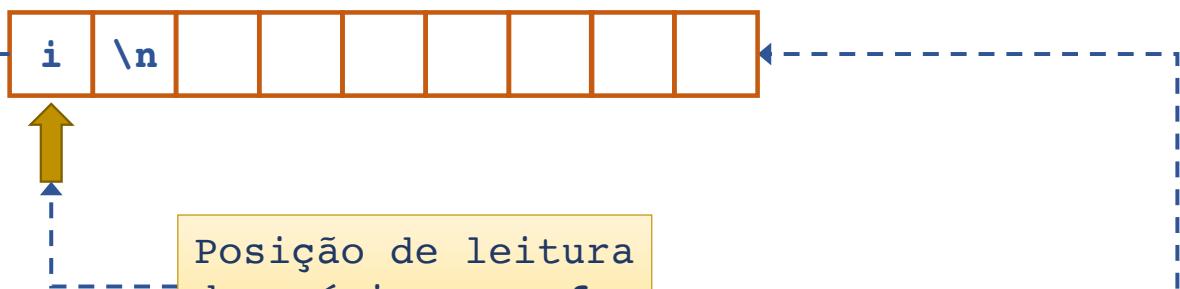
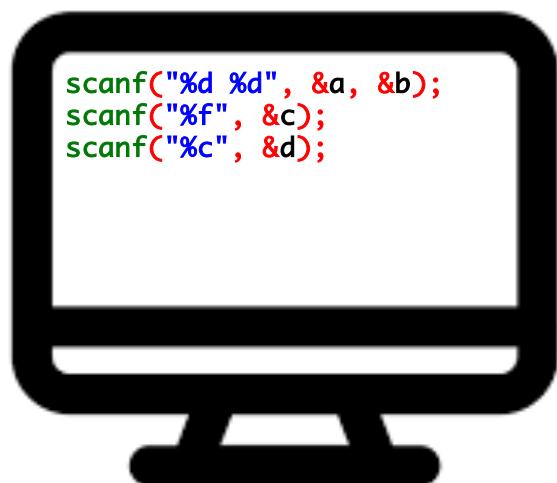
Ao verificar o caractere do buffer, o **%c** não pula os brancos, pois tudo digitado é um caractere, inclusive o **\n**.

E esse será o valor da variável **d**.

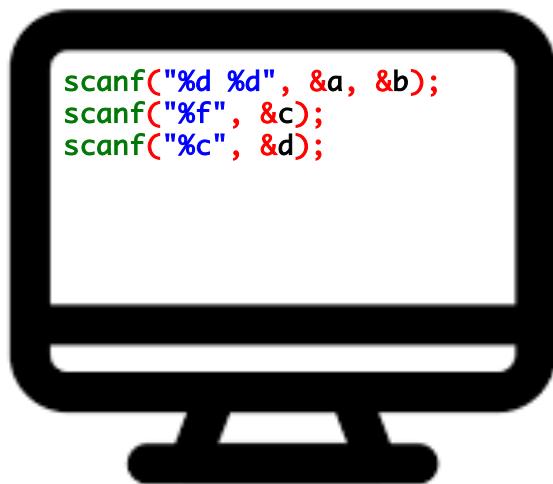


Um problema ocorre quando o próximo **scanf** for de um **%c**.

Depois de armazenar **\n** na variável **d** o **buffer** fica assim.



Um problema ocorre quando o próximo **scanf** for de um **%c**.



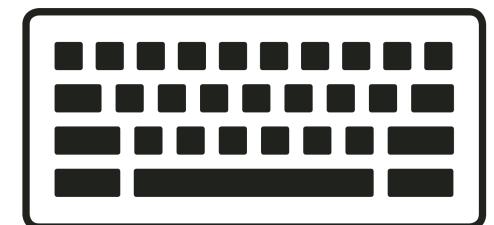
```
scanf("%d %d", &a, &b);
scanf("%f", &c);
scanf("%c", &d);
```

Depois de armazenar **\n** na variável **d** o **buffer** fica assim.

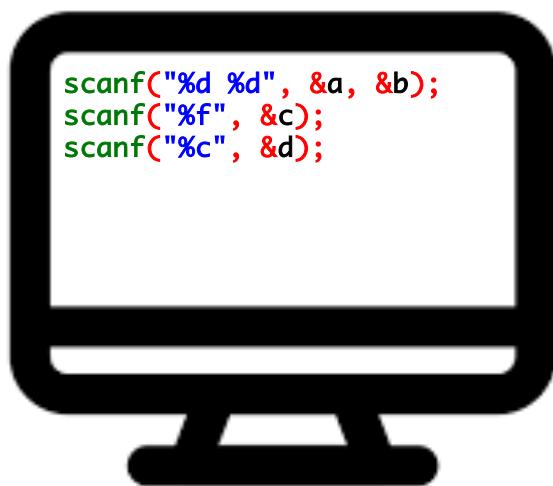


Posição de leitura do próximo **scanf**.

Ou seja, se o próximo **scanf** não for de outro **%c** – que receberia o valor '**i**' – temos um problema, pois nenhum tipo numérico aceitará o '**i**' como parte de um número.



Um problema ocorre quando o próximo **scanf** for de um **%c**.



```
scanf("%d %d", &a, &b);
scanf("%f", &c);
scanf("%c", &d);
```

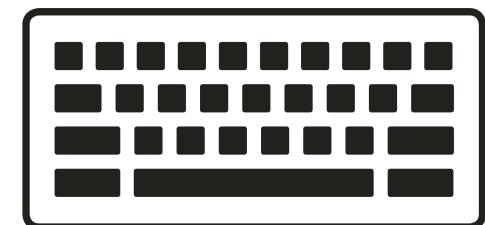
Depois de armazenar **\n** na variável **d** o **buffer** fica assim.



Posição de leitura do próximo **scanf**.

Então precisamos de uma maneira de **limpar o buffer** antes de fazer a leitura de um **%c**, para evitar que a leitura seja do '**\n**' remanescente.

Ou seja, se o próximo **scanf** não for de outro **%c** – que receberia o valor '**i**' – temos um problema, pois nenhum tipo numérico aceitará o '**i**' como parte de um número.



Um problema ocorre quando o próximo **scanf** for de um **%c**.

Se a pessoa digitar o caractere 'i' e apertar o **enter**, o **buffer** ficará assim.

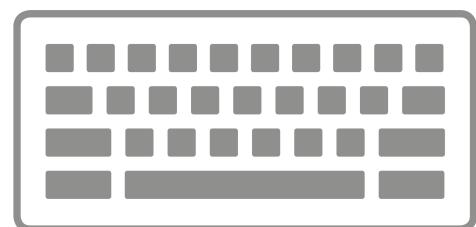
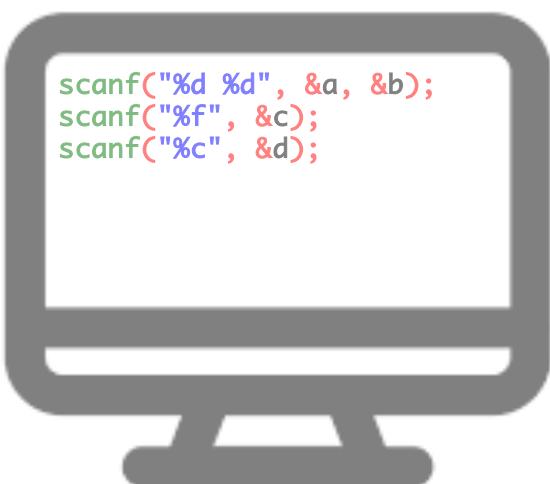
Então precisamos de uma maneira de **limpar o buffer** antes de fazer a leitura de um **%c**, para evitar que a leitura seja do '**\n**' remanescente.

Temos que evitar a situação ilustrada aqui.



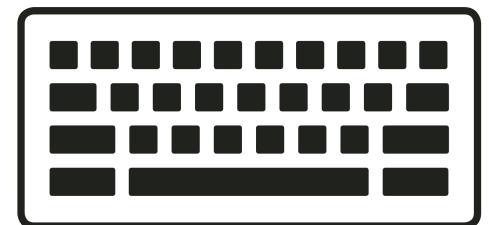
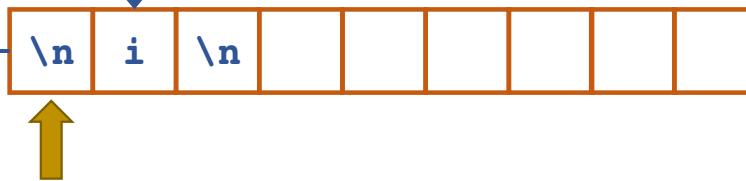
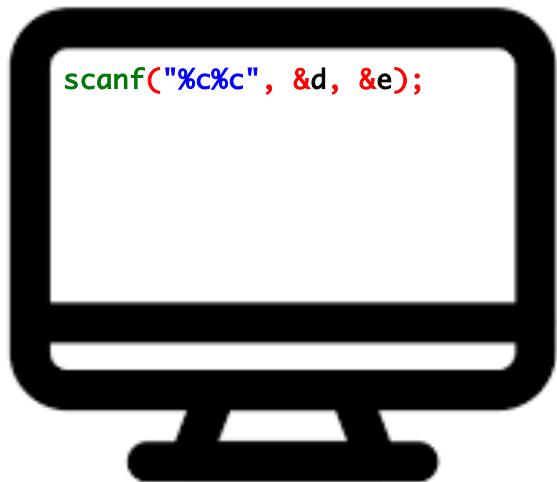
Ao verificar o caractere do buffer, o **%c** não pula os brancos, pois tudo digitado é um caractere, inclusive o **\n**.

E esse será o valor da variável **d**.



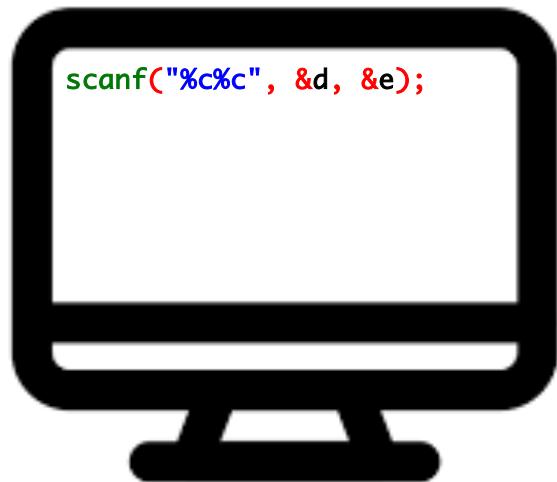
Caso o **scanf**
fosse esse.

E o buffer
estivesse
assim.

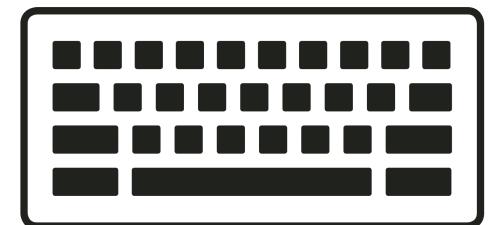


Caso o **scanf**
fosse esse.

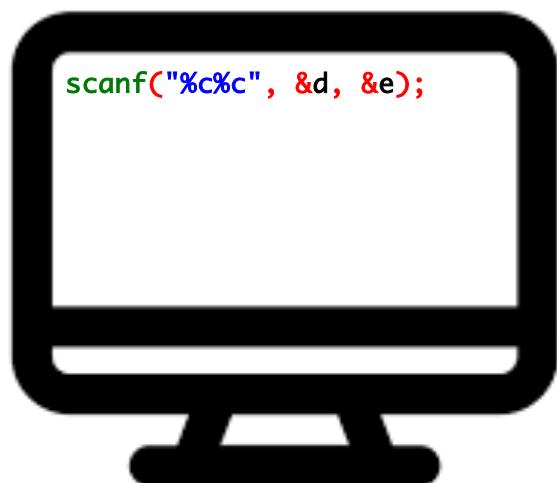
E o buffer
estivesse
assim.



A partir da posição do **buffer**,
a variável **d** receberia o valor
'\n' e a variável **e** receberia
o valor 'i', que é o desejado.

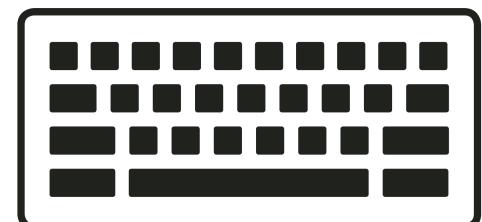


Caso o **scanf**
fosse esse.



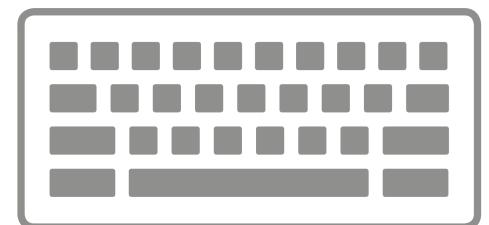
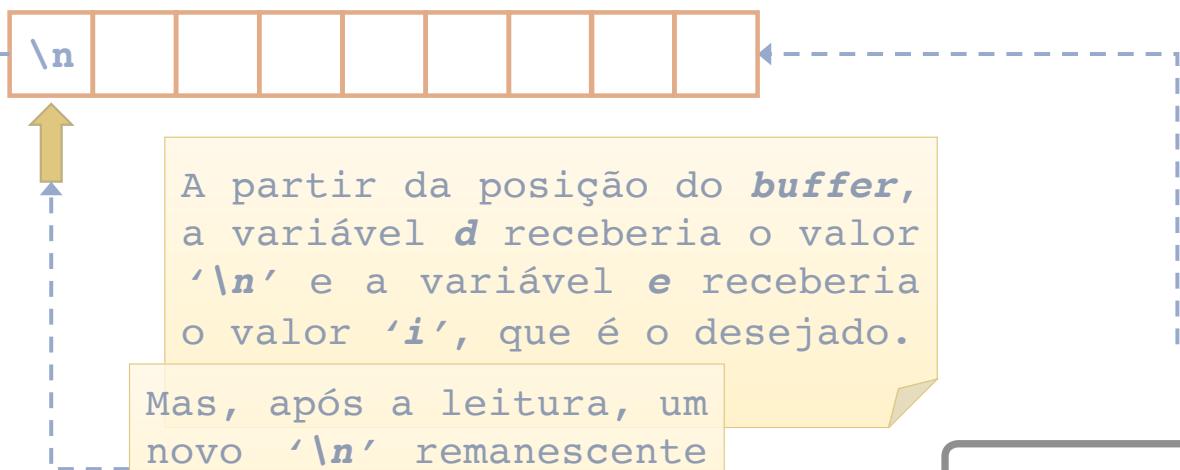
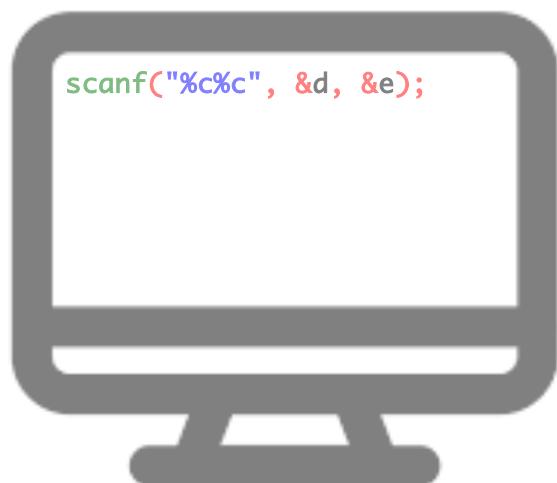
A partir da posição do **buffer**,
a variável **d** receberia o valor
'\n' e a variável **e** receberia
o valor 'i', que é o desejado.

Mas, após a leitura, um
novo '\n' remanescente
permaneceria.



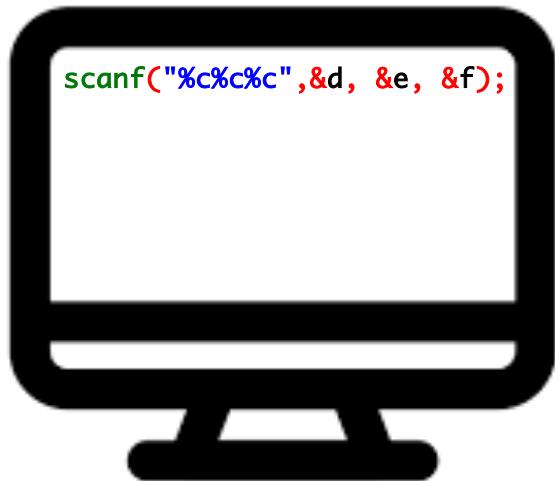
Vamos voltar e colocar mais um `%c`...

Caso o `scanf`
fosse esse.

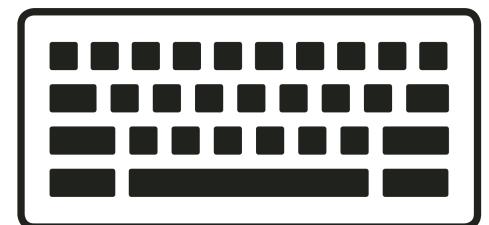
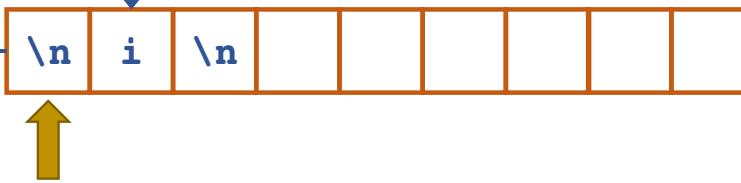


Caso o **scanf**
fosse esse.

E o buffer
estivesse
assim.



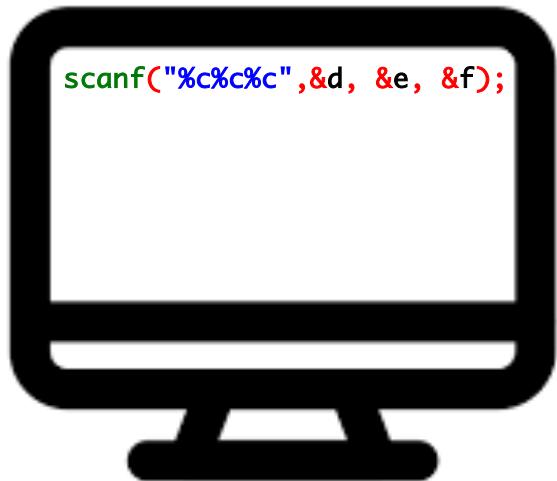
```
scanf("%c%c%c", &d, &e, &f);
```



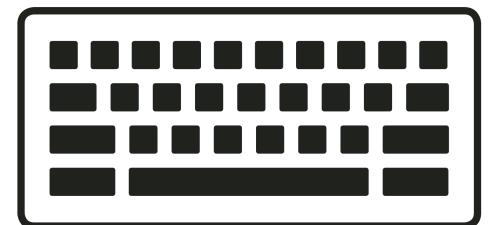
Caso o **scanf**
fosse esse.

E o buffer
estivesse
assim.

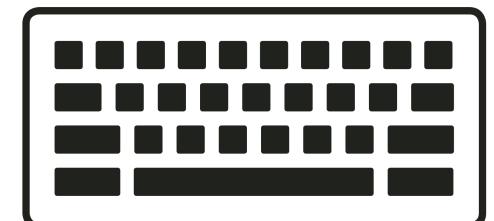
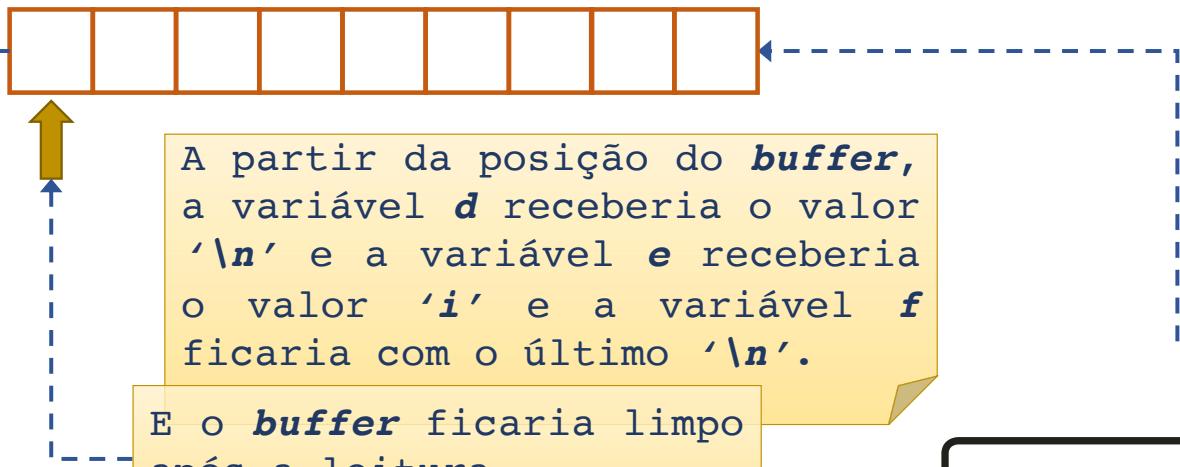
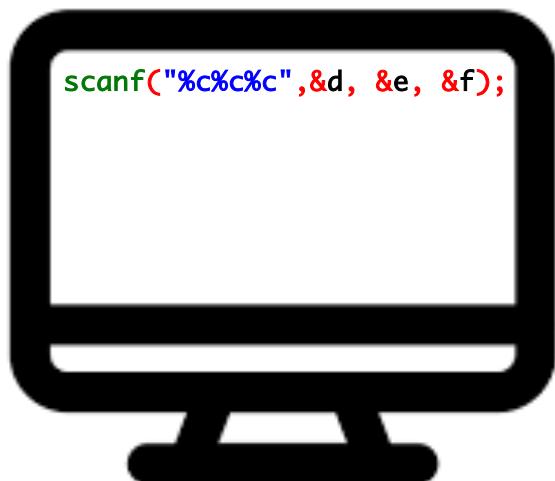
```
scanf("%c%c%c", &d, &e, &f);
```



A partir da posição do **buffer**,
a variável **d** receberia o valor
'\n' e a variável **e** receberia
o valor 'i' e a variável **f**
ficaria com o último '\n'.

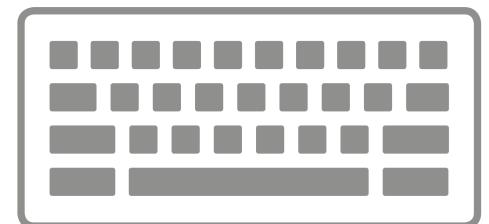
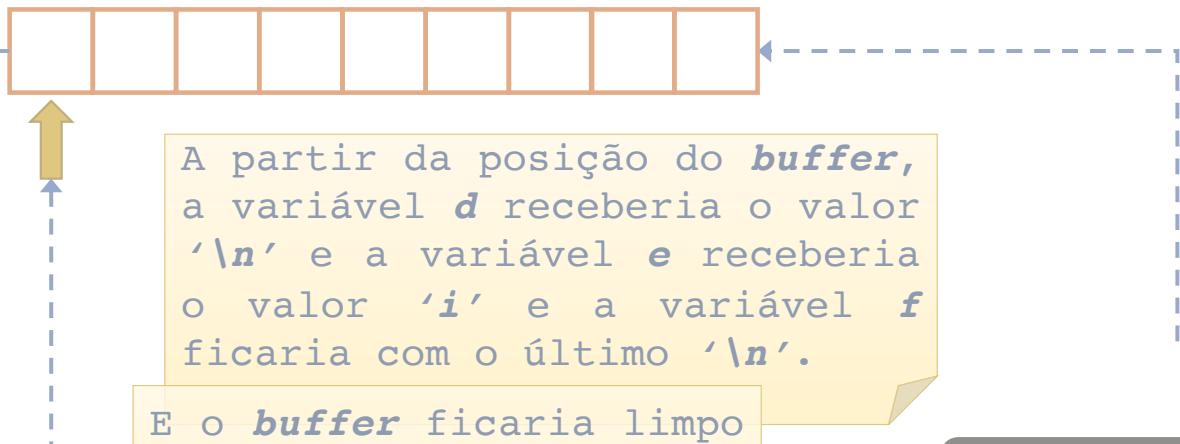


Caso o **scanf**
fosse esse.



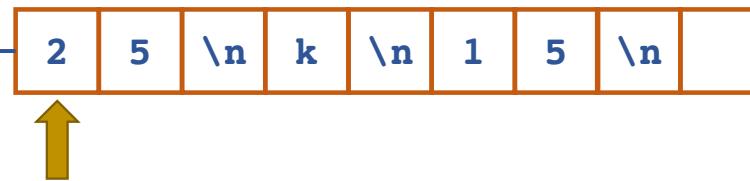
Então, parece que se colocarmos um `%c` no final de todos os `scans`, não deixaremos um '`\n`' remanescente...

Caso o `scanf` fosse esse.

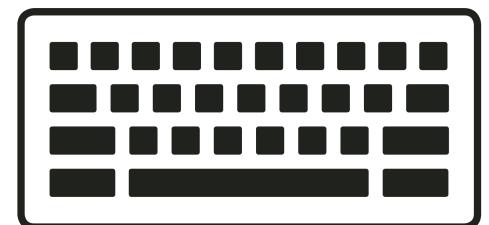
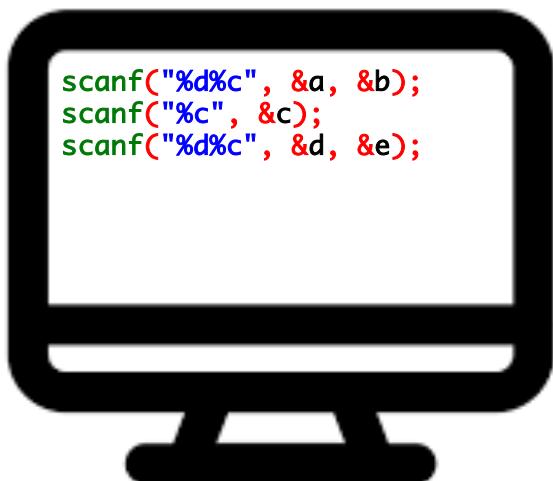


E que a pessoa tenha
preenchido o **buffer**
dessa forma.

Suponha agora
essa sequência
de **scans**.



```
scanf("%d%c", &a, &b);
scanf("%c", &c);
scanf("%d%c", &d, &e);
```



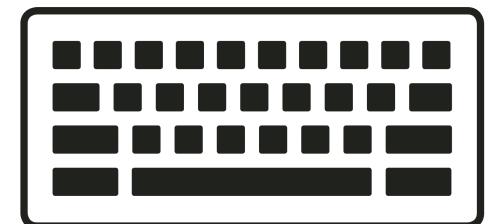
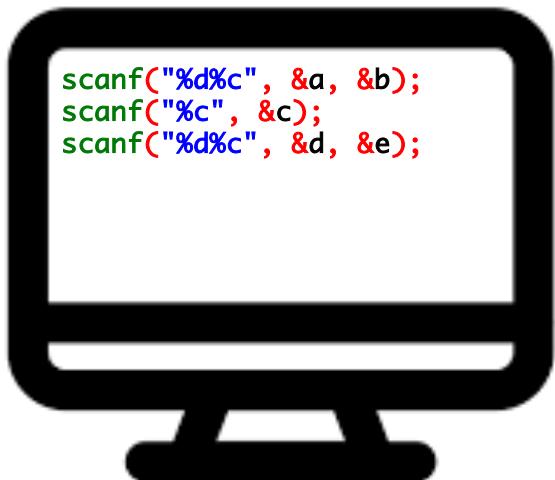
E que a pessoa tenha preenchido o **buffer** dessa forma.

Suponha agora essa sequência de **scans**.

```
scanf("%d%c", &a, &b);
scanf("%c", &c);
scanf("%d%c", &d, &e);
```



Às variáveis **a**, **b**, **c**, **d** e **e** serão atribuídos os valores 25, '\n', 'k', 15 e '\n', respectivamente.



E que a pessoa tenha preenchido o **buffer** dessa forma.

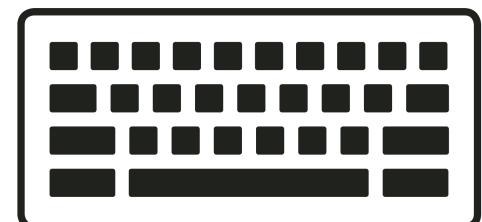
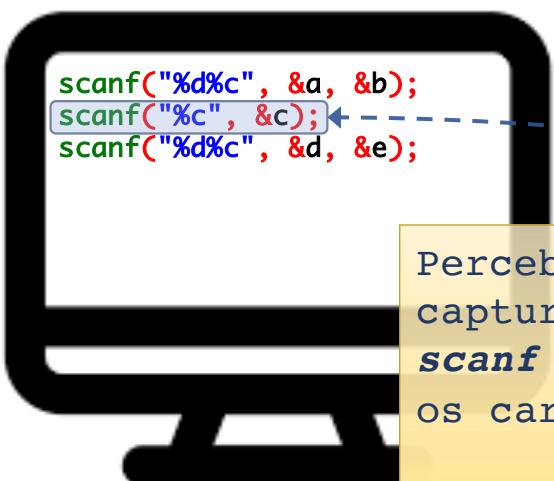
Suponha agora essa sequência de **scanf**s.

```
scanf("%d%c", &a, &b);
scanf("%c", &c);
scanf("%d%c", &d, &e);
```



Às variáveis **a**, **b**, **c**, **d** e **e** serão atribuídos os valores 25, '\n', 'k', 15 e '\n', respectivamente.

Perceba que não colocamos o %c adicional para captura do '\n' remanescente pois o próximo **scanf** é de um %d e os tipos numéricos ignoram os caracteres brancos do início do **buffer**.



E que a pessoa tenha preenchido o **buffer** dessa forma.

Suponha agora essa sequência de **scans**.

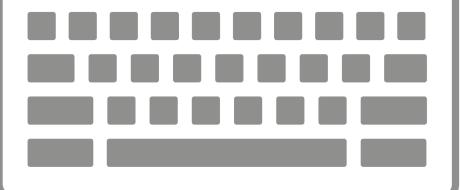
```
scanf("%d%c", &a, &b);
scanf("%c", &c);
scanf("%d%c", &d, &e);
```

Bom, o problema parece então estar resolvido: basta colocar um **%c** adicional, antes do **scanf** de um **%c**, para que não sobre um '**\n**' no **buffer**.



As variáveis **a**, **b**, **c**, **d** e **e** serão atribuídos os valores 25, '\n', 'k', 15 e '\n', respectivamente.

Perceba que não colocamos o **%c** adicional para captura do '**\n**' remanescente pois o próximo **scanf** é de um **%d** e os tipos numéricos ignoram os caracteres brancos do início do **buffer**.



E que a pessoa tenha preenchido o **buffer** dessa forma.

Suponha agora essa sequência de **scans**.

```
scanf("%d%c", &a, &b);
scanf("%c", &c);
scanf("%d%c", &d, &e);
```

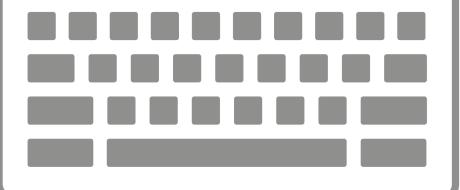
Bom, o problema parece então estar resolvido: basta colocar um **%c** adicional, antes do **scanf** de um **%c**, para que não sobre um '**\n**' no **buffer**.

Mas é meio chato ter que declarar variáveis para que recebam esses '**\n**'...



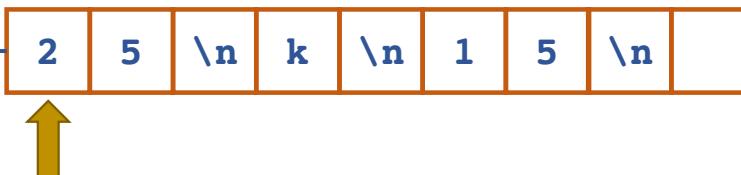
As variáveis **a**, **b**, **c**, **d** e **e** serão atribuídos os valores 25, '\n', 'k', 15 e '\n', respectivamente.

Perceba que não colocamos o **%c** adicional para captura do '**\n**' remanescente pois o próximo **scanf** é de um **%d** e os tipos numéricos ignoram os caracteres brancos do início do **buffer**.



E que a pessoa tenha preenchido o **buffer** dessa forma.

Suponha agora essa sequência de **scans**.

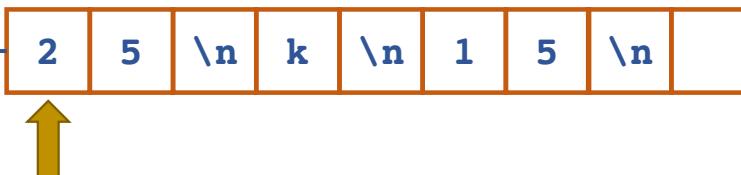


```
scanf("%d%c", &a);
scanf("%c", &b);
scanf("%d%c", &c);
```

Se colocarmos um * no formatador, entre o % e o identificador do tipo, como esses **%*c**, estamos instruindo o **scanf** a ler um caractere mas ignorá-lo, não armazenando esse valor em nenhuma variável.

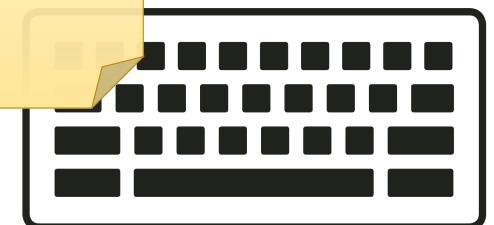
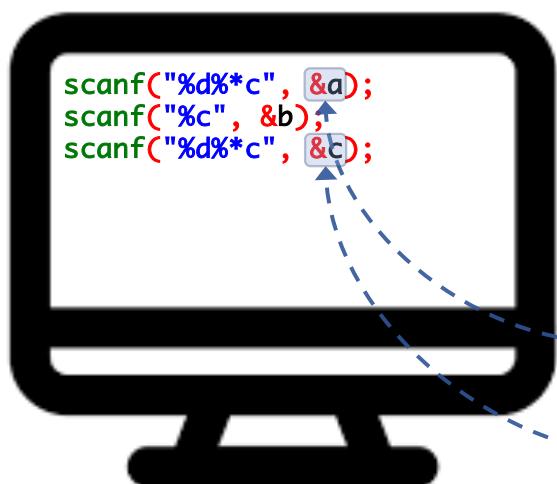
E que a pessoa tenha preenchido o **buffer** dessa forma.

Suponha agora essa sequência de **scanf**s.



Se colocarmos um * no formatador, entre o % e o identificador do tipo, como esses %*c, estamos instruindo o **scanf** a ler um caractere mas ignorá-lo, não armazenando esse valor em nenhuma variável.

Então não precisamos colocar as variáveis no **scanf**.



E que a pessoa tenha preenchido o **buffer** dessa forma.

Suponha agora essa sequência de **scanf**s.

```
scanf("%d%c", &a);
scanf("%c", &b);
scanf("%d%c", &c);
```

No exemplo ilustrado as variáveis **a**, **b** e **c** recebem os valores **25**, **'k'** e **15**, respectivamente, e, ao final, o **buffer** ficará vazio.



Se colocarmos um * no formatador, entre o % e o identificador do tipo, como esses %*c, estamos instruindo o **scanf** a ler um caractere mas ignorá-lo, não armazenando esse valor em nenhuma variável.

Então não precisamos colocar as variáveis no **scanf**.

E que a pessoa tenha preenchido o **buffer** dessa forma.

Suponha agora essa sequência de **scanf**s.

```
scanf("%d%c", &a);
scanf("%c", &b);
scanf("%d%c", &c);
```

No exemplo ilustrado as variáveis **a**, **b** e **c** recebem os valores **25**, **'k'** e **15**, respectivamente, e, ao final, o **buffer** ficará vazio.

[Vídeo com exemplos.](#)

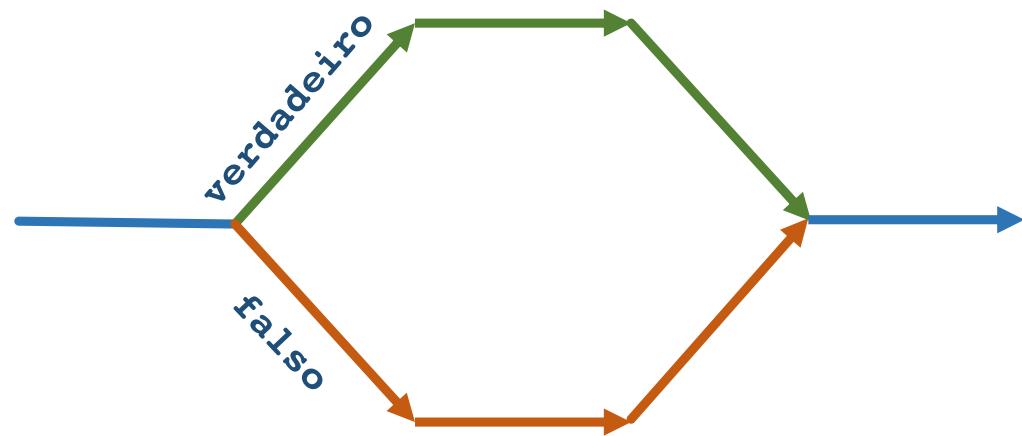


Se colocarmos um * no formatador, entre o % e o identificador do tipo, como esses %*c, estamos instruindo o **scanf** a ler um caractere mas ignorá-lo, não armazenando esse valor em nenhuma variável.

Então não precisamos colocar as variáveis no **scanf**.

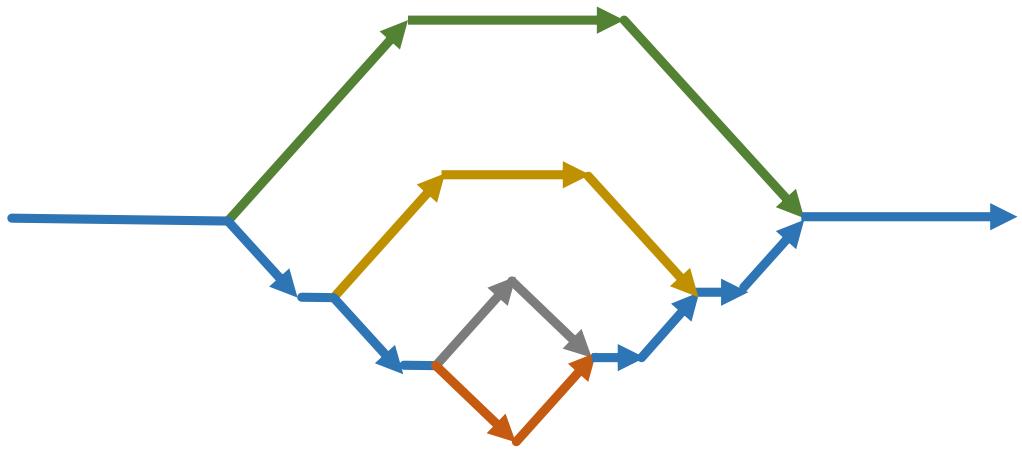
Desvios múltiplos

O desvio condicional ***if*** permite ao programa decidir por ***dois caminhos distintos***, de acordo com o contexto do programa no momento da execução, avaliado através de uma condição.



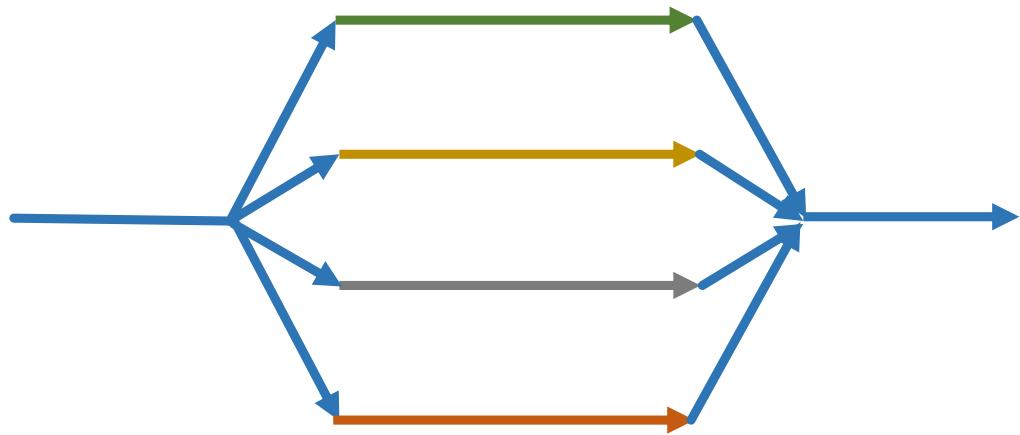
```
if (a > b){  
    puts("a eh maior do que b.");  
}else{  
    puts("b eh maior do que a.");  
}
```

Com algum esforço conseguimos aninhar os *ifs* e criar um código com múltiplos desvios.



```
if ( idade < 13 ){
    puts("Criança");
}else if ( idade < 21 ){
    puts("Jovem");
}else if ( idade < 60 ){
    puts("Adulto");
}else{
    puts("Idoso");
}
```

Mas já existe uma estrutura de controle que consegue criar os **desvios múltiplos** nativamente: o **switch-case**.



```
switch (genero){  
    case 'h':  
        puts("Homem, cis ou nao");  
        break;  
    case 'm':  
        puts("Mulher, cis ou nao");  
        break;  
    case 'n':  
        puts("Nao binario");  
        break;  
    default:  
        puts("Outros");  
}
```

Essa é a sintaxe do desvio ***switch-case***.

```
switch (exp){  
    case vl :  
        bloco de instruções;  
    default:  
        bloco de instruções;  
}
```

Essa é a sintaxe do desvio ***switch-case***.

Nesse espaço colocamos uma **expressão**, uma **constante** ou uma **variável**. O importante é que o **resultado** seja um **número inteiro** ou um **caractere** (que é um número inteiro também).

```
switch (exp){  
    case vl:  
        bloco de instruções;  
    default:  
        bloco de instruções;  
}
```

Essa é a sintaxe do desvio **switch-case**.

Nesse espaço colocamos uma **expressão**, uma **constante** ou uma **variável**. O importante é que o **resultado** seja um **número inteiro** ou um **caractere** (que é um **número inteiro** também).

Em cada cláusula **case** associamos um **valor constante inteiro** ou caractere. Só são permitidas expressões se não fizerem uso de variáveis, como na expressão **1+7**.

```
switch (exp){  
    case vl:  
        bloco de instruções;  
    default:  
        bloco de instruções;  
}
```

Essa é a sintaxe do desvio **switch-case**.

Nesse espaço colocamos uma **expressão**, uma **constante** ou uma **variável**. O importante é que o **resultado** seja um **número inteiro** ou um **caractere** (que é um **número inteiro** também).

Em cada cláusula **case** associamos um **valor constante inteiro** ou caractere. Só são permitidas expressões se não fizerem uso de variáveis, como na expressão **1+7**.

Cada cláusula **case** ou **default** acompanha um **bloco de instruções** com as instruções específicas daquele desvio.

```
switch (exp){  
    case vl:  
        bloco de instruções;  
    default:  
        bloco de instruções;  
}
```

Um exemplo do desvio ***switch-case***.

```
switch (dia){  
    case 1:  
        puts("Domingo");  
    case 2:  
        puts("Segunda-feira");  
    case 3:  
        puts("Terca-feira");  
    case 4:  
        puts("Quarta-feira");  
    case 5:  
        puts("Quinta-feira");  
    case 6:  
        puts("Sexta-feira");  
    case 7:  
        puts("Sabado");  
    default:  
        puts("Valor inesperado");  
}
```

1 Ao chegar em uma instrução **switch**, a expressão entre parênteses é avaliada.

Suponha que o valor seja 12.

```
switch 1(dia){  
    case 1:  
        puts("Domingo");  
    case 2:  
        puts("Segunda-feira");  
    case 3:  
        puts("Terca-feira");  
    case 4:  
        puts("Quarta-feira");  
    case 5:  
        puts("Quinta-feira");  
    case 6:  
        puts("Sexta-feira");  
    case 7:  
        puts("Sabado");  
    default:  
        puts("Valor inesperado");  
}
```

1 Ao chegar em uma instrução **switch**, a expressão entre parênteses é avaliada.

2 Com o resultado, o **switch** vai procurar em cada cláusula **case** uma com o mesmo valor.

Suponha que o valor seja 12.

```
switch(12){  
    case 1:  
        puts("Domingo");  
    case 2:  
        puts("Segunda-feira");  
    case 3:  
        puts("Terca-feira");  
    case 4:  
        puts("Quarta-feira");  
    case 5:  
        puts("Quinta-feira");  
    case 6:  
        puts("Sexta-feira");  
    case 7:  
        puts("Sabado");  
    default:  
        puts("Valor inesperado");  
}
```

1 Ao chegar em uma instrução **switch**, a expressão entre parênteses é avaliada.

2 Com o resultado, o **switch** vai procurar em cada cláusula **case** uma com o mesmo valor.

Suponha que o valor seja 12.

```
switch (dia){  
    case 1:  
        puts("Domingo");  
    case 2:  
        puts("Segunda-feira");  
    case 3:  
        puts("Terca-feira");  
    case 4:  
        puts("Quarta-feira");  
    case 5:  
        puts("Quinta-feira");  
    case 6:  
        puts("Sexta-feira");  
    case 7:  
        puts("Sabado");  
    default:  
        puts("Valor inesperado");  
}
```

1 Ao chegar em uma instrução **switch**, a expressão entre parênteses é avaliada.

2 Com o resultado, o **switch** vai procurar em cada cláusula **case** uma com o mesmo valor.

Suponha que o valor seja 12.

```
switch (dia){  
    case 1:  
        puts("Domingo");  
    case 2:  
        puts("Segunda-feira");  
    case 3:  
        puts("Terca-feira");  
    case 4:  
        puts("Quarta-feira");  
    case 5:  
        puts("Quinta-feira");  
    case 6:  
        puts("Sexta-feira");  
    case 7:  
        puts("Sabado");  
    default:  
        puts("Valor inesperado");  
}
```

1 Ao chegar em uma instrução **switch**, a expressão entre parênteses é avaliada.

2 Com o resultado, o **switch** vai procurar em cada cláusula **case** uma com o mesmo valor.

Suponha que o valor seja 12.

```
switch (dia){  
    case 1:  
        puts("Domingo");  
    case 2:  
        puts("Segunda-feira");  
    case 3:  
        puts("Terca-feira");  
    case 4:  
        puts("Quarta-feira");  
    case 5:  
        puts("Quinta-feira");  
    case 6:  
        puts("Sexta-feira");  
    case 7:  
        puts("Sabado");  
    default:  
        puts("Valor inesperado");  
}
```

1 Ao chegar em uma instrução **switch**, a expressão entre parênteses é avaliada.

2 Com o resultado, o **switch** vai procurar em cada cláusula **case** uma com o mesmo valor.

Suponha que o valor seja 12.

```
switch (dia){  
    case 1:  
        puts("Domingo");  
    case 2:  
        puts("Segunda-feira");  
    case 3:  
        puts("Terca-feira");  
    case 4:  
        puts("Quarta-feira");  
    case 5:  
        puts("Quinta-feira");  
    case 6:  
        puts("Sexta-feira");  
    case 7:  
        puts("Sabado");  
    default:  
        puts("Valor inesperado");  
}
```

1 Ao chegar em uma instrução **switch**, a expressão entre parênteses é avaliada.

2 Com o resultado, o **switch** vai procurar em cada cláusula **case** uma com o mesmo valor.

Suponha que o valor seja 12.

```
switch (dia){  
    case 1:  
        puts("Domingo");  
    case 2:  
        puts("Segunda-feira");  
    case 3:  
        puts("Terca-feira");  
    case 4:  
        puts("Quarta-feira");  
    case 5:  
        puts("Quinta-feira");  
    case 6:  
        puts("Sexta-feira");  
    case 7:  
        puts("Sabado");  
    default:  
        puts("Valor inesperado");  
}
```

1 Ao chegar em uma instrução **switch**, a expressão entre parênteses é avaliada.

2 Com o resultado, o **switch** vai procurar em cada cláusula **case** uma com o mesmo valor.

Suponha que o valor seja 12.

```
switch (dia){  
    case 1:  
        puts("Domingo");  
    case 2:  
        puts("Segunda-feira");  
    case 3:  
        puts("Terca-feira");  
    case 4:  
        puts("Quarta-feira");  
    case 5:  
        puts("Quinta-feira");  
    case 6:  
        puts("Sexta-feira");  
    case 7:  
        puts("Sabado");  
    default:  
        puts("Valor inesperado");  
}
```

1 Ao chegar em uma instrução **switch**, a expressão entre parênteses é avaliada.

2 Com o resultado, o **switch** vai procurar em cada cláusula **case** uma com o mesmo valor.

2 Apenas quando não ocorre o casamento do valor da expressão com algum **case** é que cláusula **default** é escolhida. O **default** é opcional.

Suponha que o valor seja 12.

```
switch (dia){  
    case 1:  
        puts("Domingo");  
    case 2:  
        puts("Segunda-feira");  
    case 3:  
        puts("Terca-feira");  
    case 4:  
        puts("Quarta-feira");  
    case 5:  
        puts("Quinta-feira");  
    case 6:  
        puts("Sexta-feira");  
    case 7:  
        puts("Sabado");  
    default:  
        puts("Valor inesperado");  
}
```

1 Ao chegar em uma instrução **switch**, a expressão entre parênteses é avaliada.

2 Com o resultado, o **switch** vai procurar em cada cláusula **case** uma com o mesmo valor.

2 Apenas quando não ocorre o casamento do valor da expressão com algum **case** é que cláusula **default** é escolhida. O **default** é opcional.

3 Quando ocorre o casamento com um dos **cases** ou com o **default**, o **bloco associado** à essa cláusula é executado.

Suponha que o valor seja 12.

```
switch (dia){  
    case 1:  
        puts("Domingo");  
    case 2:  
        puts("Segunda-feira");  
    case 3:  
        puts("Terca-feira");  
    case 4:  
        puts("Quarta-feira");  
    case 5:  
        puts("Quinta-feira");  
    case 6:  
        puts("Sexta-feira");  
    case 7:  
        puts("Sabado");  
    default:  
        puts("Valor inesperado");  
}
```

1 Ao chegar em uma instrução **switch**, a expressão entre parênteses é avaliada.

2 Com o resultado, o **switch** vai procurar em cada cláusula **case** uma com o mesmo valor.

2 Apenas quando não ocorre o casamento do valor da expressão com algum **case** é que cláusula **default** é escolhida. O **default** é opcional.

3 Quando ocorre o casamento com um dos **cases** ou com o **default**, o **bloco associado** à essa cláusula é executado.

4 Depois o **switch** é encerrado e o programa segue.

Suponha que o valor seja 12.

```
switch (dia){  
    case 1:  
        puts("Domingo");  
    case 2:  
        puts("Segunda-feira");  
    case 3:  
        puts("Terca-feira");  
    case 4:  
        puts("Quarta-feira");  
    case 5:  
        puts("Quinta-feira");  
    case 6:  
        puts("Sexta-feira");  
    case 7:  
        puts("Sabado");  
    default:  
        puts("Valor inesperado");  
}
```

1 Ao chegar em uma instrução **switch**, a expressão entre parênteses é avaliada.

2 Com o resultado, o **switch** vai procurar em cada cláusula **case** uma com o mesmo valor.

3 Apenas quando não ocorre o casamento do valor da expressão com algum **case** é que cláusula **default** é escolhida. O **default** é opcional.

4 Quando ocorre o casamento com um dos **cases** ou com o **default**, o bloco associado à essa cláusula é executado.

Depois o **switch** é encerrado e o programa segue.

Agora suponha que o valor seja 5.

```
switch 1 dia){  
    case 1:  
        puts("Domingo");  
    case 2:  
        puts("Segunda-feira");  
    case 3:  
        puts("Terca-feira");  
    case 4:  
        puts("Quarta-feira");  
    case 5:  
        puts("Quinta-feira");  
    case 6:  
        puts("Sexta-feira");  
    case 7:  
        puts("Sabado");  
    default:  
        2 puts("Valor inesperado");  
    }  
    3  
    4
```

Com **dia** sendo igual a **5**, será selecionada essa cláusula **case**.

```
switch (dia){  
    case 1:  
        puts("Domingo");  
    case 2:  
        puts("Segunda-feira");  
    case 3:  
        puts("Terca-feira");  
    case 4:  
        puts("Quarta-feira");  
    case 5:  
        puts("Quinta-feira");  
    case 6:  
        puts("Sexta-feira");  
    case 7:  
        puts("Sabado");  
    default:  
        puts("Valor inesperado");  
}
```

Agora suponha que o valor seja **5**.

Com **dia** sendo igual a **5**, será selecionada essa cláusula **case**.

Com isso, esperamos que o **bloco de código** dela seja executado.

```
switch (dia){  
    case 1:  
        puts("Domingo");  
    case 2:  
        puts("Segunda-feira");  
    case 3:  
        puts("Terca-feira");  
    case 4:  
        puts("Quarta-feira");  
    case 5:  
        puts("Quinta-feira");  
    case 6:  
        puts("Sexta-feira");  
    case 7:  
        puts("Sabado");  
    default:  
        puts("Valor inesperado");  
}
```

Agora suponha que o valor seja **5**.

Com `dia` sendo igual a `5`, será selecionada essa cláusula `case`.

Com isso, esperamos que o **bloco de código** dela seja executado.

E ele até é executado. Mas, uma vez que ocorre o casamento com um `case`, são executados todos os blocos de códigos da cláusula do casamento em diante.

Agora suponha que o valor seja `5`.

```
switch (dia){  
    case 1:  
        puts("Domingo");  
    case 2:  
        puts("Segunda-feira");  
    case 3:  
        puts("Terca-feira");  
    case 4:  
        puts("Quarta-feira");  
    case 5:  
        puts("Quinta-feira");  
    case 6:  
        puts("Sexta-feira");  
    case 7:  
        puts("Sabado");  
    default:  
        puts("Valor inesperado");  
}
```

Com `dia` sendo igual a `5`, será selecionada essa cláusula `case`.

Com isso, esperamos que o **bloco de código** dela seja executado.

E ele até é executado. Mas, uma vez que ocorre o casamento com um `case`, são executados todos os blocos de códigos da cláusula do casamento em diante.

Normalmente não é esse o comportamento desejado. Muitas vezes o ideal é que apenas o código de um único `case` seja executado.

Agora suponha que o valor seja `5`.

```
switch (dia){  
    case 1:  
        puts("Domingo");  
    case 2:  
        puts("Segunda-feira");  
    case 3:  
        puts("Terca-feira");  
    case 4:  
        puts("Quarta-feira");  
    case 5:  
        puts("Quinta-feira");  
    case 6:  
        puts("Sexta-feira");  
    case 7:  
        puts("Sabado");  
    default:  
        puts("Valor inesperado");  
}
```

Normalmente não é esse o comportamento desejado. Muitas vezes o ideal é que apenas o código de um único **case** seja executado.

Para implementar esse comportamento, podemos adicionar o **desvio incondicional break** ao final de cada bloco.

```
switch (dia){  
    case 1:  
        puts("Domingo");  
        break;  
    case 2:  
        puts("Segunda-feira");  
        break;  
    case 3:  
        puts("Terca-feira");  
        break;  
    case 4:  
        puts("Quarta-feira");  
        break;  
    case 5:  
        puts("Quinta-feira");  
        break;  
    case 6:  
        puts("Sexta-feira");  
        break;  
    case 7:  
        puts("Sabado");  
        break;  
    default:  
        puts("Valor inesperado");  
}
```

Normalmente não é esse o comportamento desejado. Muitas vezes o ideal é que apenas o código de um único **case** seja executado.

Para implementar esse comportamento, podemos adicionar o **desvio incondicional break** ao final de cada bloco.

```
switch (dia){  
    case 1:  
        puts("Domingo");  
        break;  
    case 2:  
        puts("Segunda-feira");  
        break;  
    case 3:  
        puts("Terca-feira");  
        break;  
    case 4:  
        puts("Quarta-feira");  
        break;  
    case 5:  
        puts("Quinta-feira");  
        break;  
    case 6:  
        puts("Sexta-feira");  
        break;  
    case 7:  
        puts("Sabado");  
        break;  
    default:  
        puts("Valor inesperado");  
}
```

Normalmente não é esse o comportamento desejado. Muitas vezes o ideal é que apenas o código de um único **case** seja executado.

Para implementar esse comportamento, podemos adicionar o **desvio incondicional break** ao final de cada bloco.

```
switch (dia){  
    case 1:  
        puts("Domingo");  
        break;  
    case 2:  
        puts("Segunda-feira");  
        break;  
    case 3:  
        puts("Terca-feira");  
        break;  
    case 4:  
        puts("Quarta-feira");  
        break;  
    case 5:  
        puts("Quinta-feira");  
        break;  
    case 6:  
        puts("Sexta-feira");  
        break;  
    case 7:  
        puts("Sabado");  
        break;  
    default:  
        puts("Valor inesperado");  
}
```

Agora suponha que o valor seja 3.

Com **dia** sendo igual a **3**, será selecionada essa cláusula **case**.

Agora suponha que o valor seja **3**.

```
switch (dia){  
    case 1:  
        puts("Domingo");  
        break;  
    case 2:  
        puts("Segunda-feira");  
        break;  
    case 3:  
        puts("Terca-feira");  
        break;  
    case 4:  
        puts("Quarta-feira");  
        break;  
    case 5:  
        puts("Quinta-feira");  
        break;  
    case 6:  
        puts("Sexta-feira");  
        break;  
    case 7:  
        puts("Sabado");  
        break;  
    default:  
        puts("Valor inesperado");  
}
```

Com `dia` sendo igual a `3`, será selecionada essa cláusula `case`.

E o bloco de código a ser executado será apenas esse.

Agora suponha que o valor seja `3`.

```
switch (dia){  
    case 1:  
        puts("Domingo");  
        break;  
    case 2:  
        puts("Segunda-feira");  
        break;  
    case 3:  
        puts("Terca-feira");  
        break;  
    case 4:  
        puts("Quarta-feira");  
        break;  
    case 5:  
        puts("Quinta-feira");  
        break;  
    case 6:  
        puts("Sexta-feira");  
        break;  
    case 7:  
        puts("Sabado");  
        break;  
    default:  
        puts("Valor inesperado");  
}
```

Com **dia** sendo igual a **3**, será selecionada essa cláusula **case**.

E o **bloco de código** a ser executado será apenas esse.

Pois assim que for executada, a instrução **break** desviará o fluxo de execução do programa para a próxima instrução após o **switch** - *independente de qualquer condição*.

Agora suponha que o valor seja **3**.

```
switch (dia){  
    case 1:  
        puts("Domingo");  
        break;  
    case 2:  
        puts("Segunda-feira");  
        break;  
    case 3:  
        puts("Terca-feira");  
        break;  
    case 4:  
        puts("Quarta-feira");  
        break;  
    case 5:  
        puts("Quinta-feira");  
        break;  
    case 6:  
        puts("Sexta-feira");  
        break;  
    case 7:  
        puts("Sabado");  
        break;  
    default:  
        puts("Valor inesperado");  
}
```

Com `dia` sendo igual a `3`, será selecionada essa cláusula `case`.

E o bloco de código a ser executado será apenas esse.

Pois assim que for executada, a instrução `break` desviará o fluxo de execução do programa para a próxima instrução após o `switch` - independente de qualquer condição.

[Vídeo com exemplos.](#)

```
switch (dia){  
    case 1:  
        puts("Domingo");  
        break;  
    case 2:  
        puts("Segunda-feira");  
        break;  
    case 3:  
        puts("Terca-feira");  
        break;  
    case 4:  
        puts("Quarta-feira");  
        break;  
    case 5:  
        puts("Quinta-feira");  
        break;  
    case 6:  
        puts("Sexta-feira");  
        break;  
    case 7:  
        puts("Sabado");  
        break;  
    default:  
        puts("Valor inesperado");  
}
```

Agora suponha que o valor seja `3`.

Desvios incondicionais

Desvio incondicional *break*

- Ao ser executado, *desvia* o fluxo de execução do programa, normalmente encerrando a execução da estrutura de controle onde está inserido.
 - Pode ser usado apenas dentro do **switch-case** e dos **laços de repetição**.
 - O fluxo de execução do programa passa para a próxima instrução após a estrutura de controle contendo o **break**.

Exemplos do **break** em laços de repetição.

```
i = 0;  
while (i++ < 10){  
    printf("%d\n", i);  
    if (i == 5){  
        break;  
    }  
}
```

```
i = 0;  
do{  
    printf("%d\n", ++i);  
    if (i == 5){  
        break;  
    }  
}while (i < 10);
```

```
for (i = 1; i <= 10; i++){  
    printf("%d\n", i);  
    if (i == 5){  
        break;  
    }  
}
```

Exemplos do ***break*** em laços de repetição.

```
i = 0;  
while (i++ < 10){  
    printf("%d\n", i);  
    if (i == 5){  
        break;--  
    }  
}
```

```
i = 0;  
do{  
    printf("%d\n", ++i);  
    if (i == 5){  
        break;--  
    }  
}while (i < 10);
```

```
for (i = 1; i <= 10; i++){  
    printf("%d\n", i);  
    if (i == 5){  
        break;--  
    }  
}
```

Em todos os casos, assim que a instrução ***break*** for atingida, os laços de repetição serão imediatamente encerrados e o programa seguirá para a próxima instrução após eles.

Então, quais são as saídas desses programas?

Exemplos do **break** em laços de repetição.

```
i = 0;  
while (i++ < 10){  
    printf("%d\n", i);  
    if (i == 5){  
        break;--  
    }  
}
```

```
i = 0;  
do{  
    printf("%d\n", ++i);  
    if (i == 5){  
        break;--  
    }  
}while (i < 10);
```

```
for (i = 1; i <= 10; i++){  
    printf("%d\n", i);  
    if (i == 5){  
        break;--  
    }  
}
```

Em todos os casos, assim que a instrução **break** for atingida, os laços de repetição serão imediatamente encerrados e o programa seguirá para a próxima instrução após eles.

Então, quais são as saídas desses programas?

Todos os trechos exibidos imprimem:

1
2
3
4
5

Problema Idades.

```
#include <stdio.h>

int main(void){
    int idade, soma = 0, q = 0;

    while(1){
        scanf("%d", &idade);
        if (idade < 0){
            break;
        }
        soma += idade;
        q++;
    }

    printf("%.2lf\n", (double) soma/q);

    return 0;
}
```

Problema Senha Fixa.

```
#include <stdio.h>

int main(void){
    int senha;

    while(1){
        scanf("%d", &senha);
        if (senha == 2002){
            break;
        }
        puts("Senha Invalida");
    }

    puts("Acesso Permitido");

    return 0;
}
```

Os **breaks** são muitos utilizados em conjunto com os *loops* infinitos propositais.

Problema Idades.

```
#include <stdio.h>

int main(void){
    int idade, soma = 0, q = 0;
    while(1){
        scanf("%d", &idade);
        if (idade < 0){
            break;
        }
        soma += idade;
        q++;
    }
    printf("%.2lf\n", (double) soma/q);
    return 0;
}
```

Problema Senha Fixa.

```
#include <stdio.h>

int main(void){
    int senha;
    while(1){
        scanf("%d", &senha);
        if (senha == 2002){
            break;
        }
        puts("Senha Invalida");
    }
    puts("Acesso Permitido");
    return 0;
}
```

Os **breaks** são muitos utilizados em conjunto com os *loops* infinitos propositais.

Problema Idades.

```
#include <stdio.h>

int main(void){
    int idade, soma = 0, q = 0;

    while(1){
        scanf("%d", &idade);
        if (idade < 0){
            break;
        }
        soma += idade;
        q++;
    }
    printf("%.2f", (float)soma/q);
    return 0;
}
```

Problema Senha Fixa.

```
#include <stdio.h>

int main(void){
    int senha;

    while(1){
        scanf("%d", &senha);
        if (senha == 2002){
            break;
        }
        puts("Senha Invalida");
    }
    return 0;
}
```

Sempre que eu crio um *loop* infinito eu tenho que colocar pelo menos um **break** para encerrar o laço.

Os **breaks** são muitos utilizados em conjunto com os *loops* infinitos propositais.

Problema Idades.

```
#include <stdio.h>

int main(void){
    int idade, soma = 0, q = 0;
    while(1){
        scanf("%d", &idade);
        if (idade < 0){
            break;
        }
        soma += idade;
        q++;
    }
    printf("%.2f", soma / q);
    return 0;
}
```

Essa é uma estratégia bastante utilizada e pode simplificar o código e melhorar a legibilidade, principalmente em problemas como esses, onde deve-se repetir um processamento até a entrada de determinado valor.

Um laço infinito pode ser criado de diversas maneiras, mas o nosso **jargão** de programadores é de usar sempre o **while(1)**. O que já serve até de indicação de que não foi um erro e sim algo proposital.

Perceba que o laço é infinito porque a sua condição, a constante **1**, é considerada **verdadeira** – e sempre será a cada vez que a condição for avaliada.

Sempre que eu crio um *loop* infinito eu tenho que colocar pelo menos um **break** para encerrar o laço.

return 0;

Desvio incondicional *continue*

- Ao ser executado, *desvia* o fluxo de execução do programa, normalmente encerrando a execução apenas da iteração atual – da repetição atual.
 - Pode ser usado apenas dentro dos **laços de repetição**.
 - No **while** e **do-while** o programa vai imediatamente avaliar a condição novamente.
 - No **for** o programa vai imediatamente avaliar a terceira instrução.

Exemplos do ***continue*** em laços de repetição.

```
i = 0;  
while (i++ < 10){  
    if (i == 5){  
        continue;  
    }  
    printf("%d\n", i);  
}
```

```
i = 0;  
do{  
    if (i == 5){  
        continue;  
    }  
    printf("%d\n", ++i);  
}while (i < 10);
```

```
for (i = 1; i <= 10; i++){  
    if (i == 5){  
        continue;  
    }  
    printf("%d\n", i);  
}
```

Exemplos do **continue** em laços de repetição.

```
i = 0;  
while (i++ < 10){  
    if (i == 5){  
        continue;---'  
    }  
    printf("%d\n", i);  
}
```

```
i = 0;  
do{  
    if (i == 5){  
        continue;-----'  
    }  
    printf("%d\n", ++i);  
}while (i < 10);  
-----'
```

```
for (i = 1; i <= 10; i++){  
    if (i == 5){  
        continue;-----'  
    }  
    printf("%d\n", i);  
}
```

Em todos os casos, assim que a instrução **continue** for atingida, a iteração atual dos laços de repetição será imediatamente encerrada e o programa passará a se preparar para a próxima iteração, avaliando a condição ou incrementando o contador.

Então, quais são as saídas desses programas?

Exemplos do **continue** em laços de repetição.

```
i = 0;  
while (i++ < 10){  
    if (i == 5){  
        continue;---'  
    }  
    printf("%d\n", i);  
}
```

```
i = 0;  
do{  
    if (i == 5){  
        continue;-----'  
    }  
    printf("%d\n", ++i);  
}while (i < 10);  
                  ↑
```

```
for (i = 1; i <= 10; i++){  
    if (i == 5){  
        continue;-----'  
    }  
    printf("%d\n", i);  
}
```

Em todos os casos, assim que a instrução **continue** for atingida, a **iteração atual** dos laços de repetição será imediatamente encerrada e o programa passará a se preparar para a próxima iteração, avaliando a condição ou incrementando o contador.

Então, quais são as saídas desses programas?

A iteração que imprimiria o **5** é pulada.

Todos os trechos exibidos imprimem:

1
2
3
4
6
7
8
9
10

Exemplos do ***continue*** em laços de repetição.

```
i = 0;  
while (i++ < 10){  
    if (i == 5){  
        continue;----'  
    }  
    printf("%d\n", i);  
}
```

```
i = 0;  
do{  
    if (i == 5){  
        continue;----'  
    }  
    printf("%d\n", ++i);  
}while (i < 10);
```

```
for (i = 1; i <= 10; i++){  
    if (i == 5){  
        continue;----'  
    }  
    printf("%d\n", i);  
}
```

Em todos os casos, assim que a instrução ***continue*** for atingida, a iteração atual dos laços de repetição será imediatamente encerrada e o programa passará a se preparar para a próxima iteração, avaliando a condição ou incrementando o contador.

Então, quais são as saídas desses programas?

A iteração que imprimiria o **5** é pulada.

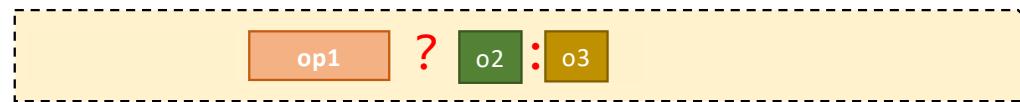
[Vídeo](#) com exemplos de ***break*** e ***continue***.

Todos os trechos exibidos imprimem:

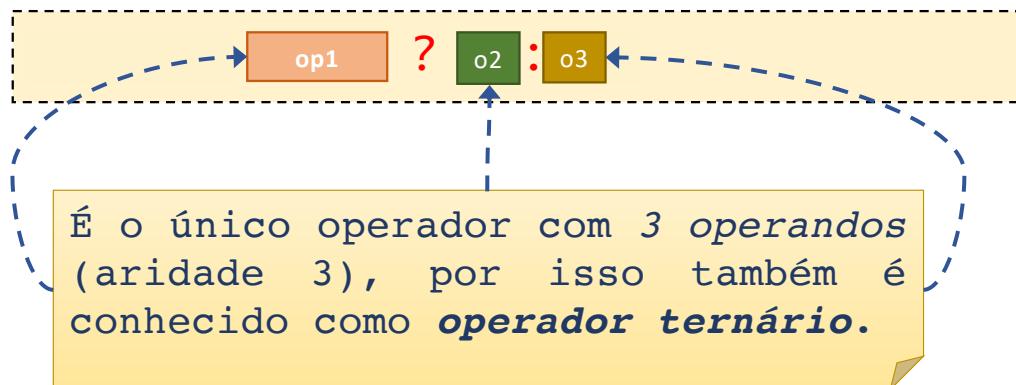
1
2
3
4
6
7
8
9
10

Operador condicional

Essa é a sintaxe do *operador condicional*.



Essa é a sintaxe do *operador condicional*.



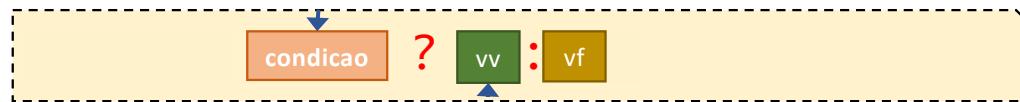
Essa é a sintaxe do **operador condicional**.

O primeiro operando a ser avaliado é o primeiro, ele funciona como uma **condição**. Portanto, o valor da **expressão**, **variável** ou **constante** aqui será considerado um valor **booleano**, verdadeiro ou falso.



Essa é a sintaxe do **operador condicional**.

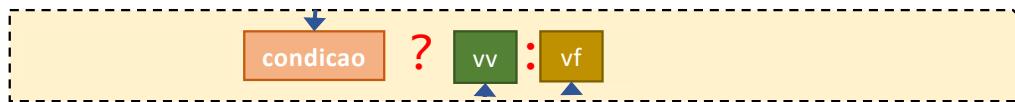
O primeiro operando a ser avaliado é o primeiro, ele funciona como uma **condição**. Portanto, o valor da **expressão**, **variável** ou **constante** aqui será considerado um valor **booleano**, verdadeiro ou falso.



Quando o resultado do **operando 1** for **verdadeiro**, o resultado do **operador condicional** será o resultado do **operando 2**.

Essa é a sintaxe do **operador condicional**.

O primeiro operando a ser avaliado é o primeiro, ele funciona como uma **condição**. Portanto, o valor da **expressão**, **variável** ou **constante** aqui será considerado um valor **booleano**, verdadeiro ou falso.



Quando o resultado do **operando 1** for **verdadeiro**, o resultado do **operador condicional** será o resultado do **operando 2**.

Já quando o resultado do **operando 1** for **falso**, o resultado do **operador condicional** será o resultado do **operando 3**.

Exemplo de uso do ***operador condicional***.

```
maior = a > b ? a : b;
```

Exemplo de uso do *operador condicional*.

A variável **maior** receberá o resultado do operador condicional.

```
maior = [a > b ? a : b];
```

Exemplo de uso do *operador condicional*.

A variável **maior** receberá o resultado do operador condicional.

```
maior = a > b ? a : b;
```

1 Primeiramente o **operando 1** é avaliado.

Exemplo de uso do *operador condicional*.

A variável **maior** receberá o resultado do operador condicional.

```
maior = a > b ? a : b;
```

1 Primeiramente o **operando 1** é avaliado.

Quando o resultado é **verdadeiro**.

2 O **operando 2** é avaliado e o resultado dele será o resultado do operador condicional.

Exemplo de uso do *operador condicional*.

A variável **maior** receberá o resultado do operador condicional.

Esse será o efeito.

```
maior = a > b ? a : b;
```

1 Primeiramente o **operando 1** é avaliado.

Quando o resultado é **verdadeiro**.

2 O **operando 2** é avaliado e o resultado dele será o resultado do operador condicional.

Exemplo de uso do *operador condicional*.

A variável **maior** receberá o resultado do operador condicional.

```
maior = a > b ? a : b;
```

1 Primeiramente o **operando 1** é avaliado.

Quando o resultado é verdadeiro.

Quando o resultado é **falso**.

O **operando 2** é avaliado e o resultado dele será o resultado do operador condicional.

O **operando 3** é avaliado e o resultado dele será o resultado do operador condicional.

Exemplo de uso do *operador condicional*.

A variável **maior** receberá o resultado do operador condicional.

```
maior = a > b ? a : b;
```

1 Primeiramente o **operando 1** é avaliado.

Quando o resultado é verdadeiro.

Quando o resultado é **falso**.

O **operando 2** é avaliado e o resultado dele será o resultado do operador condicional.

O **operando 3** é avaliado e o resultado dele será o resultado do operador condicional.

Peso Ideal

```
#include <stdio.h>

int main(void){
    char sexo;
    float altura, pesoIdeal;

    printf("Digite sua altura em metros: ");
    scanf("%f%c", &altura);
    printf("Digite seu sexo,\n"
           "(m)asculino ou (f)eminino: ");
    scanf("%c", &sexo);

    if (sexo == 'm'){
        pesoIdeal = (72.7 * altura) - 58;
    }else{
        pesoIdeal = (62.1 * altura) - 44.7;
    }

    printf("O peso ideal para um(a) %c de "
           "altura %.2f eh: %.1fkg\n",
           sexo, altura, pesoIdeal);

    return 0;
}
```

Lembra desse
programa?

Peso Ideal

```
#include <stdio.h>

int main(void){
    char sexo;
    float altura, pesoIdeal;

    printf("Digite sua altura em metros: ");
    scanf("%f%c", &altura);
    printf("Digite seu sexo,\n"
           "(m)asculino ou (f)eminino: ");
    scanf("%c", &sexo);

    if (sexo == 'm'){
        pesoIdeal = (72.7 * altura) - 58;
    }else{
        pesoIdeal = (62.1 * altura) - 44.7;
    }

    printf("O peso ideal para um(a) %c de "
           "altura %.2f eh: %.1fkg\n",
           sexo, altura, pesoIdeal);

    return 0;
}
```

O operador condicional é muito utilizado para substituir esse tipo de desvio.

Peso Ideal

```
#include <stdio.h>

int main(void){
    char sexo;
    float altura, pesoIdeal;

    printf("Digite sua altura em metros: ");
    scanf("%f%c", &altura);
    printf("Digite seu sexo,\n"
           "(m)asculino ou (f)eminino: ");
    scanf("%c", &sexo);

    if (sexo == 'm'){
        pesoIdeal = (72.7 * altura) - 58;
    }else{
        pesoIdeal = (62.1 * altura) - 44.7;
    }

    printf("O peso ideal para um(a) %c de "
           "altura %.2f eh: %.1fkg\n",
           sexo, altura, pesoIdeal);

    return 0;
}
```

O operador condicional é muito utilizado para substituir esse tipo de desvio.

Esse é um desvio que apenas atribui um novo valor à variável **pesoIdeal**, escolhendo entre *dois valores diferentes* de acordo com uma *condição*.

Peso Ideal

```
#include <stdio.h>

int main(void){
    char sexo;
    float altura, pesoIdeal;

    printf("Digite sua altura em metros: ");
    scanf("%f%c", &altura);
    printf("Digite seu sexo,\n"
           "(m)asculino ou (f)eminino: ");
    scanf("%c", &sexo);

    if (sexo == 'm'){
        pesoIdeal = (72.7 * altura) - 58;
    }else{
        pesoIdeal = (62.1 * altura) - 44.7;
    }

    printf("O peso ideal para um(a) %c de "
           "altura %.2f eh: %.1fkg\n",
           sexo, altura, pesoIdeal);

    return 0;
}
```

`pesoIdeal = (sexo == 'm') ? (72.7 * altura) - 58 : (62.1 * altura) - 44.7;`

O operador condicional é muito utilizado para substituir esse tipo de desvio.

Esse é um desvio que apenas atribui um novo valor à variável **pesoIdeal**, escolhendo entre *dois valores diferentes* de acordo com uma *condição*.

Peso Ideal (v2)

```
#include <stdio.h>

int main(void){
    char sexo;
    float altura, pesoIdeal;

    printf("Digite sua altura em metros: ");
    scanf("%f%c", &altura);
    printf("Digite seu sexo, (m)asculino ou (f)eeminino: ");
    scanf("%c", &sexo);

    pesoIdeal = (sexo == 'm') ? (72.7 * altura) - 58 : (62.1 * altura) - 44.7;

    printf("O peso ideal para um(a) %c de altura %.2f eh: %.1fkg\n",
           sexo, altura, pesoIdeal);

    return 0;
}
```

Peso Ideal (v2)

```
#include <stdio.h>

int main(void){
    char sexo;
    float altura, pesoIdeal;

    printf("Digite sua altura em metros: ");
    scanf("%f%c", &altura);
    printf("Digite seu sexo, (m)asculino ou (f)eeminino: ");
    scanf("%c", &sexo);

    pesoIdeal = (sexo == 'm') ? (72.7 * altura) - 58 : (62.1 * altura) - 44.7;

    printf("O peso ideal para um(a) %c de altura %.2f eh: %.1fkg\n",
           sexo, altura, pesoIdeal);

    return 0;
}
```

Operador condicional.

Peso Ideal (v2)

```
#include <stdio.h>

int main(void){
    char sexo;
    float altura, pesoIdeal;

    printf("Digite sua altura em metros: ");
    scanf("%f%c", &altura);
    printf("Digite seu sexo, (m)asculino ou (f)eminino: ");
    scanf("%c", &sexo);

    pesoIdeal = (sexo == 'm') ? (72.7 * altura) - 58 : (62.1 * altura) - 44.7;

    printf("O peso ideal para um(a) %c de altura %.2f eh: %.1fkg\n",
           sexo, altura, pesoIdeal);

    return 0;
}
```

Percebeu também o `%*c` antes do `scanf` de um `%c`?

Operador condicional.

Fibonacci

```
#include <stdio.h>

int main(void){
    int n, i;
    int ant1, ant2, atual;

    printf("Digite o n: ");
    scanf("%d", &n);

    printf("Os %d primeiros termos da sequencia de Fibonacci sao: ", n);
    for (i = 1; i <= n; i = i + 1){
        if (i == 1 || i == 2){
            atual = 1;
        }else{
            atual = ant2 + ant1;
        }

        if (i > 1 && i != n){
            printf(", "); /* separar com uma virgula */
        }
        if (i == n && n > 1){
            printf(" e "); /* separar com a letra e */
        }

        printf("%d", atual);
        ant2 = ant1;
        ant1 = atual;
    }
    printf(".\n"); /* concluir a saida com um ponto final */

    return 0;
}
```

Fibonacci

```
#include <stdio.h>

int main(void){
    int n, i;
    int ant1, ant2, atual;

    printf("Digite o n: ");
    scanf("%d", &n);

    printf("Os %d primeiros termos da sequencia de Fibonacci sao: ", n);
    for (i = 1; i <= n; i = i + 1){
        if (i == 1 || i == 2){
            atual = 1;
        }else{
            atual = ant2 + ant1;
        }
        if (i > 1 && i != n){
            printf(", "); /* separar com uma virgula */
        }
        if (i == n && n > 1){
            printf(" e "); /* separar com a letra e */
        }
        printf("%d", atual);
        ant2 = ant1;
        ant1 = atual;
    }
    printf(".\n"); /* concluir a saida com um ponto final */

    return 0;
}
```

Dá pra substituir esses dois desvios com o uso do operador condicional. Como?

Fibonacci

```
#include <stdio.h>

int main(void){
    int n, i;
    int ant1, ant2, atual;

    printf("Digite o n: ");
    scanf("%d", &n);

    printf("Os %d primeiros termos da sequencia de Fibonacci sao: ", n);
    for (i = 1; i <= n; i = i + 1){
        if (i == 1 || i == 2){
            atual = 1;
        }else{
            atual = ant2 + ant1;
        }
        if (i > 1 && i != n){
            printf(", "); /* separar com uma virgula */
        }
        if (i == n && n > 1){
            printf(" e "); /* separar com a letra e */
        }
        printf("%d", atual);
        ant2 = ant1;
        ant1 = atual;
    }
    printf(".\n"); /* concluir a saida com um ponto final */

    return 0;
}
```

The code is a C program that prints the first n terms of the Fibonacci sequence. It uses a for loop to iterate from 1 to n . Inside the loop, it checks if i is 1 or 2, in which case $atual$ is set to 1. Otherwise, it calculates $atual$ as the sum of $ant2$ and $ant1$. After printing each term, it adds a comma (except for the last term) or the word "e" (for the second-to-last term). Finally, it prints a new line character.

Annotations:

- A dashed blue box highlights the assignment statement `atual = (i == 1 || i == 2) ? 1 : ant2 + ant1;`.
- A dashed blue box highlights the printf statement `printf((i > 1 && i != n) ? ", " : " e ");`.

Fibonacci (v2)

```
#include <stdio.h>

int main(void){
    int n, i;
    int ant1, ant2, atual;

    printf("Digite o n: ");
    scanf("%d", &n);

    printf("Os %d primeiros termos da sequencia de Fibonacci sao: ", n);
    for (i = 1; i <= n; i = i + 1){
        atual = (i == 1 || i == 2) ? 1 : ant2 + ant1;

        printf( (i > 1 && i != n) ? ", " : " e ");

        printf("%d", atual);
        ant2 = ant1;
        ant1 = atual;
    }
    printf(".\n");

    return 0;
}
```

Fibonacci (v2)

```
#include <stdio.h>

int main(void){
    int n, i;
    int ant1, ant2, atual;

    printf("Digite o n: ");
    scanf("%d", &n);

    printf("Os %d primeiros termos da sequencia de Fibonacci sao: ", n);
    for (i = 1; i <= n; i = i + 1){
        atual = (i == 1 || i == 2) ? 1 : ant2 + ant1;

        printf( (i > 1 && i != n) ? ", " : " e ");

        printf("%d", atual);
        ant2 = ant1;
        ant1 = atual;
    }
    printf(".\n");

    return 0;
}
```

[Vídeo](#) com esse e outros exemplos.

Operador vírgula

Essa é a sintaxe do *operador vírgula*.



Essa é a sintaxe do *operador vírgula*.

op1 , op2

O *operador vírgula* tem
dois operandos, separados
por uma vírgula.

Essa é a sintaxe do **operador vírgula**.

Ele tem ordem de avaliação bem definida. O **operando 1** é sempre avaliado primeiro que o **operando 2**.

op1 , op2

O **operador vírgula** tem **dois operandos**, separados por uma vírgula.

Essa é a sintaxe do **operador vírgula**.

Ele tem ordem de avaliação bem definida. O **operando 1** é sempre avaliado primeiro que o **operando 2**.

op1 , op2

O **operador vírgula** tem **dois operandos**, separados por uma vírgula.

Sua **precedência** é uma das mais **baixas** da linguagem e é **associado à direita**.

Essa é a sintaxe do **operador vírgula**.

Ele tem ordem de avaliação bem definida. O **operando 1** é sempre avaliado primeiro que o **operando 2**.

O **resultado** do operador é **sempre** o valor do **operando 2**.

op1 , op2

O **operador vírgula** tem **dois operandos**, separados por uma vírgula.

Sua **precedência** é uma das mais **baixas** da linguagem e é **associado à direita**.

Essa é a sintaxe do **operador vírgula**.

op1 , op2

O resultado do operador
é sempre o valor do
operando 2.

Ele não é muito utilizado e serve para permitir que o programador coloque duas expressões/instruções onde normalmente só uma é permitida.

Exemplo de uso do **operador vírgula**.

i = a++, a * a;

Exemplo de uso do **operador vírgula**.

A variável **i** receberá
o resultado do
operador vírgula.

```
i = a++, a * a;
```

Exemplo de uso do **operador vírgula**.

A variável **i** receberá o resultado do operador vírgula.

```
i = a++, a * a;
```

1 Primeiramente o **operando 1** é avaliado.

Isso garante que a variável **a** seja incrementada antes da avaliação do **operando 2**.

Exemplo de uso do **operador vírgula**.

A variável **i** receberá o resultado do operador vírgula.

`i = a++, a * a;`

1 Primeiramente o **operando 1** é avaliado.

Isso garante que a variável **a** seja incrementada antes da avaliação do **operando 2**.

2 Depois o **operando 2** é avaliado e o resultado dele será o resultado do operador vírgula.

Exemplo de uso do **operador vírgula**.

A variável **i** receberá o resultado do operador vírgula.

O efeito será **sempre** esse.

1 Primeiramente o **operando 1** é avaliado.

Isso garante que a variável **a** seja incrementada antes da avaliação do **operando 2**.

2 Depois o **operando 2** é avaliado e o resultado dele será o resultado do operador vírgula.

`i = a++, a * a;`

```
#include <stdio.h>

int main(){
    int i, j;

    j = 10;
    for(i = 1; i <= 10; i++){
        printf("(%d, %d)\n", i, j);
        j--;
    }

    return 0;
}
```

O que esse programa imprime ao executar?

```
#include <stdio.h>

int main(){
    int i, j;

    j = 10;
    for(i = 1; i <= 10; i++){
        printf("(%d, %d)\n", i, j);
        j--;
    }

    return 0;
}
```

O que esse programa imprime ao executar?

- (1, 10)
- (2, 9)
- (3, 8)
- (4, 7)
- (5, 6)
- (6, 5)
- (7, 4)
- (8, 3)
- (9, 2)
- (10, 1)

```
#include <stdio.h>

int main(){
    int i, j;
    j = 10;
    for(i = 1; i <= 10; i++){
        printf("(%d, %d)\n", i, j);
        j--;
    }
    return 0;
}
```

Podemos usar o **operador vírgula** para iniciar e decrementar o **j** junto com com a iniciação e incremento do **i**.

O que esse programa imprime ao executar?

- (1, 10)
- (2, 9)
- (3, 8)
- (4, 7)
- (5, 6)
- (6, 5)
- (7, 4)
- (8, 3)
- (9, 2)
- (10, 1)

```
#include <stdio.h>

int main(){
    int i, j;

    j = 10;
    for(i = 1; i <= 10; i++){
        printf("(%d, %d)\n", i, j);
        j--;
    }

    return 0;
}
```

Podemos usar o **operador vírgula** para iniciar e decrementar o **j** junto com com a iniciação e incremento do **i**.

```
#include <stdio.h>

int main(){
    int i, j;

    for(i = 1, j = 10; i <= 10; i++, j--){
        printf("(%d, %d)\n", i, j);
    }

    return 0;
}
```

```
#include <stdio.h>

int main(){
    int i, j;

    j = 10;
    for(i = 1; i <= 10; i++){
        printf("(%d, %d)\n", i, j);
        j--;
    }

    return 0;
}
```

Podemos usar o **operador vírgula** para iniciar e decrementar o **j** junto com a iniciação e incremento do **i**.

```
#include <stdio.h>

int main(){
    int i, j;

    for(i = 1, j = 10; i <= 10; i++, j--){
        printf("(%d, %d)\n", i, j);
    }

    return 0;
}
```

```
#include <stdio.h>

int main(){
    int i, j;

    j = 10;
    for(i = 1; i <= 10; i++){
        printf("(%d, %d)\n", i, j);
        j--;
    }

    return 0;
}
```

```
#include <stdio.h>

int main(){
    int i, j;

    for(i = 1, j = 10; i <= 10; i++, j--){
        printf("(%d, %d)\n", i, j);
    }

    return 0;
}
```

Essa situação, de fazer o for controlar duas variáveis, é como o operador vírgula é mais utilizado na prática.