



Estrutura de Dados

```
var atpos=inputs[i].index();
var dotpos=inputs[i].lastIndex();
if (atpos<1 || dotpos<atpos) {
    document.getElementById("text").value =
        inputs[i].value.substring(0, atpos) +
        "•" +
        inputs[i].value.substring(dotpos);
} else {
    document.getElementById("text").value =
        inputs[i].value.substring(0, atpos) +
        inputs[i].value.substring(dotpos);
}
```

**COMANDOS BÁSICOS
E CONTROLADORES
DE FLUXO**

APRESENTAÇÃO

Prezado(a) aluno(a), neste módulo conversaremos sobre os comandos básicos que nos auxiliam a escrever os códigos que precisamos, recebendo ou enviando comunicações ao usuário dos programas.

E também falaremos sobre os comandos controladores de fluxo, que nos permitem tomar decisões e efetuar repetições necessárias em nossos códigos.

Vamos começar?!

OBJETIVOS DE APRENDIZAGEM

Ao final desse módulo você deverá ser capaz de:

- Comunicar eficazmente com o usuário do programa;
- Identificar um comando de decisão, que permite executar códigos específicos dependendo da condição imposta;
- Diferenciar e utilizar comandos de repetição, que facilitam e otimizam a escrita de um código.

COMANDOS BÁSICOS E CONTROLADORES DE FLUXO

Uma boa comunicação com o usuário do programa é essencial para que possamos ser compreendidos, e, a partir de agora falaremos sobre os comandos necessários a esta ação, que nos permitirá comunicar com o usuário e também executarmos de forma eficiente as decisões que ele solicitar. Esta é uma parte muito importante da nossa lógica, por isso, fique atento!



Comandos básicos

Chamamos de comandos básicos aqueles que fazem uma comunicação com o usuário em uma escrita de programa. Para isto, é necessário utilizar comandos que nos permitam comunicar com usuário, conhecidos como comandos de saída, e, também, comandos que permita que o usuário se comunique conosco, conhecidos como comandos de entrada. Agora falaremos sobre cada um deles!



Comando de saída

O primeiro exemplo é o comando de saída. Para que você se comunique com o usuário do programa, é importante exibir mensagens que permitam essa troca de informações. Estas mensagens são exibidas pelos comandos de saída.

É importante se lembrar que quem está utilizando o programa não conhece o seu código, por isso sua comunicação com ele é essencial.

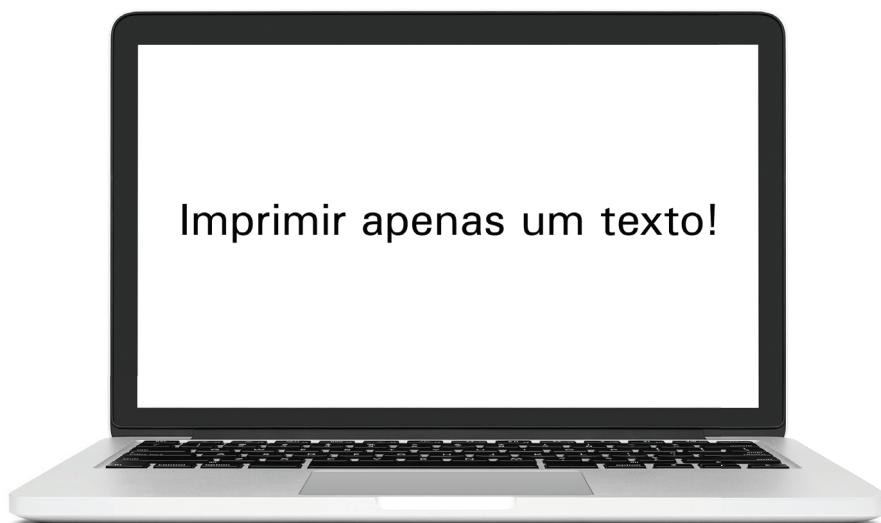
Apesar da linguagem C possuir vários comandos de entrada, neste estudo utilizaremos o comando `printf` para nossa comunicação. Ele pertence a biblioteca `stdio.h` e escreve o que for solicitado no cursor, além de permitir formatação para qualquer tipo de dado.

Sua utilização poderá ser dividida em três partes distintas, e muito comuns em linguagens de programação: a primeira, onde deseja-se exibir apenas um texto na tela; a segunda, onde deseja-se exibir apenas o conteúdo de uma variável na tela; e a última, onde deseja-se exibir um texto junto com o conteúdo de uma variável. Vamos compreender melhor como ele pode ser utilizado?

Se você deseja exibir apenas um texto na tela, basta colocar seu texto dentro do comando entre aspas duplas, da seguinte forma:

```
1 #include <stdio.h>
2 int main () {
3     printf ("\nImprimir apenas um texto!")
4     return 0;
5 }
```

Observe que o comando tem apenas uma frase, e saiba que ela será exibida ao usuário, exatamente como você a escreveu.



O caractere especial `\n` é utilizado para criar uma nova linha no texto, por isso não foi exibido na tela. A seguir você poderá ver uma tabela com outros caracteres especiais existentes.

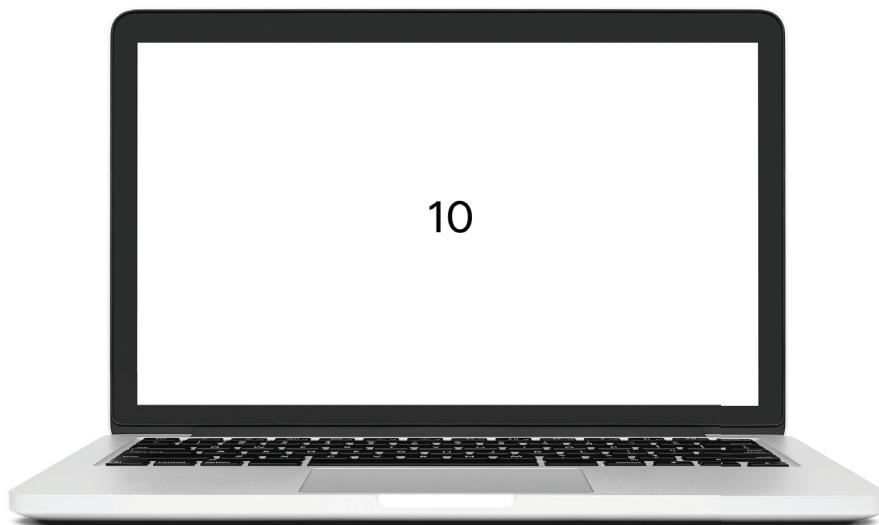


Se você deseja exibir apenas o conteúdo de uma variável, basta incluir, entre aspas, a formatação (*string* de controle) referente ao tipo da variável que utilizada, e, fora das aspas, indique a variável (nome) que terá seu conteúdo impresso. Vamos ver um exemplo para ficar mais claro!

```
1 #include <stdio.h>
2 int main () {
3     int tam = 10;
4     printf ("%d. ", tam);
5     return 0;
6 }
```

O caractere %d é chamado de *string* de controle, *string* de formatação ou caractere de controle. Ele é utilizado para indicar o formato que será utilizado para exibir na tela, neste caso, ele está indicando que será formatada como números inteiros (conteúdo de uma variável inteira).

Mais abaixo você poderá ver uma tabela com outros tipos de formatação possíveis.



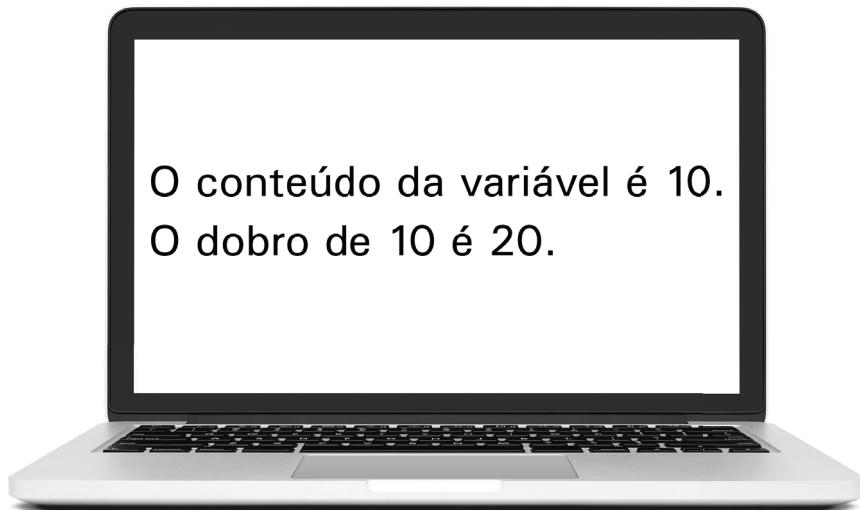
Se você deseja exibir um texto, juntamente com o conteúdo de uma variável, deverá escrever seu texto, entre as aspas duplas, e no local onde deseja exibir o conteúdo da sua variável, incluir uma *string* de controle. Após fechar as aspas, coloque sua(s) variável(is), ou cálculo(s), separados por vírgula, na ordem em que colocou as *string*(s) de controle.

EXEMPLO

```
1 #include <stdio.h>
2 int main () {
3     int tam = 10;
4     printf ("\nO conteúdo da variável é %d. ", tam);
5     printf ("\nO dobro de %d é %d. ", tam, tam * 2);
6     return 0;
7 }
```



O texto será exibido normalmente, e no local onde foi incluído o caractere %d aparecerá o conteúdo da variável e/ou o cálculo correspondente.



No exemplo, utilizei apenas variáveis inteiros, mas você pode utilizar outros tipos, veja a tabela!

TIPO DA VARIÁVEL	STRING DE CONTROLE
Cadeia de caracteres (strings)	%s
Char	%c
Double	%lf
Endereço de memória	%p
Float	%f
Int	%d ou %i

Para complementar a escrita do seu código, você também pode utilizar caracteres especiais que permitem tal formatação. Basta utilizar estes caracteres dentro do texto.

Veja os tipos disponíveis:

CARACTERE ESPECIAL	FUNÇÃO
\n	Nova linha
\t	Tabulação
\'	Imprimir aspas simples
\"	Imprimir aspas duplas
\\\	Imprimir \
\0	Nulo
\%	Imprimir %



Comando de entrada

Agora que você se comunicou com o usuário do seu programa, precisamos que ele se comunique com você.

O comando de entrada permite que algo digitado, ou “enviado”, ao computador, seja “lido” e utilizado no código.

Um dos comandos responsável por receber os dados enviados pelo usuário é o comando scanf. Ele pertence a biblioteca stdio.h.

Para utilizá-lo é necessário incluir a string de formatação (a mesma tabela anterior) e o endereço de memória da variável em que o conteúdo será armazenado. Vamos compreender melhor tudo isso!

```
scanf("string de controle", &variável);
```

Neste caso, onde:

- **String de controle:** indica qual será o tipo do dado a ser digitado (inteiro, caractere, real, etc). Ele deve ser colocado entre aspas.
- **Variável:** será o nome da variável que receberá o valor digitado, precedido de &. O & permite ao comando saber o endereço de memória da variável.

Vamos ver alguns exemplos da utilização deste comando:

EXEMPLO

```
1 #include <stdio.h>
2 int main (){
3     //Inclusão de um valor inteiro
4     int x;
5     printf ("\nDigite um valor inteiro : ");
6     scanf ("%d", &x);
7
8     //Inclusão de um valor float
9     float y;
10    printf ("\nDigite um valor real : ");
11    scanf ("%f", &y);
12
13    //Inclusão de um caractere
14    char c;
15    printf ("\nDigite um caractere: ");
16    fflush(stdin);
17    scanf ("%c", &c);
18
19    //Impressão dos valores digitados
20    printf ("\nValor inteiro digitado %d "
21            "\nValor real digitado %f "
22            "\nCaractere digitado %c ",
23            x, y, c);
24
25    return 0;
26 }
```



Comandos controladores de fluxo

A partir de agora falaremos sobre os comandos que permitem um controle do código, através de execução de blocos de comandos, desvios ou repetições.

ESTRUTURA DE DECISÃO

Nosso primeiro contato será com a estrutura de decisão, ela permite que um código específico seja executado, considerando a condição ela efetuará um desvio no código para atender esta condição.

Comando Se (if)

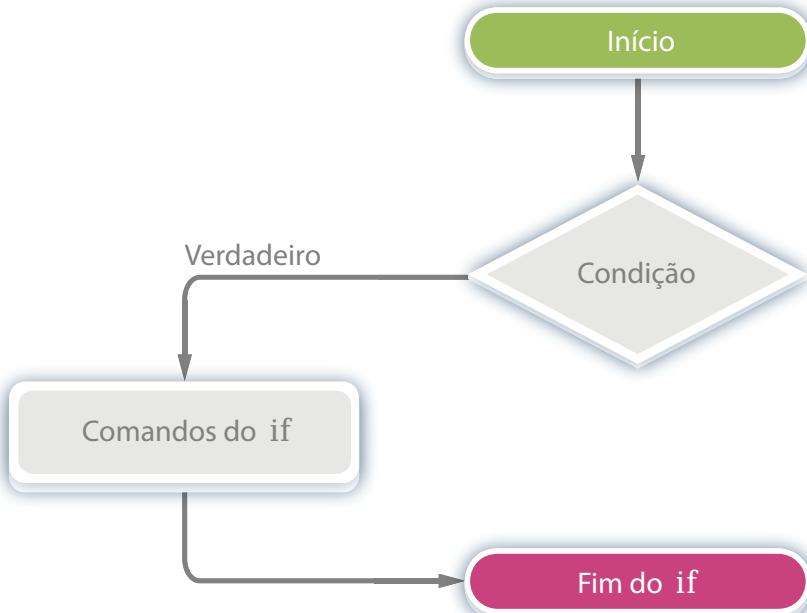
O comando mais utilizado para a estrutura de decisão é o **if**, este comando pode ser utilizado de forma simples, composta ou aninhadas.



Vamos exemplificar, e compreender, cada uma? Vem comigo!

ESTRUTURA DE DECISÃO SIMPLES

A forma de decisão simples efetua uma avaliação na condição, e, caso ela seja verdadeira, executa os comandos escritos no **if**.



Observe que, na forma simples, os comandos só serão executados se a condição imposta for verdadeira, não existem comandos a serem executados quando a condição for falsa.



No código, o comando dever ser utilizado da seguinte forma:

```
if (condição) {
```

Comandos executados, se a condição aplicada ao `if` for verdadeira.

```
}
```

ATENÇÃO

A condição imposta SEMPRE deverá ter seu resultado como verdadeiro (*true*) ou falso (*false*). Para escrevê-la pode-se utilizar operadores relacionais e/ou lógicos, e estas expressões também podem conter operadores aritméticos.

Quaisquer comandos poderão ficar subordinados ao `if`, mas lembre-se, eles só serão executados se a condição imposta for verdadeira.



Veja a seguir um exemplo de aplicação da estrutura de decisão simples:

```
1 #include <stdio.h>
2 int main () {
3     int numero;
4     printf ("\nDigite um número: ");
5     scanf ("%d", &numero);
6
7     if (numero % 2 == 0) {
8         printf ("\nNúmero par. ");
9     }
10    return 0;
11 }
```

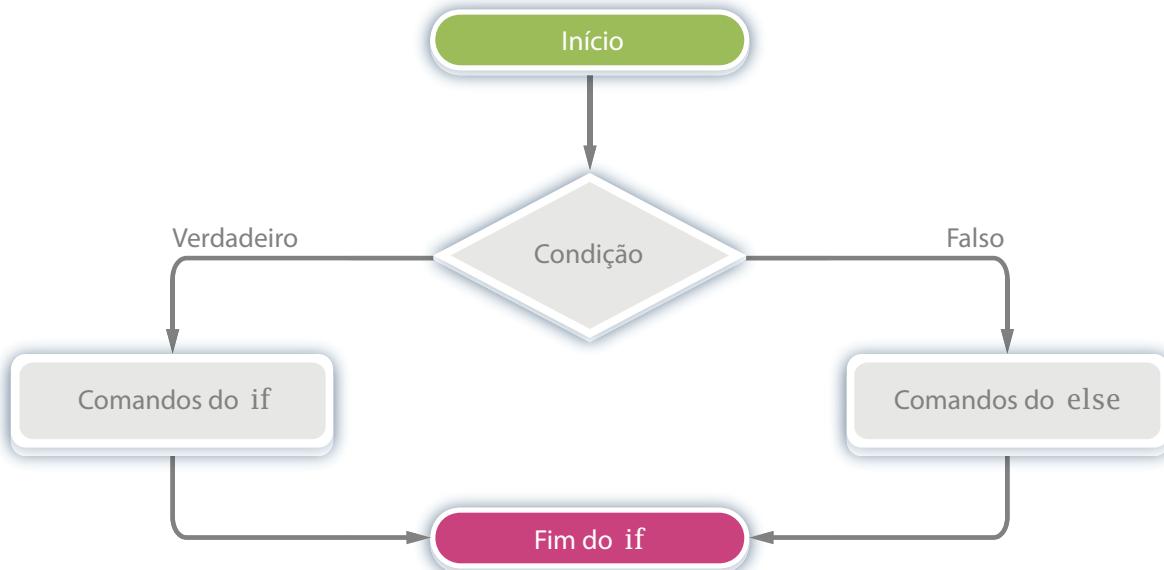
OBSERVAÇÃO

Observe no código que, na linha 7 do programa, o comando `if` avalia a expressão determinando se é falsa ou verdadeira, por isso, o comando de saída da linha 8 só será executado se a condição da linha anterior for verdadeira, neste caso, se número digitado for par.



ESTRUTURA DE DECISÃO COMPOSTA

A forma de decisão composta efetua uma avaliação na condição, e, caso ela seja verdadeira, executa os comandos escritos no `if`, caso seja falsa, executa os comandos o `else`.



Observe que esta é a forma completa do comando, ele está preparado para atender as duas respostas, verdadeiro (`if`) ou falso (`else`).

No código, o comando dever ser utilizado da seguinte forma?

```
if (condição) {  
    Comandos executados, se a condição aplicada ao if for verdadeira.  
}  
else {  
    Comandos executados, se a condição aplicada ao if for falsa.  
}
```



O comando será executado sob duas situações distintas!

A primeira situação ocorrerá quando a condição for verdadeira, e executará os comandos descritos sob o `if`.

A segunda situação ocorrerá quando a condição for falsa, e executará os comandos descritos sob o `else`.

O comando NUNCA executará as duas situações ao mesmo tempo.

Veja a seguir um exemplo de aplicação da estrutura de decisão composta:



```

1 #include <stdio.h>
2 int main () {
3     int numero;
4     printf ("\nDigite um número: ");
5     scanf ("%d", &numero);
6
7     if (numero % 2 == 0) {
8         printf ("\nNúmero par. ");
9     } else {
10        printf ("\nNúmero ímpar. ");
11    }
12
13    return 0;
14 }

```

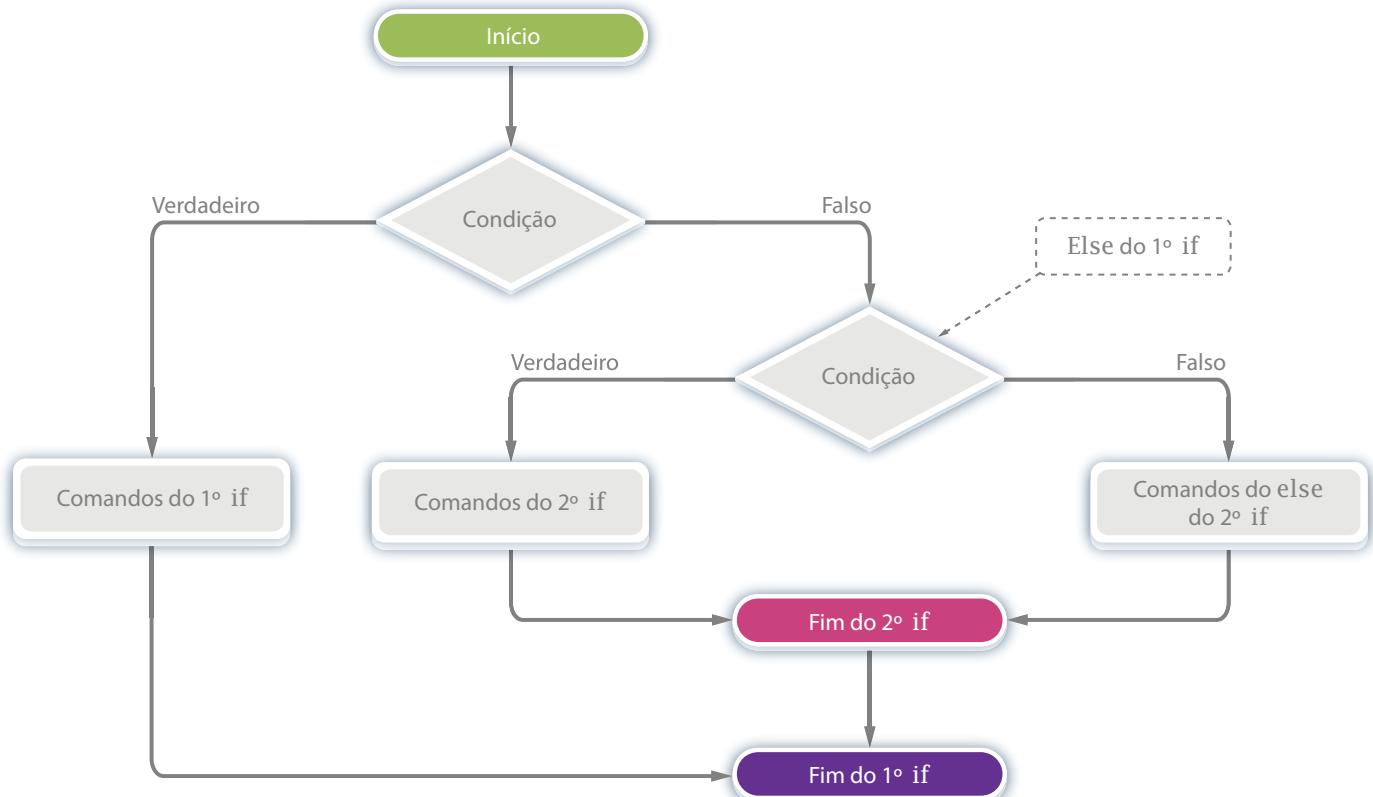
ATENÇÃO

No código anterior, a linha 10 só será executada se a condição, imposta na linha 7, for falsa, neste caso, se o número for ímpar.



ESTRUTURA DE DECISÃO ANINHADA

Sabendo que o comando if, completo, atende duas situações (verdadeiro ou falso), para aplicarmos mais condições será necessário abrir novos comandos if's.



IMPORTANTE

Observe que, aplicar `else` faz com que o comando seja mais bem aproveitado, uma vez que já sabemos que, se o comando não entrar no `if` testará o `else`, nunca executando ambos.



No código, o comando dever ser utilizado da seguinte forma:

```
if (condição) {
```

Comandos executados, se a condição aplicada ao `if` for verdadeira.

```
} else {(condição) {
```

Comandos executados, se a condição aplicada ao primeiro `if` for falsa. Porém, uma nova análise de condição será aplicada, por isso, esses comandos só serão executados se a condição do segundo `if` for verdadeira.

```
} else {
```

Comandos executados, se a condição do segundo `if` for falsa.

```
}
```

Você pode utilizar quantos `if`'s aninhados desejar, bastando respeitar a hierarquia de criação.

Veja a seguir um exemplo de aplicação da estrutura de decisão aninhada:

```
1 #include <stdio.h>
2 int main () {
3     int numero;
4     printf ("\nDigite um número: ");
5     scanf ("%d", &numero);
6
7     if (numero < 0) {
8         printf ("\nNúmero negativo. ");
9     } else if (numero > 0) {
10        printf ("\nNúmero positivo. ");
11    } else {
12        printf ("\nNúmero nulo. ");
13    }
14
15    return 0;
16 }
```

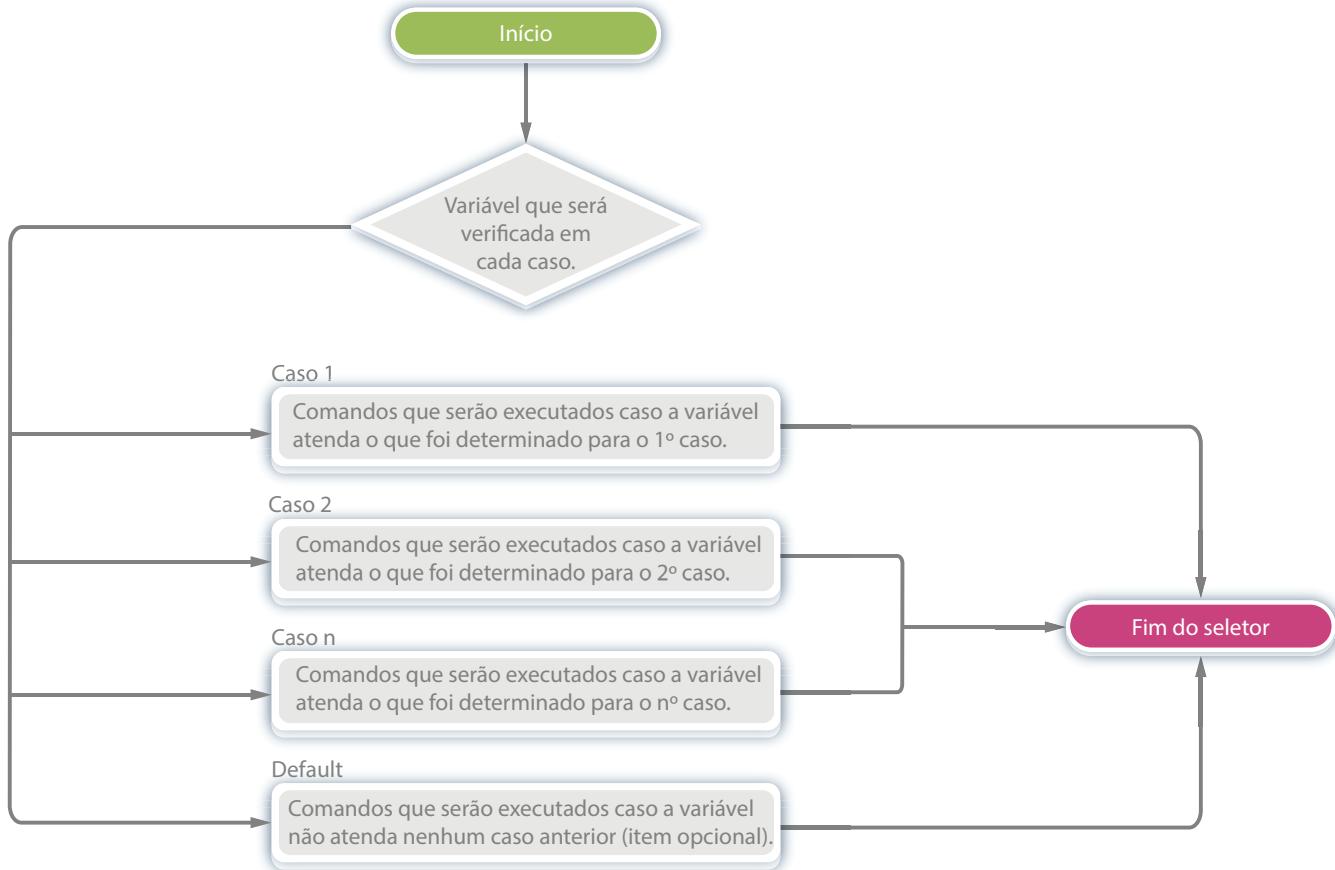
OBSERVAÇÃO

No código anterior, a linha 10 só será executada se a condição, imposta nas linhas 7 e 9, forem falsas.



Switch

Outro comando de decisão é o *switch*, ele permite que você tenha opções distintas, para determinados valores da variável avaliada.



Este comando trabalha de forma parecida com o comando *if*, ou seja, ele decidirá, a partir de uma condição, qual (ou quais) comando(s) será(ão) executado(s). Ele só atenderá valores fixos, sem possibilidade de expressões. Além disto, para a avaliação, a variável deverá ser inteira (*int*) ou caractere (*char*).

A sintaxe deste comando é:

```
switch (variável) {  
    case <constante1> :  
        <comandos1>;  
        break;  
    case <constante2> :  
        <comandos2>;  
        break;  
    ...  
    case <constanteN> :  
        <comandosN>;  
        break;  
    default :  
        <comandosDefault>;  
        break;  
}
```



IMPORTANTE



Algumas observações são importantes:

- 1º Após indicar os comandos, de cada caso, é imprescindível colocar o comando break, para que não teste todos os casos previstos posteriormente;
- 2º A opção default é opcional, e só servirá para as situações onde nenhum caso indicado anteriormente corresponder ao valor da variável testada;
- 3º A variável deverá atender os tipos citados anteriormente (inteira ou caractere), e seu valor será comparado a cada caso indicado;
- 4º Os comandos só serão executados se atender ao caso, ou se caírem na opção default

Mas atenção: ao utilizar um comando de decisão, seja ele o if ou switch, é importante se lembrar que uma vez avaliada a condição os comandos só serão executados uma vez, ou seja, decidiu, passa para o próximo comando do código. Esta NÃO é uma estrutura de repetição, APENAS de decisão.

```
1 #include <stdio.h>
2 int main () {
3     int dia;
4     printf ("\nDigite o número do dia: ");
5     scanf ("%d", &dia);
6
7     switch (dia) {
8         case 1:
9             printf ("\nDomingo ");
10            break;
11        case 2:
12            printf ("\nSegunda ");
13            break;
14        case 3:
15            printf ("\nTerça ");
16            break;
17        case 4:
18            printf ("\nQuarta ");
19            break;
20        case 5:
21            printf ("\nQuinta ");
22            break;
23        case 6:
24            printf ("\nSexta ");
25            break;
26        case 7:
27            printf ("\nSábado ");
28            break;
29        default :
30            printf ("\nValor inválido ");
31            break;
32    }
33    return 0;
34 }
```



ESTRUTURA DE REPETIÇÃO

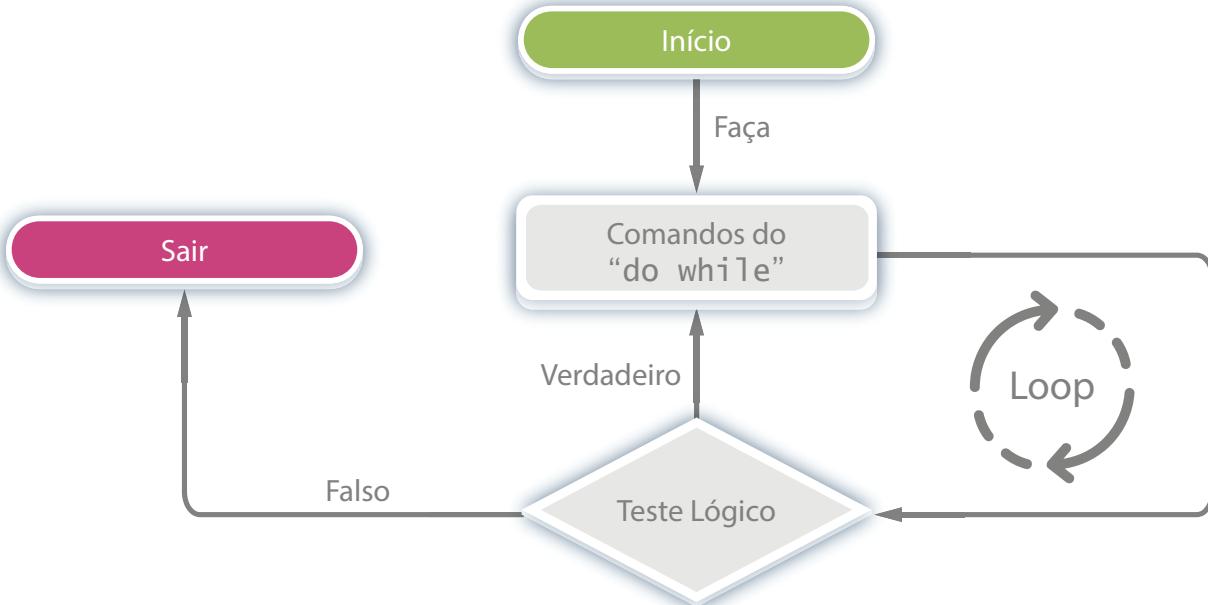
Diversas vezes é necessário escrever um código e repeti-lo para que execute mais de uma vez. Bem, para estes casos o ideal é que você escreva seu código uma única vez, confirme se ele está funcionando corretamente, e depois, coloque-o dentro de uma estrutura de repetição, afinal, elas foram criadas para repetir seu código, sem a necessidade de exigir que você o reescreva.

Os comandos de repetição exigem que você inclua uma condição de execução, assim, ele executará seus comandos enquanto a condição imposta for verdadeira, e finalizará a repetição no momento em que esta condição for falsa.

O comum é que se aplique uma condição para que os comandos efetuem a repetição, e, geralmente as linguagens possuem três tipos distintos de repetição: condicionando no início, no final ou como contador.

Repetição com condição no final - do while

O comando `do while` efetua a repetição do código e só depois testa a condição imposta, e, se esta condição for verdadeira, repete os comandos, caso contrário, sai da estrutura e continua no próximo comando após o final dela.



A sintaxe deste comando é:

```
do {  
    Comandos executados, enquanto a condição aplicada ao do while for verdadeira.  
} while (condição);
```

Todos os comandos serão executados uma vez, só depois da primeira execução eles serão testados, e, a partir daí os comandos serão repetidos enquanto a condição for verdadeira.

Por isso dizemos que esta estrutura executa seus comandos, uma ou mais vezes.



Veja a seguir uma aplicação do comando:

```
1 #include <stdio.h>
2 int main () {
3     int numero, i = 0;
4     do {
5         do {
6             printf ("\nExecução %d", i + 1);
7             printf ("\nDigite um número positivo: ");
8             scanf ("%d", &numero);
9         } while (numero <= 0);
10
11     if  (numero % 2 == 0) {
12         printf ("\nNúmero par. ");
13     } else {
14         printf ("\nNúmero ímpar. ");
15     }
16     i++;
17 } while (i < 5);
18 return 0;
19 }
```

REFLITA

Ao analisar o código, podemos perceber que existem duas estruturas de repetição do while.

Primeiramente vamos analisar a mais interna, iniciada na linha 5, para este caso, os comandos das linhas 6 a 8 serão executados enquanto o número digitado for menor ou igual a zero (condição da linha 9 – while).

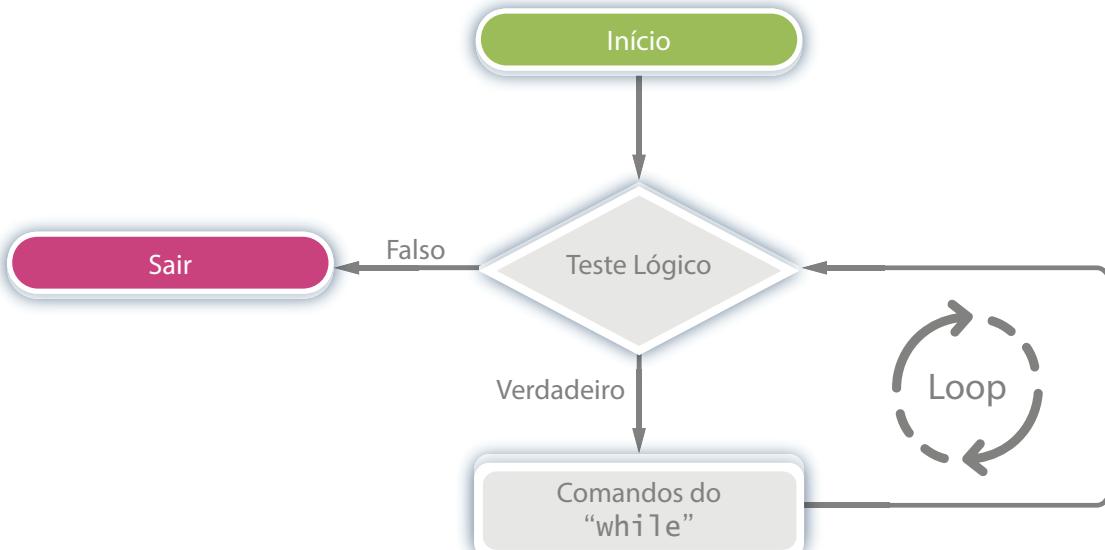
Agora analisaremos a mais externa, iniciada na linha 4, nela, os comandos das linhas 5 a 16 serão executados enquanto o valor da variável *i* for menor que cinco. A variável foi inicializada com 0 (zero) e a cada iteração será somado +1 para seu valor (linha 16), e, enquanto este valor for menor ou igual a cinco a condição continuará sendo verdadeira, repetindo assim os comandos das linhas 5 a 16.

E veja que, para cada iteração da estrutura de repetição mais externa, haverá uma, ou mais, iterações da estrutura mais interna.



Repetição com condição no início - while

O comando while testa a condição imposta, e, se esta condição for verdadeira, repete os comandos, caso contrário, sai da estrutura e continua no próximo comando após o final dela.



Sua sintaxe é :

```
while (condição) {
```

Comandos executados, enquanto a condição aplicada ao while for verdadeira.

```
}
```

A condição é testada antes da execução dos comandos, e estes serão repetidos enquanto a condição for verdadeira. Por isso dizemos que esta estrutura executa seus comandos, zero ou mais vezes.

Veja a seguir uma aplicação do comando:

```
1 #include <stdio.h>
2 int main () {
3     int numero, i = 0;
4     while (i < 5){
5         numero = 0;
6         while (numero <= 0){
7             printf ("\nExecução %d", i + 1);
8             printf ("\nDigite um número positivo: ");
9             scanf ("%d", &numero);
10        }
11
12        if (numero % 2 == 0) {
13            printf ("\nNúmero par. ");
14        } else {
15            printf ("\nNúmero ímpar. ");
16        }
17        i++;
18    }
19    return 0;
20 }
```

Ao analisar o código, podemos perceber que existem duas estruturas de repetição while.

Primeiramente vamos analisar a mais interna, iniciada na linha 6, para este caso, os comandos das linhas 7 a 9 serão executados enquanto o número digitado for menor ou igual a zero. Veja que, na linha 5, a variável numero foi inicializada com o valor 0 (zero), assim, quando executar a condição do while o resultado será verdadeiro. Isso só é necessário para que o código entre na estrutura while, uma vez lá dentro, a verificação acontecerá considerando o que for digitado pelo usuário, neste caso, enquanto ele digitar um número menor ou igual a zero o código continuará repetindo e solicitando nova digitação.

Agora analisaremos a mais externa, iniciada na linha 4, nela, os comandos das linhas 5 a 17 serão executados enquanto o valor da variável i for menor que cinco. A variável foi inicializada com 0 (zero) para ser testada no início da estrutura while. A cada iteração será somado + 1 para seu valor (linha 17), e, enquanto este valor for menor ou igual a cinco a condição continuará sendo verdadeira, repetindo assim os comandos das linhas 5 a 17.

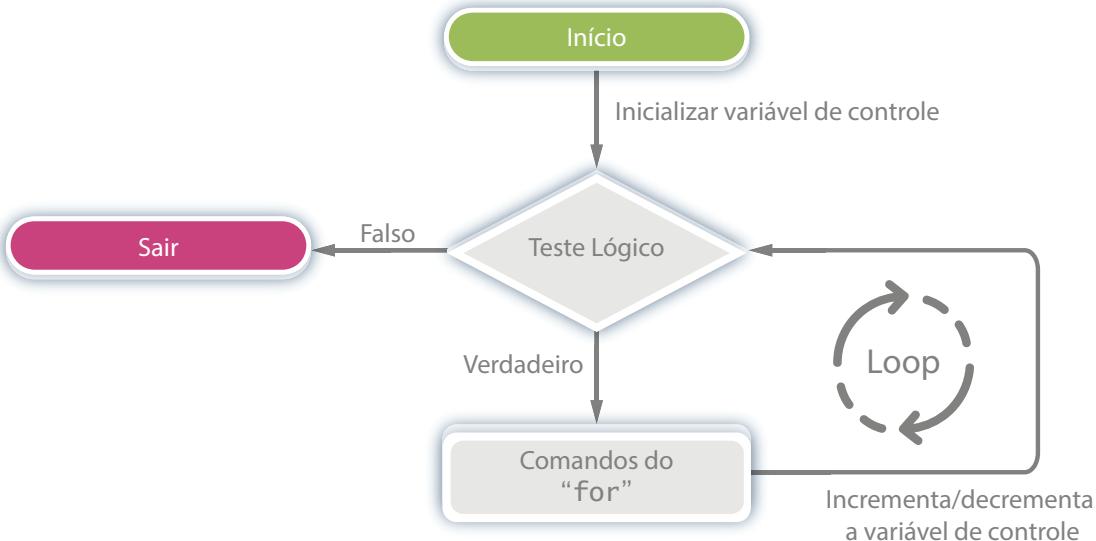
E veja que, para cada iteração da estrutura de repetição mais externa, haverá uma, ou mais, iterações da estrutura mais interna, pois esta está contida naquela.



Repetição com contador de iterações - for

A estrutura do comando for foi criada para efetuar o controle das iterações necessárias ao código, ao aplicá-lo, será necessário inicializar a variável, indicar a condição de repetição, e, por fim, o incremento, ou decremento, da variável que controla das iterações.

Ao iniciar a estrutura, a variável será inicializada e a condição testada, se for verdadeira, os comandos serão executados. A partir da segunda iteração, a variável sofrerá o incremento (ou decremento) e novamente a condição será testada, se for verdadeira executará novamente os comandos, e repetirá este processo até que a condição seja falsa.



Sua sintaxe é:

```
for (inicialização; condição; incremento/decremento) {
```

Comandos executados, enquanto a condição aplicada ao comando "for" for verdadeira.

```
}
```

Veja a seguir uma aplicação do comando

```
1 #include <stdio.h>
2 int main (){
3     int numero, i;
4     for (i = 0; i < 5; i++){
5         printf ("\nExecução %d", i + 1);
6         printf ("\nDigite um número positivo: ");
7         scanf ("%d", &numero);
8         while (numero <= 0){
9             printf ("\nValor inválido. Digite novamente. ");
10            printf ("\nDigite um número positivo: ");
11            scanf ("%d", &numero);
12        }
13
14        if (numero % 2 == 0) {
15            printf ("\nNúmero par. ");
16        } else {
17            printf ("\nNúmero ímpar. ");
18        }
19    }
20    return 0;
21 }
```



Agora vamos analisar o código, o comando `for` começa na linha 4, na primeira execução a variável `i` será inicializada com o valor 0 (zero) e a condição testada. Considerando que 0 (zero) é menor que 5 (cinco) os comandos da linha 5 até a 18 serão executados.

Na segunda execução, a variável `i` será incrementada com 1 (`i++`), a condição novamente testada e, se verdadeira, os comandos das linhas 5 a 18 serão executados. Este processo ocorrerá até que a condição seja falsa.

Comando auxiliar – `break`

O comando `break` é utilizado para finalizar uma estrutura de repetição, sem que a sua condição seja analisada, ou para finalizar uma condição do comando `switch`.

Em estruturas de repetição, a sua utilização finaliza o loop mesmo que a condição não seja atendida.

```
1 #include <stdio.h>
2 int main () {
3     int numero, i;
4     for (i = 0; i < 5; i++) {
5         printf ("\nExecução %d", i + 1);
6         printf ("\nPara finalizar o programa digite um número negativo ");
7         printf ("\nDigite um número positivo : ");
8         scanf ("%d", &numero);
9
10    if (numero < 0) {
11        break;
12    } else if (numero % 2 == 0) {
13        printf ("\nNúmero par. ");
14    } else {
15        printf ("\nNúmero ímpar. ");
16    }
17 }
18 return 0;
19 }
```

Na linha 5 do código existe uma estrutura de decisão que condiciona se o número digitado é menor que 0 (negativo), se for, o comando `break` será executado, e neste caso, a estrutura de repetição será finalizada e o programa continua a partir da linha 17, ignorando todos os outros comandos, mesmo que a condição da linha 4 não seja atendida (verdadeira).

Este é um comando muito utilizado com loop's escritos de forma eterna (que não possuem fim, e que nunca atenderão a condição imposta).

Veja abaixo dois exemplos da utilização do comando `break` com o comando `for`, mas lembre-se que este comando finaliza qualquer estrutura de repetição, portanto, também podemos utilizar com o `while` e do `while`.

```
1 #include <stdio.h>
2 int main () {
3     int numero, i;
4     for (i = 0; ; i++) {
5         printf ("\nExecução %d", i + 1);
6         printf ("\nPara finalizar o programa digite um número negativo ");
7         printf ("\nDigite um número positivo: ");
8         scanf ("%d", &numero);
9
10    if (numero < 0) {
11        break;
12    } else if (numero % 2 == 0) {
13        printf ("\nNúmero par. ");
14    } else {
15        printf ("\nNúmero ímpar. ");
16    }
17 }
18 return 0;
19 }
```



No exemplo anterior, o comando `for` é utilizado sem condição de parada, observe que, na linha 4 do código, não existe condição determinada, portanto, a única forma de finalizar a repetição é utilizando o comando `break` (da linha 10).

Apesar de não ter condição de finalização, é possível saber quantas iterações foram realizadas, afinal, a variável `i` ainda é incrementada no comando.

Vamos ver outro exemplo para utilização deste comando:

```
1 #include <stdio.h>
2 int main () {
3     int numero;
4     for ( ; ; ) {
5         printf ("\nPara finalizar o programa digite um número negativo ");
6         printf ("\nDigite um número positivo: ");
7         scanf ("%d", &numero);
8
9         if (numero < 0) {
10             break;
11         } else if (numero % 2 == 0) {
12             printf ("\nNúmero par. ");
13         } else {
14             printf ("\nNúmero ímpar. ");
15         }
16     }
17     return 0;
18 }
```

No exemplo, o comando `for` é utilizado sem inicialização, sem condição de parada e sem incremento/decremento. Observe que, na linha 4 do código, não existe nada determinado, portanto, a única forma de finalizar a repetição é utilizando o comando `break` (da linha 10), e, como não temos variável de controle, através do comando, não será possível saber quantas iterações foram executadas.



Síntese

Neste módulo falamos sobre os comandos que nos permitem controlar o código.

Começamos pelos comandos básicos, aqueles que fazem comunicação direta com o usuário do seu programa. Você descobriu que o comando de saída permite que o seu programa converse com o usuário, e aqui utilizaremos o comando printf. Descobriu também que o comando de entrada é aquele onde o seu usuário conversa com o seu programa, e aqui utilizaremos o comando scanf.

Para controlar seu código, com o intuito de realizar decisões e repetições necessárias ao bom funcionamento, aqui falamos sobre os controladores de fluxos.

O primeiro controlador foi a estrutura de decisão, que servem para efetuar um desvio no programa, executando apenas os comandos que atendem a uma determinada condição, se não atender, tais comandos não serão executados. Conhecemos o comando if em sua versão simples, composta ou aninhada, este é o comando mais completo quando necessitamos de decisão, e, por fim, finalizamos com o comando switch, que nos permite executar casos específicos dependendo dos valores das variáveis.

Falamos também sobre as estruturas de repetição, elas nos permitem executar um ou mais comandos diversas vezes, dependendo da condição inserida.

Neste caso conhecemos as três possíveis para utilização: do while, que executa o bloco de comandos uma vez e depois condiciona, se esta condição for verdadeira repete o bloco, caso contrário finaliza o comando; while, que primeira testa a condição, se esta condição for verdadeira executa o bloco de comandos, caso contrário finaliza o comando; e, por último, o comando for, que inicializa uma variável e executa a condição, se esta condição for verdadeira executa o bloco de comandos, incrementa/decrementa a variável para um novo teste, caso contrário finaliza o comando.

Para finalizar, o comando break, sendo este utilizado para finalizar qualquer estrutura de repetição, continuando a execução a partir da próxima linha disponível.

Espero que este conteúdo tenha sido proveitoso, agora já estamos prontos para nossos códigos.

Até mais!

Referências

CELES FILHO, W. **Introdução a estrutura de dados:** com técnicas de programação em c. Rio de Janeiro: Elsevier, 2004. 294 p.

DAMAS, Luis. **Linguagem C.** 10. ed. Rio de Janeiro: Ltc, 2016.

MANZANO, José Augusto N G. **Linguagem C:** Acompanhada de uma xícara de café. São Paulo: Érica, 2015.

MIZRAHI, Victorine Viviane. **Treinamento em linguagem C.** São Paulo: Pearson Prentice Hall, 2008. 407 p.