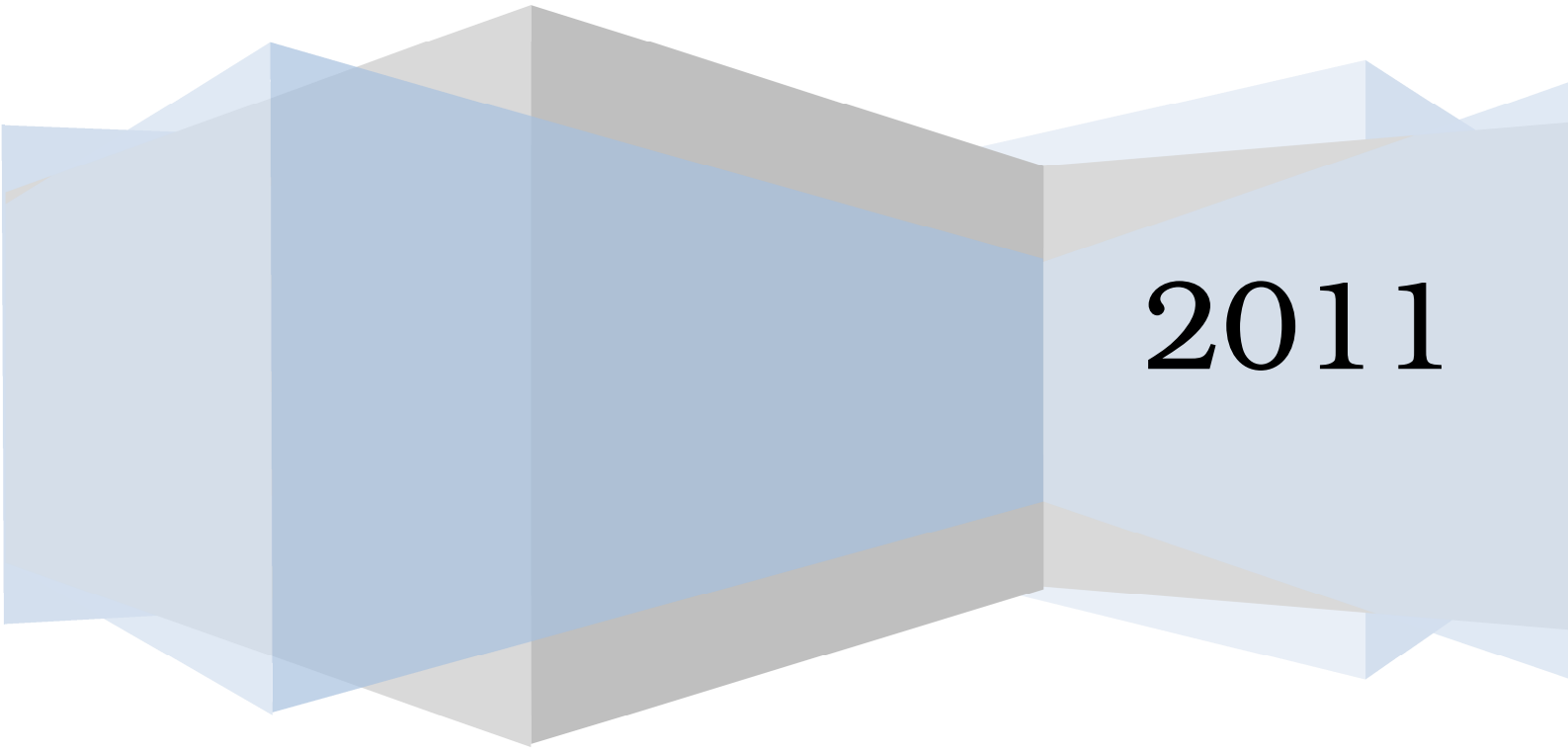
Flávio Augusto de Freitas
Introdução à Programação em Linguagem C/C++

http://flavioaf.blogspot.com

C/C++

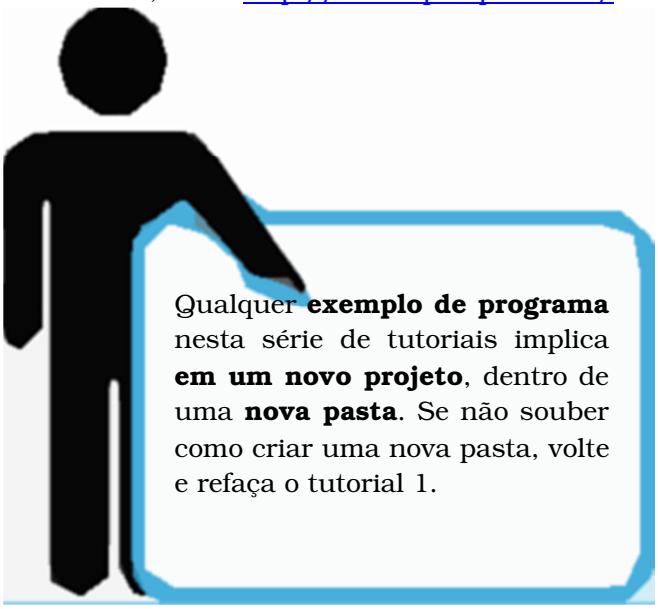
Tutorial 6 (usando Dev-C++ versão 4.9.9.2)



2011

1 Introdução

Esta série de tutoriais foi escrita usando o **Microsoft Windows 7 Ultimate** e o **Bloodshed Dev-C++** versão 4.9.9.2, que pode ser baixada em http://www.bloodshed.net. Se alguém quiser adquirir mais conhecimentos e quiser aprofundar no assunto, visite http://www.cplusplus.com/.



2 Controle de fluxo

Dificilmente um programa em C irá executar sempre as mesmas instruções, na mesma ordem, independentemente do que tenha acontecido anteriormente ou do valor que foi fornecido. É muito comum que alguém queira que um pedaço de código só seja executado se certa condição for verdadeira; também é comum querer que um pedaço de código seja repetido várias vezes, de tal maneira que simplesmente copiar o código não resolveria o problema ou seria trabalhoso demais. Para casos como esses, existem as estruturas de controle de fluxo.

Em C, existem várias instruções relacionadas ao controle de fluxo:

if, que executa um bloco apenas se uma condição for verdadeira;

```
int x = 1;
if(x == 1) printf("Isto sera impresso!\n");
```

switch, que executa um bloco de acordo com o valor de uma expressão ou variável;

```
int x = 1;
switch(x) {
  case 1: printf("Um\n"); break;
  case 1: printf("Dois\n"); break;
  default: printf("Nem 1, nem 2\n"); break;
}
```

for, que executa um bloco repetidas vezes enquanto uma condição for verdadeira, executando uma instrução (geralmente de incremento ou decremento de uma variável) após cada execução;

```
for(int x=1; x<5;x++) printf("%d\n",x);
```

while, que executa um bloco enquanto uma condição for verdadeira;

```
while(atoi( getchar() ) != 27)
  printf("Pressione Esc para sair...\n");
```

do, semelhante ao while, mas a condição é avaliada após a execução (e não antes);

```
do {
  printf("Pressione Esc para sair...\n");
} while( atoi( getchar() ) != 27 );
```

goto, que simplesmente pula para um lugar pré-definido.

Porém, antes de entrar no estudo dessas estruturas, você deve saber como escrever uma condição. É o que explicamos a seguir.

3 Expressões de condição

Uma expressão de condição é uma expressão normal em C que, quando avaliada, será interpretada como verdadeira ou falsa. Em C, na verdade, esse valor é um valor inteiro que sendo 0 (zero) significa falso, sendo qualquer outro número significa verdadeiro.

Geralmente em expressões condicionais usamos os operadores relacionais, ou seja, que avaliam a relação entre seus dois operandos. Existem seis deles:

Operador	**Significado**
>	maior que
>=	maior ou igual a
<	menor que
<=	menor ou igual a
==	igual a
!=	diferente de

Todos esses operadores são binários, ou seja, trabalham com dois valores ou operandos. Esses operadores sempre comparam o valor da esquerda com o da direita, ou seja, a expressão `a > b` significa "a é maior que b".

Note que para saber se dois números são iguais devemos usar dois sinais de igual. Um erro muito comum é esquecer-se de um deles,

transformando a comparação numa atribuição, por exemplo:

```
if(x = 1)
    ...
```

O que acontece aqui é que a variável x recebe o valor 1, de modo que a expressão entre parênteses também terá o valor 1 — tornando a "condição" sempre verdadeira. Similarmente, se usássemos o número zero, a expressão sempre seria falsa. Portanto, sempre tome cuidado com esse tipo de comparação. A maneira certa de comparar com um número é:

```
if(x == 1)
    ...
```

Também é comum que combinemos condições. Por exemplo, podemos querer que um número seja menor que 10 ou maior que 50. Como o operador "ou" é "||", escreveríamos: `n < 10 || n > 50`.

A seguir você vê os operadores lógicos:

Operador	**Significado**
||	ou (OR)
&&	e (AND)
!	não (NOT)

Algumas explicações sobre os operadores lógicos:

O operador "não" é unário, ou seja, é uma operação que envolve apenas um valor. O que ele faz é inverter o valor de seu operando: retorna falso se a expressão for verdadeira e vice-versa. Deve-se usar parênteses ao negar uma expressão: `!(x > 6)`, por exemplo.

O operador "ou" retorna "verdadeiro" se pelo menos um dos operandos for verdadeiro; retorna "falso" apenas se ambos forem falsos.

O operador "e" retorna "verdadeiro" apenas se ambos os seus operandos forem verdadeiros.

Observação Se você quer saber se um número está entre outros dois, a sintaxe matemática (10 < n < 50) não funcionará. Se você usar esse código, na verdade primeiramente será avaliada a expressão 10 < n, que poderá resultar em 0 ou 1. Portanto, a expressão equivale a (0 ou 1) < 50, o que é sempre verdadeiro.

A comparação correta envolveria o operador "e" (&&): `10 < n && n < 50`.

Pelo fato de todo valor diferente de zero ser avaliado como verdadeiro e zero como falso, existem as seguintes equivalências (apenas quando estas expressões são usadas como condições):

```
(x == 0) equivale a (!x)
(x != 0) equivale a (x)
```

4 Testes

Testes são estruturas de controle que executam certos blocos de código apenas se uma certa condição for verdadeira. Existem três estruturas desse tipo em C:

4.1 if

O teste if avalia uma condição e, se ela for verdadeira, executa um bloco de código. A sintaxe correspondente a isso é:

```
if(condição) {
    ... /* bloco de comandos */
}
```

Mas também podemos especificar um bloco a ser executado caso a condição for falsa. Nesse caso, escrevemos:

```
if(condição) {
    ... /* comandos se verdadeiro */
}
else {
    ... /* comandos se falso */
}
```

As chaves podem ser omitidas caso haja apenas uma instrução no bloco. Por exemplo:

```
if(x == 5)
    printf("x é igual a 5.\n");
```

Perceba que, se esquecermos das chaves, o compilador não deverá dar nenhum erro; no entanto, tudo que exceder a primeira instrução será executado incondicionalmente, mesmo que esteja na mesma linha! No exemplo a seguir, a frase "x é igual a 5" seria exibida mesmo que o número não fosse 5!

```
if(x == 5)
    j++;
printf("x é igual a 5.\n");
```

Podemos avaliar diversas condições com os testes if, bastando para isso colocar um novo teste no bloco else. Também é possível aninhar blocos if, ou seja, colocar um dentro de outro:

```
if(x > 9) {
   printf ("x > 9.\n");
}
else
if(x >= 5) {
   printf("x >= 5, mas não > 9.\n");
}
else {
   if(x == 0) {
      printf("x = zero.\n");
   }
   else {
      printf("x é não-nulo e < 5.\n");
   }
}
```

4.2 SWITCH

O teste switch compara uma expressão com diversos valores que podem estar associados a blocos de códigos diferentes, e executa o bloco de código correspondente ao valor encontrado. Você também pode especificar um bloco que deve ser executado caso nenhum dos outros valores seja encontrado: é o bloco default ("padrão" em inglês).

```
switch(expressão) {
   case valor1:
      instruções;
      break;
   case valor2:
      instruções;
      break;
   ...
   default:
      instruções;
}
```

Note que no teste switch não precisamos usar chaves em volta dos blocos, a menos que declaremos variáveis neles. Um exemplo da utilização de switch seria a criação de um menu:

```
int opcao;
printf("[1] Cadastrar cliente\n"
       "[2] Procurar cliente\n"
       "[3] Inserir pedido\n"
       "[0] Sair\n\n"
       "Digite sua escolha: ");
scanf("%d", &opcao);

switch(opcao) {
   case 1:
      cadastra_cliente();
      break;
   case 2:
      procura_cliente();
      break;
   case 3:
      insere_pedido();
      break;
   case 0:
      return 0;
   default:
      printf ("Opção inválida!\n");
}
```

A instrução break indica que o programa continua a execução após o final do bloco switch (pulando o que estiver no meio). Se ela não fosse usada, para certo valor encontrado, seriam executadas também as instruções de todos os valores abaixo dele. Em alguns casos, podemos omitir intencionalmente a instrução break. Por exemplo, no exemplo acima, não colocamos uma instrução break para o valor zero, pois quando retornamos de uma função (return 0) o bloco switch já é abandonado.

Também podemos querer que uma instrução seja executada para mais de um valor. Vamos supor que no nosso menu as duas primeiras opções fossem "Cadastrar pessoa física" e "Cadastrar pessoa jurídica", e tivéssemos uma função que faz o cadastro diferentemente dependendo do valor da variável pessoa_fisica. Poderíamos fazer um código assim:

```
switch(opcao) {
   case 1: /* pessoa física */
      pessoa_fisica = 1;
   case 2:
      cadastra();
      break;
   ...
}
```

Nesse caso, para qualquer uma das duas opções seria executada a função cadastra, mas se selecionarmos "pessoa física" a variável será atribuída antes.

4.3 OPERADOR TERNÁRIO " ? : "

O operador ternário ?: é uma alternativa abreviada da estrutura if/else. Ele avalia uma expressão e retorna certo valor se ela for verdadeira, ou outro valor se ela for falsa. Sua sintaxe é:

condição ? valorSeVerdadeira : valorSeFalsa

Note que, ao contrário de if, ao usarmos o operador condicional ?: precisamos sempre prover tanto o valor para o caso de a condição ser falsa quanto o valor para o caso de ela ser verdadeira.

O operador condicional pode ser usado em situações como essa:

```
int horaAbertura = (diaSemana == DOM) ? 11 : 9;
printf ("Abrimos às %d horas", horaAbertura);
```

Ou seja, se o dia da semana for domingo, a variável horaAbertura será definida para 11; caso contrário, será definida para 9.

Outro exemplo:

```
if(numMensagens > 0) {
   printf ("Você tem %d mensage%s",
     numMensagens,
     (numMensagens > 1) ? "ns" : "m");
}
```

Neste caso, o programa utilizaria "mensagens" caso houvesse mais de uma mensagem, e "mensagem" caso houvesse apenas uma mensagem.

4.4 Loops

Loops são conjuntos de instruções que devem ser executadas repetidas vezes, enquanto uma condição for verdadeira. Em C há 3 tipos de loops: while, do ... while e for.

4.4.1 while

O loop while testa uma condição; se ela for verdadeira, o bloco correspondente é executado e o teste é repetido. Se for falsa, a execução continua logo após o bloco. A sintaxe de while é:

```
while(condição) {
   ...
}
```

Por exemplo:

```
while(a < b)
{
   printf ("%d é menor que %d", a, b);
   a++;
}
```

Este código seria executado até que a fosse igual a b; se a fosse igual ou maior que b, nada seria executado. Por exemplo, para b = 10 e a < 10, a última mensagem que o usuário veria é "9 é menor que 10".

Repare que o loop while é como fosse um if, ou seja, o bloco é executado se a condição for verdadeira. A diferença é que ao final da execução, o while é executado novamente, mas o if não. No loop while (assim como nos loops do e for) também podemos usar a sintaxe abreviada para apenas uma instrução:

```
while(a < b)
   a++;
```

Exemplo

Um aparelho de CD foi adquirido por R$ 300,00 e revendido por R$ 240,00. Qual foi a porcentagem de lucro na transação?

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
  float v_ant = 300.0; // valor anterior
  float v_nov = 340.0; // valor novo
  float p_lucro = 0.0; // porcentagem de lucro

  while(v_ant + ((p_lucro/100)*v_ant) < v_nov) {
    p_lucro += 0.1;
  }

  printf("% de lucro = %f%%\n", p_lucro);
  // 13.39

  system("PAUSE"); // pausa o programa
  return EXIT_SUCCESS;
}
```

4.4.2 Loops infinitos

Você pode fazer loops infinitos com while, usando uma condição que é sempre verdadeira, como "1 == 1" ou simplesmente "1" (que, como qualquer valor não-nulo, é considerado "verdadeiro"):

```
while(1) {
   ...
}
```

Você pode sair de um loop — infinito ou não — com a instrução break, que você já viu no teste switch e será explicada mais abaixo.

4.4.3 do ... while

O loop "do ... while" é exatamente igual ao "while" exceto por um aspecto: a condição é testada depois do bloco, o que significa que o bloco é

executado pelo menos uma vez. A estrutura do ... while executa o bloco, testa a condição e, se esta for verdadeira, volta para o bloco de código. Sua sintaxe é:

```
do {
   ...
} while(condição);
```

Note que, ao contrário das outras estruturas de controle, é necessário colocar um ponto-e-vírgula após a condição.

```
do {
   printf("%d\n", a);
   a++;
} while (a < b);
```

Um exemplo de utilização de do ... while é em um menu. Pediríamos que o usuário escolhesse uma opção até que ele escolhesse uma opção válida:

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
  int i;

  do {
    printf("Escolha uma fruta:\n\n");
    printf("\t(1) Mamão\n");
    printf("\t(2) Abacaxi\n");
    printf("\t(3) Laranja\n\n");
    scanf("%d", &i);
  } while(i < 1 || i > 3);

  switch(i) {
    case 1:
      printf("Escolheu mamão.\n");
      break;
    case 2:
      printf("Escolheu abacaxi.\n");
      break;
    case 3:
      printf("Escolheu laranja.\n");
      break;
  }

  system("PAUSE");
  return EXIT_SUCCESS;
}
```

4.4.4 FOR

O loop for é nada mais que uma abreviação do loop while, que permite que alguma inicialização seja feita antes do loop e que um incremento (ou alguma outra ação) seja feita após cada execução sem incluir o código dentro do bloco. A sua forma geral é

```
for(inicialização; condição; incremento) {
   instruções;
}
```

E equivale a

```
inicialização;
while(condição) {
   instruções; incremento;
}
```

Um exemplo do uso de for:

```
for(a = 1; a < 10; a++) {
   ...
}
```

Nesse exemplo, primeiro definimos o valor de a como 1; depois, o código (...) é repetido enquanto a for menor que dez, incrementando em uma unidade o valor de a após cada execução do código. Analisando essas condições, você podera perceber que o código será executado nove vezes: na primeira execução, temos a = 1; após a nona execução, a é igual a 10 e, portanto o bloco não será mais repetido.

Também podemos dar mais de uma instrução de inicialização ou de incremento (separadas por vírgula), além de poder usar naturalmente condições compostas com o uso dos operadores lógicos:

```
for(a = 1, b = 1;
    a < 10 && (b/a) < 20; a++, b *= 2) {
   ...
}
```

Nesse exemplo, "a" e "b" são inicializados com o valor 1. A cada loop, o valor de "a" é incrementado em uma unidade e o de "b" é dobrado. Isso ocorre enquanto "a" for menor que 10 e a razão entre "b" e "a" for menor que 20. Se você construir uma tabela com os valores de cada variável a cada loop (ou colocar algum contador dentro do loop), verá que ocorrem sete execuções.

Assim como while, o loop for testa a condição; se a condição for verdadeira ele executa o bloco, faz o incremento e volta a testar a condição. Ele repete essas operações até que a condição seja falsa.

Podemos omitir qualquer um dos elementos do for se desejarmos. Se omitirmos a inicialização e

o incremento, o comportamento será exatamente igual ao de while. Se omitirmos a condição, ficaremos com um loop infinito:

```
for(inicialização; ; incremento) {
   ...
}
```

Podemos também omitir o bloco de código, se nos interessar apenas fazer incrementos ou se quisermos esperar por alguma situação que é estabelecida por uma função externa; nesse caso, usamos o ponto-e-vírgula após os parênteses de for. Isso também é valido para o loop while:

```
for(inicialização; condição; incremento) ;

while(condição) ;
```

Por exemplo, suponha que temos uma biblioteca gráfica que tem uma função chamada graphicsReady(), que indica se podemos executar operações gráficas. Este código executaria a função repetidas vezes até que ela retornasse "verdadeiro" e então pudéssemos continuar com o programa:

```
while(!graphicsReady()) ;
```

4.4.5 BREAK E CONTINUE

Você já viu break sendo usado para sair do teste switch; no entanto, ele funciona também nos loops — while, do e for. Nos três casos, ele sai do último loop iniciado (mesmo que haja mais de um). Por exemplo:

```
while(1) {
   if(a > b)
      break;
   a++;
}
```

break sempre faz com que a execução do programa continue na primeira instrução seguinte ao loop ou bloco.

A instrução continue é parecida com break, porém ao executá-la saltamos para a próxima iteração loop ao invés de terminá-lo. Usar continue equivale a chegar ao final do bloco; os incrementos são realizados (se estivermos em um loop for) e a condição é reavaliada (qualquer que seja o loop atual).

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
  int opcao = 0;
  while(opcao != 5) {
    printf("Escolha opção de 1 a 5: ");
    scanf("%d", &opcao);

    // se opção inválida, reinicia loop
    if(opcao > 5 || opcao < 1) continue;
    switch(opcao) {
      case 1:
        printf("\n --> Primeira opcao...");
        break;
      case 2:
        printf("\n --> Segunda opcao...");
        break;
      case 3:
        printf("\n --> Terceira opcao...");
        break;
      case 4:
        printf("\n --> Quarta opcao...");
        break;
      case 5:
        printf("\n --> Abandonando...");
        break;
    } // fim do switch
  } // fim do while

  system("PAUSE");
  return EXIT_SUCCESS;
}
```

Esse exemplo recebe uma opção do usuário. Se ele digitar uma opção inválida (ou seja, não for um número de 1 a 5), a instrução continue voltará ao começo do loop e o programa pedirá novamente a entrada do usuário. Se ele digitar uma opção válida, o programa seguirá normalmente.

4.4.6 LOOPS SEM CONTEÚDO

Em alguns casos pode ser interessante o uso de loops sem conteúdo — ou seja, sem instruções dentro de seu corpo.

4.4.7 SALTOS INCONDICIONAIS: GOTO

O goto é uma instrução que salta incondicionalmente para um local específico no programa. Esse local é identificado por um rótulo. A sintaxe da instrução goto é:

```
goto nome_do_rótulo;
```

Os nomes de rótulo são identificadores sufixados por dois-pontos (:), no começo de uma linha (podendo ser precedidos por espaços). Por exemplo:

```
nome_do_rótulo:
...

goto nome_do_rótulo;
```

Muitos programadores evitam usar o goto, pois a maioria dos saltos pode ser feita de maneira mais clara com outras estruturas da linguagem C. Na maioria das aplicações usuais, pode-se substituir o goto por testes, loops e chamadas de funções.

5 Exercícios Propostos

a) Crie um programa que exiba a tabela verdade da função **e**.
b) Crie um programa que exiba a tabela verdade da função **ou**.
c) Crie um programa que exiba a tabela verdade da função **não**.
d) Faça um programa que exiba a tabela verdade da função S = A **e** B **ou não** C.
e) Faça um programa que exiba a tabela verdade da função S = (A **e** B) **ou não** (C **e não** D).
f) Crie um programa que execute um loop infinito usando **for** e só saia dele se o usuário informar um valor **falso**.
g) Faça um programa que leia dois números e use **switch** para escolher entre as operações soma, subtração, multiplicação, divisão e potenciação e calcule o resultado da operação escolhida.

6 Terminamos

Terminamos por aqui. Saia do Dev-C++ e corra para o próximo tutorial.