

Controle de fluxo em C



Decisões

comando if

```
if (expression)
    command ;
```

ou

```
if (expression)
{
    command ;
    command ;
    ...
}
```



O comando “if” pode testar qualquer expressão cujo resultado possa ser avaliada como número, sendo que 0 (zero) é considerado falso e não-zero é verdadeiro.

Ifs aninhados:

```
if (expression1)
    if (expression2)
    {
        command ;
        command ;
        ...
    }
```

Comando if-else

```
if (expression)
    command1 ; // <--- there is a semicolon here!
else
    command2 ;
```

ou

```
if (expression)
{
    command1a ;
    command1b ;
    command1c ;
}
else
{
    command2a ;
    command2b ;
}
```

If-else aninhados:

```
if (expression1)
    if (expression2)
        command1 ;
    else
        command2 ;
else
    command3 ;
```



A cláusula `else` sempre se combina com o último `if` em aberto; caso se deseje forçar outra combinação, deve-se usar chaves (`{...}`) para organizar a abertura/fechamento dos comandos "if".

Comando switch

```
switch (expression)
{
    case constant1:
        command1 ;
        ...
        break ;
    case constant2:
        command2 ;
        ...
        break ;
    case constant3:
    case constant4:
        command34a ;
        ...
        break ;
    ...
    default:
        commandNa;
        ...
        break ;
}
```

Exemplo (contagem de carros):

```
ch = getchar();

switch(ch)
{
  case 'C':
  case 'c':
    corsa++ ;
    break ;
  case 'P':
    prisma++ ;
    break ;
  case 'p':
    palio++ ;
    break ;
  default:
    outros++ ;
    break ;
}
```

Comando condicional ternário

A expressão

```
expression1 ? expression2 : expression3 ;
```

é equivalente a

```
if (expression1)
  expression2 ;
else
  expression3 ;
```

Exemplo:

```
x = x < 5 ? x + 1 : 1 ;
```

equivale a:

```
if (x < 5)
  x = x + 1 ;
else
  x = 1 ;
```

Laços

Comando while

```
while (expression)
  command ;
```

ou

```
while (expression)
{
    command1 ;
    command2 ;
    ...
}
```

Exemplo:

```
i = 0 ;
while (i < 100)
{
    printf ("i vale %d\n", i) ;
    i++ ;
}
```

Comando do-while

```
do
    command ;
while (expression) ;
```

ou

```
do
{
    command1 ;
    command2 ;
    ...
}
while (expression) ;
```

Exemplo:

```
i = 100 ;
do
{
    printf ("i vale %d\n", i) ;
    i-- ;
}
while i ;
```

Comando for

```
for (initializations; conditions; updates)
    command ;
```

ou

```
for (initializations; conditions; updates)
{
    command1 ;
    command2 ;
}
```

```
...  
}
```

Exemplo: um laço de repetição com a variável de controle indo de 0 a 99:

```
for (i = 0; i < 100; i++)  
    printf ("i vale %d", i) ;
```

Deve-se observar que o comando for equivale a:

```
initializations;  
while (conditions)  
{  
    command1 ;  
    command2 ;  
    ...  
    updates ;  
}
```

Então o exemplo acima pode ser reescrito como:

```
i = 0 ; // inicialização  
while (i < 100) // condição  
{  
    printf ("i vale %d", i) ;  
    i++ ; // update  
}
```

Uma forma peculiar uso do for é o laço infinito, que não tem inicialização, condição nem update:

```
for (;;)   
{  
}
```

Desvios

Comando break

O comando break só é usado dentro de blocos switch, do, while ou for. Ao ser acionado, ele salta a execução para o primeiro comando após o bloco atual.

Exemplo:

```
for (;;)   
{  
    printf ("Aceita? (s/n) ") ;  
    resp = getchar() ;  
    if(resp == 's' || resp == 'n')  
        break;  
    printf ("\n") ;  
}  
// o break salta para cá
```

Comando continue

O comando `continue` é usado em laços `for`, `do` e `while`. Ele interrompe a iteração atual e inicia a próxima iteração do laço onde se encontra.

Exemplo:

```
int i;

for (i = -10; i < 10; i++)
{
    if (i == 0)
        continue; // pula para a próxima iteração do laço

    printf("%f\n", 15.0/i);
    // ...
}
```

Comando goto

O comando `goto` permite saltar a execução para locais definidos por rótulos ou etiquetas (*labels*), como mostra o exemplo a seguir:

```
goto PARTE2;

// ... (qualquer código)

PARTE2:
i = 0 ;
// ...
```



O comando `goto` deve ser usado com muito cuidado, pois ele pode tornar o fluxo de execução muito complexo (e portanto mais sujeito a erros). Como regra geral, **evite utilizá-lo**, a não ser que seja absolutamente necessário.

Comando return

O comando `return` encerra a função atual e retorna ao ponto de onde ela foi chamada. Ele pode ser chamado em qualquer local da função, não necessariamente no final dela.

Exemplo:

```
// compara dois inteiros "a" e "b", retornando:
// +1 se a > b
// -1 se a < b
// 0 se a = b

int compara (int a, b)
{
    if (a > b) return 1 ;
    if (a < b) return -1 ;
}
```

```
return 0 ; // a == b  
}
```

Comando exit

A função `exit (int status)` é provida pela biblioteca de sistema `stdlib.h` e permite o encerramento da aplicação. O status informado como parâmetro é devolvido ao sistema operacional (mais especificamente ao shell que lançou a aplicação).

```
exit (0) ;
```

From:
<https://wiki.inf.ufpr.br/maziero/> - **Prof. Carlos Maziero**

Permanent link:
https://wiki.inf.ufpr.br/maziero/doku.php?id=c:controle_de_fluxo

Last update: **2024/09/05 16:55**

