Introdução à Programação #10

Operadores Lógicos

Os operadores lógicos são, normalmente, utilizados em testes lógicos quando queremos incluir mais do que uma condição para que algo aconteça. Existem três operadores lógicos que vamos abordar agora: "&&", "||" e "!".

Operador "&&"

Na programação, o operador "&&" funciona como um "e" na nossa língua: para adicionar algo, neste caso, uma condição. Veja alguns exemplos:

```c
#include <stdio.h>

int main()
{
    int num1;
    num1 = 4;

    if (num1 > 0 && num1 <= 10) {
        printf("O número %d está entre 1 e 10.", num1);
    }

    return 0;
}
```

O que está expresso na condição acima é: se "num1" for maior que 0 e menor ou igual a 10 então executa o código que está ali dentro.

Este comando pode ser utilizado mais do que uma vez dentro de um teste condicional pois podemos colocar duas ou mais condições utilizando este operador. Basta adicionar "&&".

Operador "||"

O operador "||" refere-se a "ou", ou seja, utilizando este operador podemos executar uma condição do seguinte tipo: executa isto se x acontecer ou se y acontecer. Exemplo:

```c
#include <stdio.h>

int main()
{
    int num1;
    num1 = 3;

    if (num1 == 3 || num1 == 5) {
        printf("O número é 3 ou 5.\n");
    }

    return 0;
}
```

Como pode visualizar, o teste condicional acima verifica se a variável "num1" é 3 ou cinco e só depois executa o código condicionado ou não.

Operador "!"

Podemos traduzir o operador "!" para "o contrário de". Quando utilizado numa condição, por exemplo, este nega tudo o que lá está. Exemplo:

```c
#include <stdio.h>

int main()
{
    int num1;
    num1 = 0;

    if (!num1) {
        printf("O número é falso.\n");
    }

    return 0;
}
```

Relembrando que 0 é considerado falso e qualquer número maior é considerado verdadeiro, o código que está dentro do teste condicional acima só é executado se **num1 não for verdadeiro.**

Operadores de Incremento e Decremento

Os operadores de incremento e decremento são dois operadores que vão ser extremamente úteis na vida de qualquer programador.

Imagine que necessita de contar um número infinito de números independentemente se é no sentido crescente ou decrescente. Vai fazê-lo um a um? Não, não é necessário.

```c
#include <stdio.h>

int main()
{
    int num;

    num = 1;
    printf("O número é %d\n", num);

    num = num + 1;
    printf("O número é %d\n", num);

    num = num + 1;
    printf("O número é %d\n", num);

    return 0;
}
```

O código acima imprime a frase “O número é” e o número que está atribuído a “num” nesse momento. Começando em 1 e chegando a 3 no final do algoritmo.

Existe outra forma de adicionar e remover **uma unidade** a uma variável: “++” e “–“. Veja como ficaria o código anterior utilizando essa sintaxe:

```c
#include <stdio.h>

int main()
{
    int num;

    num = 1;
    printf("O número é %d\n", num);
```

```
9.
10.     num++;
11.     printf("O número é %d\n", num);
12.
13.     num++;
14.     printf("O número é %d\n", num);
15.
16.     return 0;
17. }
```

Isso irá imprimir o mesmo que o código mostrado anteriormente. Para remover uma unidade basta fazer do seguinte modo:

```
1.  #include <stdio.h>
2.
3.  int main()
4.  {
5.      int num;
6.
7.      num = 1;
8.      printf("O número é %d\n", num);
9.
10.     num--;
11.     printf("O número é %d\n", num);
12.
13.     return 0;
14. }
```

Estes operadores podem ser utilizados antes ou depois de uma variável: “num++” ou “++num”. Quando estes operadores são utilizados isoladamente, nenhuma diferença existe porém, quando estamos a efetuar uma atribuição, existe uma diferença. Analise o seguinte exemplo:

```
1.  #include <stdio.h>
2.
3.  int main()
4.  {
5.      int a, b, c;
6.
7.      a = 0;
8.      b = a++;
9.      c = ++a;
10.
11.     printf("A: %d, B: %d, C: %d", a, b, c);
12.
13.     return 0;
14. }
```

Primeiramente, declaramos três variáveis: “a”, “b” e “c”. De seguida, atribuímos o valor 0 à variável “a”. Quando atribuímos “a++” à variável “b”, o que vai acontecer?

A variável “b” vai assumir o valor 0 e um valor é incrementado a “a” ficando esta com o valor 1, ou seja, “b = a++” é um atalho para o seguinte:

```
1.  b = a;
2.  a++;
```

De seguida, quando atribuímos “++a” à variável “c”, estamos a incrementar primeiro a variável “a” e só depois é que o valor de esta é atribuído a “c”, ou seja, é um atalho para o seguinte:

```
++a;
c = a;
```

Como pode visualizar, estes operadores podem ser muito úteis em diversas situações. Mesmo que agora não esteja a visualizar estas situações, não se preocupe: irá ver muitas!

Controlo de fluxo com *while*

Já foi abordado o controlo de fluxo utilizando *if else*. Agora é a vez de um novo "tipo" de controlo de fluxo, o *while*ou, em português, "enquanto". A sintaxe do *while* é a seguinte:

```
while(condição) {
   //Algo acontece
}
```

Veja então o seguinte exemplo da utilização deste tipo de controlo de fluxo que imprime os números de zero a dez:

```
#include <stdio.h>

int main()
{
    int num;
    num = 0;

    while(num <= 10) {
        printf("%d\n", num);
        num++;
    }

    return 0;
}
```

Sem utilizar este controlo de fluxo, teríamos que escrever muito mais código de forma a conseguir imprimir os números de um a dez através de uma variável.

http://pplware.sapo.pt/tutoriais/vamos-programar-introducao-a-programacao-10