Linguagem de Programação C++

Universidade Federal de Santa Catarina

Departamento de Engenharia Elétrica, CTC

Prof. Eduardo Augusto Bezerra

"Esse material foi adaptado a partir de um website não identificado."

Operadores

Atribuição

Usado para atribuir valores localizados do lado direito de uma expressão, à variáveis localizadas do lado esquerdo do operador de atribuição da expressão:

```
a = 5;

b = a;
```

Por exemplo:

```
int a, b;    // a:?  b:?
a = 10;      // a:10 b:?
b = 4;       // a:10 b:4
a = b;       // a:4 b:4
b = 7;       // a:4 b:7
```

Resulta em **a** contendo **4** e **b** contendo **7**.

Uma diferença importante de C++ para outras linguagens é exemplificada pela seguinte expressão:

```
a = 2 + (b = 5);
```

que equivale a:

```
b = 5;
a = 2 + b;
```

Assim, a seguinte expressão também é válida em C++:

```
a = b = c = 5;
```

Operadores aritméticos (+, -, *, /, %)

Os cinco operadores aritméticos suportados pela linguagem:

+ adição
- subtração
* multiplicação
/ divisão
% módulo

O operador módulo (%) retorna o resto da divisão entre dois inteiros. Por exemplo, em **`a = 11 % 3;`**, a variável **`a`** receberá **`2`** como resultado, uma vez que este é o resto da divisão de 11 por 3.

Operadores de atribuição compostos (`+=, -=, *=, /=, %=, >>=, <<=, &=, ^=, |=`)

`valor += incremento;` equivale a **`valor = valor + incremento;`**
`a -= 5;` equivale a **`a = a - 5;`**
`a /= b;` equivale a **`a = a / b;`**
`preco *= unidades + 1;` equivale a **`preco = preco * (unidades + 1);`**

Incremento e decremento

Outros exemplos de economia na escrita de código são os operadores de incremento (**`++`**) e decremento (**`--`**), que incrementam ou reduzem de uma unidade o valor armazenado na variável. Equivalem a **`+=1`** e **`-=1`**, respectivamente:

```
a++;
a+=1;
a=a+1;
```

nos três exemplos o valor de **`a`** é incrementado de uma unidade.

Esses operadores podem ser utilizados para operações do tipo *prefix* e *suffix*:

Exemplo 1

```
B=3;
A=++B;
// A e' 4, B e' 4
```

Exemplo 2

```
B=3;
A=B++;
// A e' 3, B e' 4
```

No Exemplo 1, **`B`** é incrementado antes do seu valor ser copiado para **`A`**. Enquanto no Exemplo 2, o valor de **`B`** é primeiro copiado para **`A`** e depois esse valor de **`B`** é incrementado.

Operadores relacionais (`==, !=, >, <, >=, <=`)

Operadores relacionais são utilizados para realizar comparações entre duas expressões. O resultado da comparação é um valor booleano **`true`** ou **`false`**.

== Igual
!= Diferente

- `>` Maior que
- `<` Menor que
- `>=` Maior ou igual a
- `<=` Menor ou igual a

Exemplos:

`(7 == 5)`	Retorna **`false`**
`(5 > 4)`	Retorna **`true`**
`(3 != 2)`	Retorna **`true`**
`(6 >= 6)`	Retorna **`true`**
`(5 < 5)`	Retorna **`false`**

Variáveis podem ser utilizadas nas comparações. Supondo que **`a=2`**, **`b=3`** e **`c=6`**:

`(a == 5)`	Retorna **`false`**
`(a*b >= c)`	Retorna **`true`** pois (`2*3 >= 6`)
`(b+4 > a*c)`	Retorna **`false`** pois (`3+4 > 2*6`)
`((b=2) == a)`	Retorna **`true`**

Na última expressão (`(b=2) == a`), primeiro o valor **`2`** é atribuído para **`b`** e após isso **`b`** é comparado com **`a`**, que também possiu o valor **`2`**, resultando em **`true`**.

Em diversos compiladores anteriores a publicacação do padrão ANSI-C++, as operações de comparação retornavam 0 para “false”, e 1 para “true”.

Operadores lógicos (`!`, `&&`, `||`)

O operador **`!`** é equivalente a operação booleana NOT, e sua única função é inverter o valor do operando localizado a sua direita.

`!(5 == 5)`	retorna **`false`** uma vez que a expressão à direita(`5 == 5`) será **`true`**.
`!(6 <= 4)`	retorna **`true`** uma vez que a expressão à direita (`6 <= 4`) será **`false`**.
`!true`	retorna **`false`**.
`!false`	retorna **`true`**.

Operadores lógicos **`&&`** e **`||`** correspondem às operações lógocas *AND* e *OR* respectivamente.

Primeiro operando **a**	Segundo operando **b**	`resultado a && b`	`resultado a || b`

true	true	**true**	**true**
true	false	**false**	**true**
false	true	**false**	**true**
false	false	**false**	**false**

Por exemplo:

`( (5 == 5) && (3 > 6) )` retorna **`false`** (*`true`* && *`false`*).
`( (5 == 5) || (3 > 6))` retorna **`true`** (*`true`* || *`false`*).

Operador condicional (`?`)

`condição` **`?`** *`resultado1 : c`*

Se *`condição`* é **`true`** a expressão retorna *`resultado1`*, caso contrário irá retornar *`resultado1`*.

`7==5 ? 4 : 3`	retorna **3** uma vez que **7** é diferente de **5**.
`7==5+2 ? 4 : 3`	retorna **4** uma vez que **7** é igual a **5**+**2**.
`5>3 ? a : b`	retorna **`a`**, uma vez que **5** é maior que **3**.
`a>b ? a : b`	retorna o maior entre **`a`** ou **`b`**.

Operadores “bit-a-bit” (`&, |, ^, ~, <<, >>`)

Utilizados para realizar operações bit-a-bit entre dois operandos, considerando a representação binária desses operandos.

op	asm	Descrição
&	**AND**	E lógico
|	**OR**	OU lógico
^	**XOR**	OU exclusivo lógico
~	**NOT**	Complemento (inversão de bits)
<<	**SHL**	Deslocamento à esquerda
>>	**SHR**	Deslocamento à direita

Operadores para conversão de tipos (type casting)

Possibilitam a conversão do tipo de um dado para um outro tipo. Existem diversas formas de realizar essa conversão, sendo que a mais popular é aquela compatível com C, ou seja, preceder a expressão com o tipo a ser convertido entre parenteses **`()`**:

```
int i;
float f = 3.14;
i = (int) f;
```

Esse trecho de código converte o número em ponto flutuante (float) **`3.14`** para um valor inteiro (**`3`**). Em uma outra forma de realizar a conversão o valor a ser convertido fica entre parenteses, sendo precedido pelo tipo:

```
i = int ( f );
```

sizeof()

Retorna o número de bytes do tipo ou objeto:

```
a = sizeof (char);
```

`a` recebe 1, uma vez que **`char`** possui 1 byte. O valor retornado é uma constante.

Outros operadores

Outros operadores a serem discutidos adiante incluem os operadores utilizados em aritmética de ponteiros, e outros específicos para programação orientada a objetos.

Prioridade de operadores

Considere a expressão:

```
a = 5 + 7 % 2
```

Qual a alternativa correta?

`a = 5 + (7 % 2)` com resultado **6**, ou
`a = (5 + 7) % 2` com resultado **0**

Resultado 6 é o correto, uma % possui maior prioridade conforme pode ser observado na tabela a seguir.

Prioridade	Operador	Descrição	Direção
1	`::`	escopo	Esquerda
2	`() [ ] -> . sizeof`		Esquerda
3	`++ --`	Incremento/decremento	Direita
	`~`	Complemento (bit-a-bit)	
	`!`	NOT	
	`& *`	Referência (ponteiros)	
	(*type*)	Type casting	
	`+ -`	Sinais unários	
4	`* / %`	Operações aritméticas	Esquerda

5	`+ -`	Operações aritméticas	Esquerda
6	`<< >>`	Deslocamento (bit-a-bit)	Esquerda
7	`< <= > >=`	Operadores relacionais	Esquerda
8	`== !=`	Operadores relacionais	Esquerda
9	`& ^ |`	Operadores bit-a-bit	Esquerda
10	`&& ||`	Operadores lógicos	Esquerda
11	`?:`	Condicional	Direita
12	`= += -= *= /= %=` `>>= <<= &= ^= |=`	Atribuição	Direita
13	`,`	Vírgula, separação	Esquerda

A última coluna da tabela indica qual operador deve ser executado primeiro (mais a esquerda ou mais a direita) nos casos onde existem diversos operadores com o mesmo nível de prioridade.