Residência em Software

Linguagem C++: Expressões

Professores:

Álvaro Coelho, Edgar Alexander, Esbel Valero e Hélder Almeida

INSTITUIÇÃO EXECUTORA: cepedi, UESC

COORDENADORA: MCTI FUTURO, Softex

APOIO: Ministério da Ciência, Tecnologia e Inovação, Governo Federal Brasil União e Reconstrução

Residência em Software

Operadores: Operador de atribuição

- Como vimos na aula anterior, precisamos de um nome para poder se referir a "algo na memória";
- Este "algo na memória", que a gente chamou de objeto, é uma região contígua de armazenamento;
- Na aula anterior vimos como inicializar variáveis, que são objetos identificáveis por um nome;
- Após a definição de uma variável se pode atribuir um novo valor a um objeto utilizamos o operador atribuição.

Residência em Software

Operadores: Operador de atribuição

- Operador (=) pode ser utilizado dentro de qualquer expressão válida da linguagem;
- Sintaxe:

```
identificador_de_variavel = expressão;
```

- O destino, ou **lvalue,** tem que um operando deve ser um lvalue modificável;
- A palavra "lvalue" foi originalmente cunhada para significar "algo que pode estar no lado esquerdo de uma atribuição."

Residência em Software

Operadores: Operador de atribuição

- Operador (=) pode ser utilizado dentro de qualquer expressão válida da linguagem;
- Sintaxe:

```
identificador_de_variavel = expressão;
```

- O rvalue pode ser uma variável ou qualquer expressão válida da linguagem;
- O C++ permite atribuições múltiplas em um único comando.

```
var1 = var2 =var3 = var4 = expressão;
```

Residência em Software

Operadores: Operador de atribuição

- O resultado de uma atribuição é seu operando à esquerda.
- O tipo do resultado é o tipo do operando esquerdo.
- Se os tipos dos operandos esquerdo e direito forem diferentes, o operando direito é convertido para o tipo esquerdo
- No novo padrão, podemos usar uma lista inicializadora entre chaves em uma atribuição
- Se o operando esquerdo for de um tipo básico, a lista inicializadora pode conter no máximo um valor e esse valor não deve exigir uma conversão de tipo com perda de informação

Residência em Software

Operadores: Operador de atribuição

- Conversão de tipos: refere-se à situação em que variáveis de um tipo são misturadas com variáveis de outro tipo numa expressão válida em **C++**;
- A linguagem **C++** permite conversão automática de tipos; Nenhuma mensagem de erro ou aviso tem lugar quando uma conversão de tipos acontece;
- Regra de conversão de tipos: Em um comando de atribuição o valor do **rvalue** é convertido no tipo do **lvalue**, antes de fazer a atribuição.

```
lvalue = rvalue;
```

```
float pi = 3 + 0.1415;
int intPi = pi;
```

- As conversões de tipos apenas mudam a forma em que o valor é representado;

Residência em Software

Operadores: Operador de atribuição

- Em **C++** alguns tipos estão relacionados entre si.
- Quando dois tipos estão relacionados, podemos usar um objeto ou valor de um tipo onde se espera um operando do tipo relacionado.
- Dois tipos estão relacionados se houver uma forma de conversão entre eles.
- Esta conversão automática ou implícita entre tipos aritméticos são definidas de forma a preservar a precisão quando possível
- O compilador converte operandos automaticamente nas seguintes circunstâncias:
  - Na maioria das expressões, valores de tipos integrais (`bool`, `char` ou `int`) menores que um `int` são primeiro promovidos a um tipo integral maior apropriado.
  - Em condições, as expressões não `bool` são convertidas em `bool`.
  - Nas inicializações, o inicializador é convertido para o tipo da variável; em atribuições, o operando do lado direito é convertido para o tipo do lado esquerdo.
  - Em expressões aritméticas e relacionais com operandos de tipos mistos, os tipos são convertidos em um tipo comum.

Residência em Software

Operadores: Operador de atribuição

```
1   #include <iostream>
2   using namespace std;
3
4   int main(void)
5   {
6       char op, ch;
7       int i = 0, j, k;
8       double x, y, z;
9       bool isOk, isNotOk;
10      //o rvalue pode ser um objeto literal
11      op = 's'; // op = 115
12      ch = 78; // ch = 'N' conversão implícita de tipo int -> char
13      // atribuições múltiplas
14      j = k = i; // j = (k = i)
15      x = 0.33333;
16      y = 1.0/3.0;
17      z = 3.3333e-1;
18      isOk = true;
19      isNotOk = false;
20      return 0;
21  }
```

Residência em Software

Conversão explícita de tipo

- Às vezes, queremos forçar explicitamente um objeto a ser convertido em um tipo diferente. Nestes casos se utiliza cast para requerer esta conversão explícita de tipo.
- Embora necessários às vezes, os casts são construções inerentemente perigosas.
- Tipos de cast
  - static_cast: informa ao leitor do programa e ao compilador que estamos cientes e não estamos preocupados com a possível perda de precisão.

    ```
    static_cast<type> (expression)
    ```
  - Estilo padrão C:

    ```
    (type) expression
    ```

Residência em Software

Operadores: Operadores aritméticos

- Os operadores aritméticos tradicionais funcionam em **C** com seu significado matemático indica:
  - Estes operadores estão definidos para tipos numéricos;
  - os operadores +, -, *, / e % são classificados como operadores binários;
  - o operador – pode ser unário;
  - Quando / é aplicado a inteiro ou caractere qualquer resto é truncado (operação de divisão inteira)
  - O operador %, que não é tradicionalmente utilizado na notação matemática, devolve o resto da divisão inteira e, por tanto, ambos operandos devem ser inteiros;

Residência em Software

Operadores: Operadores aritméticos

- **C++** inclui mais dois operadores aritméticos:
  - operador de incremento ++;
  - operador de decremento --;
  - x++ é o mesmo que x = x + 1;
- Os operadores decremento e incremento são unários;
- O operador incremento(decremento) pode aparecer antes o depois do operando;

```
cout << "Usando o operador incremento: " << endl;
cout << "i = " << i << endl;
cout << "i++ = " << i++ << endl;
cout << "i = " << i << endl;
cout << "++i = " << ++i << endl;
cout << "i = " << i << endl;
```

```
Usando o operador incremento:
i = -2
i++ = -2
i = -1
++i = 0
i = 0
```

- Estes operadores ++ e -- permitem fazer as operações de forma mais eficientemente.

Operadores: Operadores aritméticos

- Em **C++** as expressões que envolvem vários operadores aritméticos, são resolvidos em uma seqüência determinada pelas regras de precedência:
  - Os parêntesis tem o mesmo significado que em expressões algébricas, ou seja, as operações entre parênteses têm maior precedência que as operações fora deles;
  - Quando dois operadores com igual precedência estão no mesmo nível, a expressão é calculada da esquerda a direita;

Operadores	**Ordem de Precedência**
(...)	[1]. Se houver parênteses aninhados se resolve de dentro para fora.
++ ou --	[2]. Quando presentes, operações de incremento e decremento são resolvidos primeiro;
*, / ou %	[3]. Posteriormente se multiplica, divide ou se calcula o resto da divisão
+ ou -	[4]. Finalmente se efetua adições ou subtrações.

Operadores: Operadores aritméticos

- Em **C++** as expressões que envolvem vários operadores aritméticos, são resolvidos em uma seqüência determinada pelas regras de precedência:
  - Os parêntesis tem o mesmo significado que em expressões algébricas, ou seja, as operações entre parênteses têm maior precedência que as operações fora deles;
  - Quando dois operadores com igual precedência estão no mesmo nível, a expressão é calculada da esquerda a direita;

```
i = ++i + 2 * --j - k / 4 + 6 % 3;  //ordem de precedência dos operadores
cout << "i = ++i + 2 * --j - k / 4 + 6 % 3 = " << i << endl;
i = (++i + 2) * (--j - k) / ((4 + 6) % 3);  //usando parenteses
cout << "i = ++(i + 2) * (--j - k) / ((4 + 6) % 3) = " << i << endl;

x = ++x + 2 * --y - z / 4.3 + 6.1 * 3.2;  //ordem de precedência dos operadores
cout << "x = ++x + 2 * --y - z / 4.3 + 6.1 * 3.2 = " << x << endl;
x = (++x + 2) * (--y - z) / ((4.3 + 6.1) * 3.2);  //usando parenteses
cout << "x = ++(x + 2) * (--y - z) / ((4.3 + 6.1) * 3.2) = " << x << endl;
```

```
i = ++i + 2 * --j - k / 4 + 6 % 3 = -1
i = ++(i + 2) * (--j - k) / ((4 + 6) % 3) = -6

x = ++x + 2 * --y - z / 4.3 + 6.1 * 3.2 = 15.3423
x = ++(x + 2) * (--y - z) / ((4.3 + 6.1) * 3.2) = -2.12745
```

Residência em Software

Operadores: Operadores aritméticos

```cpp
#include <iostream>
using namespace std;

int main(void)
{
    int i = 0, j, k;
    cout << "i = " << i << endl;
    j = i + 1;  //operador de adição
    cout << "j = i + 1 = " << j << endl;
    k = 7 - i;  //operador de subtração
    cout << "k = 7 - i = " << k << endl;
    i = i * -2;  //operador de multiplicação, operador unário -
    cout << "i = i * -2 = " << i << endl;
    j = j / 2;  //operador de divisão
    cout << "j = j / 2 = " << j << endl;
    k = k % 2;  //operador de resto
    cout << "k = k % 2 = " << k << endl;
    cout << "Usando o operador decremento: " << endl;
    cout << "i = " << i << endl;
    cout << "i-- = " << i-- << endl;
    cout << "i = " << i << endl;
    cout << "--i = " << --i << endl;
    cout << "i = " << i << endl;
    cout << "Usando o operador incremento: " << endl;
    cout << "i = " << i << endl;
    cout << "i++ = " << i++ << endl;
    cout << "i = " << i << endl;
    cout << "++i = " << ++i << endl;
    cout << "i = " << i << endl;
    i = ++i + 2 * --j - k / 4 + 6 % 3;  //ordem de precedência dos operadores
    cout << "i = ++i + 2 * --j - k / 4 + 6 % 3 = " << i << endl;
    i = (++i + 2) * (--j - k) / ((4 + 6) % 3);  //usando parenteses
    cout << "i = ++(i + 2) * (--j - k) / ((4 + 6) % 3) = " << i << endl;
    return 0;
}
```

Residência em Software

Operadores: Operadores aritméticos

```cpp
#include <iostream>
using namespace std;

int main(void)
{
    double x = 3.14, y, z;
    cout << "x = " << x << endl;
    y = x + 1.0;  //operador de adição
    cout << "y = x + 1.0 = " << y << endl;
    z = 7.0 - x;  //operador de subtração
    cout << "z = 7.0 - x = " << z << endl;
    x = x * -2.0;  //operador de multiplicação, operador unário -
    cout << "x = x * -2.0 = " << x << endl;
    y = y / 2.0;  //operador de divisão real
    cout << "y = y / 2.0 = " << y << endl;
    y = 5 / 2; //operador de divisão inteira
    cout << "y = 5 / 2 = " << y << endl;
    x = ++x + 2 * --y - z / 4.3 + 6.1 * 3.2;  //ordem de precedência dos operadores
    cout << "x = ++x + 2 * --y - z / 4.3 + 6.1 * 3.2 = " << x << endl;
    x = (++x + 2) * (--y - z) / ((4.3 + 6.1) * 3.2);  //usando parenteses
    cout << "x = ++(x + 2) * (--y - z) / ((4.3 + 6.1) * 3.2) = " << x << endl;
    return 0;
}
```

Residência em Software

Operadores: Operadores aritméticos

```
a = 1; b = 2; c = 3; d = 4; e = -1
```

Operador	Exemplo	Operação	Resultado
+=	a += 7	a = a + 7	8
-=	b -= 2	b = b - 2	0
*=	c *= 5	c = c * 5	15
/=	d /= 2	d = d / 2	2
%=	e %= 2	e = e % 2	1

Residência em Software

Operadores: Operadores relacionais

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

- Referem-se às relações que os valores podem ter uns com os outros;
- As expressões que utilizam operadores relacionais retornam valores boleanos: `true` ou `false`

Residência em Software

Operadores: Operadores lógicos

Operador	Significado
&&	AND
||	OR
!	NOT

- Os operadores lógicos aceitam operandos de qualquer tipo que possam ser convertidos em bool.
- Retornam valores do tipo bool.
- Operandos aritméticos com valor zero são falsos, todos os outros valores são verdadeiros.

Os operadores lógicos AND e OR sempre avaliam seu operando esquerdo antes do direito. Além disso, o operando direito é avaliado se e somente se o operando esquerdo não determinar o resultado.

- O lado direito de um `&&` é avaliado se e somente se o lado esquerdo for `true`.
- O lado direito de um `||` é avaliado se e somente se o lado esquerdo for `false`.

Residência em Software

Operadores: Operadores lógicos

a	b	a && b	a || b	!a
0	0	0	0	1
0	1	0	1	1
1	1	1	1	0
1	0	0	1	0

Operadores: Operadores lógicos

- Os operadores lógicos e relacionais tem menor precedência que os operadores aritméticos.

Operadores	Ordem de Precedência
!	[1]. Primeiramente se efetua qualquer operação de negação;
<, ≤, >, ≥	[2]. Logo a seguir se implementam os comparações;
==, !=	[3]. Seguido das operações de igualdade ou diferença;
&&	[4]. POsteriormente se avaliam as operações lógicas tipo AND;
||	[5]. Concluindo com as operações lógicas tipo OR.

Operadores: Operadores lógicos

```cpp
#include <iostream>
using namespace std;

int main(void)
{
    bool isTrue, isFalse;
    int a = 1, b = 2, c = 3, d = 4, e = -1;
    cout << "a = " << a << ", b = " << b
         << ", c = " << c << ", d = " << d << ", e = " << e << endl;
    isTrue = a > e; // maior que retorna true
    cout << "a > e is " << isTrue << endl;
    isFalse = a > b; // maior que retorna false
    cout << "a > b is " << isFalse << endl;
    isTrue = a < b; // menor que retorna true
    cout << "a < b is " << isTrue << endl;
    isFalse = a < e; // menor que retorna false
    cout << "a < e is " << isFalse << endl;
    isTrue = d >= 2 * b; // maior ou igual retorna true
    cout << "d >= 2 * b is " << isTrue << endl;
    isFalse = d >= 2 * c; // maior ou igual retorna false
    cout << "d >= 2 * c is " << isFalse << endl;
    isTrue = a <= 2 * b; // menor ou igual retorna true
    cout << "a <= 2 * b is " << isTrue << endl;
    isFalse = a <= 2 * c; // menor ou igual retorna false
    cout << "a <= 2 * c is " << isFalse << endl;
    isTrue = 2*a == b; // igual retorna true
    cout << "2*a == b is " << isTrue << endl;
    isFalse = b == c; // igual retorna false
    cout << "b == c is " << isFalse << endl;
    isTrue = 2*a != c; // diferente retorna true
    cout << "2*a != c is " << isTrue << endl;
    isFalse = 2*a != b; // diferente retorna false
    cout << "2*a != b is " << isFalse << endl;
    cout << "Tabela verdade do operador lógico AND:" << endl;
    cout << "true && true = " << (true && true) << endl;
    cout << "true && false = " << (true && false) << endl;
    cout << "false && true = " << (false && true) << endl;
    cout << "false && false = " << (false && false) << endl;
    cout << "Tabela verdade do operador lógico OR:" << endl;
    cout << "true || true = " << (true || true) << endl;
    cout << "true || false = " << (true || false) << endl;
    cout << "false || true = " << (false || true) << endl;
    cout << "false || false = " << (false || false) << endl;
    cout << "Tabela verdade do operador lógico NOT:" << endl;
    cout << "!true = " << (!true) << endl;
    cout << "!false = " << (!false) << endl;
    return 0;
}
```

Operadores: Operador condicional

Sintaxe:

```
(condição) ? instrução_A : instrução_B;
```

Este operador é executado avaliando a condição.

Se a condição for verdadeira, então a `instrução_A` é avaliada;

caso contrário, a `instrução_B` é avaliada.

Exemplos:

```
b = (a > 0) ? a++ : a--;
b = (c > 0) ? sqrt(c) : sqrt(-c);
```

```
isTrue = a > e; // maior que retorna true
cout << "a > e is " << (isTrue ? "true":"false") << endl;
isFalse = a > b; // maior que retorna false
cout << "a > b is " << (isFalse ? "true":"false") << endl;
```

Residência em Software

Operadores: Operador condicional

```cpp
#include <iostream>
using namespace std;

int main(void)
{
    bool isTrue, isFalse;
    int a = 1, b = 2, c = 3, d = 4, e = -1;
    cout << "a = " << a << ", b = " << b
         << ", c = " << c << ", d = " << d << ", e = " << e << endl;
    isTrue = a > e; // maior que retorna true
    cout << "a > e is " << (isTrue ? "true":"false") << endl;
    isFalse = a > b; // maior que retorna false
    cout << "a > b is " << (isFalse ? "true":"false") << endl;
    isTrue = a < b; // menor que retorna true
    cout << "a < b is " << (isTrue ? "true":"false") << endl;
    isFalse = a < e; // menor que retorna false
    cout << "a < e is " << (isFalse ? "true":"false") << endl;
    isTrue = d >= 2 * b; // maior ou igual retorna true
    cout << "d >= 2 * b is " << (isTrue ? "true":"false") << endl;
    isFalse = d >= 2 * c; // maior ou igual retorna false
    cout << "d >= 2 * c is " << (isFalse ? "true":"false") << endl;
    isTrue = a <= 2 * b; // menor ou igual retorna true
    cout << "a <= 2 * b is " << (isTrue ? "true":"false") << endl;
    isFalse = c <= 2 * a; // menor ou igual retorna false
    cout << "c <= 2 * a is " << (isFalse ? "true":"false") << endl;
    isTrue = 2*a == b; // igual retorna true
    cout << "2*a == b is " << (isTrue ? "true":"false") << endl;
    isFalse = b == c; // igual retorna false
    cout << "b == c is " << (isFalse ? "true":"false") << endl;
    isTrue = 2*a != c; // diferente retorna true
    cout << "2*a != c is " << (isTrue ? "true":"false") << endl;
    isFalse = 2*a != b; // diferente retorna false
    cout << "2*a != b is " << (isFalse ? "true":"false") << endl;
    return 0;
}
```

Residência em Software

Operadores: Operadores bit a bit

Operador	Significado
&	AND
|	OR
^	OR exclusivo (XOR)
~	Complemento de um
>>	Deslocamento à esquerda
<<	Deslocamento à direita

Os operadores bit a bit pegam operandos do tipo integral (caracteres e inteiros) que eles usam como uma coleção de bits.

Esses operadores nos permitem testar e definir bits individuais.

Residência em Software

Operadores: Operadores bit a bit

Já usamos as versões sobrecarregadas dos operadores >> e << que a biblioteca IO define para fazer entrada e saída.

- >> enviar para
- << obter de

Já como operador bit a bit o significado embutido desses operadores é que eles executam um deslocamento bit a bit em seus operandos.
Eles fornecem um valor que é uma cópia do operando esquerdo (possivelmente promovido) com os bits deslocados conforme direcionado pelo operando direito.
O operando do lado direito não deve ser negativo e deve ser um valor estritamente menor que o número de bits no resultado. Caso contrário, a operação é indefinida.
Os bits são deslocados para a esquerda (<<) ou para a direita (>>).

Operadores: Operadores bit a bit

```cpp
#include <iostream>
using namespace std;

int main(void)
{
    unsigned char byteA, byteB, byteC;
    byteA = 255; // 11111111
    byteB = 0; // 00000000
    byteC = byteA & byteB; // 00000000
    cout << "byteA & byteB = " << (int)byteC << endl;
    byteC = byteA | byteB; // 11111111
    cout << "byteA | byteB = " << (int)byteC << endl;
    byteC = byteA >> 4; // 00001111
    cout << "byteA >> 4 = " << (int)byteC << endl;
    byteC = byteA << 4; // 11110000
    cout << "byteA << 4 = " << (int)byteC << endl;
    byteC = ~byteA; // 00000000
    cout << "~byteA = " << (int)byteC << endl;
    byteC = byteA ^ (byteA << 4); // 11111111 ^ 11110000 = 00001111
    cout << "byteA ^ (byteA << 4) = " << (int)byteC << endl;
    return 0;
}
```

```
(base) evalero@Jorel output % ./"bitabit"
byteA & byteB = 0
byteA | byteB = 255
byteA >> 4 = 15
byteA << 4 = 240
~byteA = 0
byteA ^ (byteA << 4) = 15
```

Residência em Software

Operadores: Operadores bit a bit

```
unsigned char a = 255, b = 0;
```

Operador	Exemplo	Operação	Resultado
<<=	a <<= 4	a = a << 4	240
>>=	a >>= 4	a = a >> 4	15
&=	a &= b	a = a & b	0
^=	a ^= (a << 4)	a = a ^(a<<4)	15
|=	a |= b	a = a | b	255

Operadores: Operador sizeof

O operador `sizeof` retorna o tamanho, em bytes, de uma expressão ou um nome de tipo.

O resultado de sizeof é uma expressão constante size_t

O operador assume uma das duas formas:

```
sizeof (type)
sizeof expr
```

```
char letra = 'a'; // letra é uma variável de tipo char
//a cada caracteres corresponde um valor inteiro
cout << "A letra " << letra << " corresponde ao valor " << int(letra) << endl;
cout << "O tipo char ocupa " << sizeof(char) << " bytes na memória.\n";
```

```
A letra a corresponde ao valor 97
O tipo char ocupa 1 bytes na memória.
```