Sobrecarga de Operadores

Prof. Valdir Pedrinho de Tomin Junior, Eng.

Universidade Federal de Santa Catarina
Centro Tecnológico de Joinville
EMB5631 – Programação III

e-mail: valdir.pedrinho@ufsc.br

11 de setembro de 2018

Introdução

O que é a sobrecarga?

- O programador pode redefinir o significado da maior parte dos operadores de C++, quando pelo menos um operando é um objeto de classe;
- O C++ permite ao programador sobrecarregar a maioria dos operadores para que eles se tornem sensíveis ao contexto em que são utilizados.

Exemplos de Operadores Sobrecarregados

- Um exemplo de um operador sobrecarregado construído no C++ é o « que é utilizado como operador de inserção de fluxo e como operador de bits de deslocamento para a esquerda;
- O operador » também é sobrecarregado; é utilizado como operador de extração de fluxo e como operador de bits de deslocamento para a direita;
- Esses dois operadores são sobrecarregados na C++ Standard Library.

Introdução

Redução do tamanho da linguagem

Operadores tem, por padrão, mais de uma função, por exemplo * que pode significar:

- multiplicação;
- conteúdo de ponteiro;
- declaração de ponteiro.

* é inerente à linguagem C/C++ e não pode ser modificada pelo programador.

Legibilidade do código

Aumenta a legibilidade do código, por exemplo a adição de duas variáveis de um tipo qualquer definido pelo usuário. É mais natural na forma: **s = a + b**, do que: **s = soma (a, b)**, pois, na segunda, é preciso conhecer a função.

Fundamentos

Características

- Um operador é sobrecarregado escrevendo uma definição de método **não-static** ou uma definição de função global;
- Operadores que não precisam de sobrecarga:
  - O operador de atribuição (=) pode ser utilizado com toda a classe para realizar atribuição de membro a membro dos membros de dados da classe;
  - O operador de endereço (&) retorna o endereço do objeto na memória;
  - O operador vírgula (,) avalia a expressão à sua esquerda e, depois, à sua direita.

Sintaxe

```
tipo operator Simbolo_do_Operador(lista de operandos);
```

Restrições à sobrecarga de operadores

Restrições

- Não é possível criar novos operadores, só sobrecarregar operadores existentes;
- O operador sobrecarregado não pode alterar as regras de precedência e associatividade;
- A associatividade de um operador (isto é, se o operador é aplicado da direita para a esquerda ou da esquerda para a direita) não pode ser alterada pela sobrecarga;
- Não é possível alterar o número de operandos;
- Não se pode redefinir as operações para os tipos da linguagem;
- Não se pode combinar sobrecargas para gerar uma terceira função operadora.

Restrições à sobrecarga de operadores

Permitem sobrecarga

Operadores que podem ser sobrecarregados							
+	-	*	/	%	^	&	|
~	!	=	<	>	+=	-=	*=
/=	%=	^=	&=	|=	<<	>>	>>=
<<=	==	!=	<=	>=	&&	||	++
--	->*	,	->	[]	()	new	delete
new[]	delete[]						

Não permitem sobrecarga

Operadores que não podem ser sobrecarregados			
.	.*	::	?:

Métodos vs Funções Globais

Comparativo

- As funções operadoras podem ser métodos ou funções globais;
- As funções globais são frequentemente feitas **friends** por razões de desempenho;
- Os métodos utilizam o ponteiro **this** implicitamente para obter um de seus argumentos de objeto de classe (primeiro operando);
- Os argumentos para ambos os operandos de um operador binário devem ser explicitamente listados em uma chamada de função global, em um método só o operando à direita é passado.

Métodos vs Funções Globais

Operadores que devem ser sobrecarregado como métodos

(), [], -> e operadores de atribuição.

Operadores que devem ser sobrecarregado como funções

- Se o operando esquerdo precisar ser um objeto de uma classe diferente ou um tipo fundamental, essa função operadora deve ser implementada como uma função global;
- Por exemplo, o operador de inserção de fluxo («) sobrecarregado é utilizado em uma expressão em que o operando esquerdo tem o tipo **ostream &**, como em **cout « classObject**.

Exemplo

Classe caixa, sobrecarga do operador +.

Sobrecarregando Operadores

Passagem/Retorno de referência **const**

Normalmente os parâmetros passados às funções operadoras ou retornados são por referência e constantes, por quê?

- Passar por referência constante é, em termos práticos, passar por valor. A diferença se dá no fato que não será criada uma cópia do objeto dentro do escopo da função;
- Retornar uma referência constante segue o mesmo princípio e ainda evita erros de atribuições irregulares (o que aconteceria com o retorno por valor).

Exemplo

Classe Vetor2D, sobrecarga do operador +=.

Sobrecarregando Operadores

Exercício

Escreva uma classe **Circulo**, com as propriedades de um circulo e sobrecarregue o operador $>$ (maior que) para que você possa comparar dois círculos. A lógica que você usará é comparar os raios dos objetos **Circulo** e decidir qual é maior.

Exercício

Escreva uma classe **Retangulo**, com as propriedades de um retângulo e sobrecarregue o operador $<$ (menor que) para que você possa comparar dois retângulos. A lógica que você usará é comparar as áreas dos objetos **Retangulo** e decidir qual é maior.

Sobrecarregando Operadores

Exercício

Escreva uma classe **Time** que representa o tempo. A classe deve ter três campos: horas, minutos e segundos. Deve ter um construtor para inicializar as horas, minutos e segundos. Um método **printTime()** para imprimir a hora atual. Sobrecarregue os seguintes operadores: operador + (adiciona dois objetos **Time**, com base no relógio de 24 horas) e o < (compara dois objetos **Time**).

Sobrecarregando Operadores

Operadores de inserção e extração de fluxo

É possível sobrecarregar os operadores » e « para redefinir os comandos **cin** e **cout**.

Exemplo

Programa 11.3-5 do *DEITEL. C++, Como programar. 5ª ed.*

Exercício – redefinir **cin** e **cout** para **Complex**

```
class Complex {
    friend ostream & operator << (...);
    friend istream & operator >> (...);
    int real, imag;
public:
    Complex(int r = 0, int i =0);
};
```

Classes com Atributos Ponteiros

Operador de Atribuição e Ponteiros...

Vimos que o operador de atribuição não precisa ser sobrecarregado para tipos definidos pelo programador (classes), mas quando determinado objeto possuir ponteiros, ao ser feita uma atribuição, ambos os objetos apontarão para o mesmo recurso!

Construtor de Cópia e Ponteiros...

O mesmo problema vale para o construtor de cópia!

Construtor de cópia

É um construtor, normalmente implícito, que recebe uma referência constante a um objeto do sua própria classe, fazendo que o novo objeto instanciado receba os mesmos valores – em seus atributos.

Classes com Atributos Ponteiros

Uma classe com ponteiros...

```
class Inteiro {
    int *recurso;
public:
    Inteiro(int x=0) { recurso = new int(x); }
    Inteiro(const Inteiro &outro) {
        recurso = new int;
        *recurso = *(outro.recurso);
    }
    ~Inteiro() { delete recurso; }
    const Inteiro &operator=(const Inteiro &outro) {
        *recurso = *(outro.recurso);
        return *this;
    }
};
```

Sobrecarregando ++ e − −

Como o compilador difere de prefixado e pós-fixado?

Para sobrecarregar os operadores ++/− − para permitir o prefixado ou pós-fixado, toda função operadora sobrecarregada deve ter uma assinatura distinta para que o compilador consiga determinar que versão é pretendida.

Sobrecarregando o operador de incremento prefixado

- Método:

```
NomeDaClasse & operator ++ ();
```

- Função Global:

```
NomeDaClasse & operator ++ (NomeDaClasse &);
```

Sobrecarregando ++ e − −

Sobrecarregando o operador de incremento pós-fixado

- Método:

```
NomeDaClasse operator++(int)
```

- Função Global:

```
NomeDaClasse operator++(NomeDaClasse &, int)
```

Exemplo

Classe **Time**, sobrecarga ++ pós-fixado e prefixado.

Convertendo entre Tipos

Construtores de conversão ou coerção

- Frequentemente é necessário converter dados de um tipo em dados de outro tipo. Isso pode acontecer em atribuições,em cálculos, na passagem de valores para funções e no retorno de valores a partir de funções;
- Em tipos definidos pelo usuário essa conversão deve ser explicitada.

Sintaxe

```
operator tipo () const; ou
operator tipo ();
```

Exemplo

Classe **Complex**, sobrecarga de conversão.

Construtores **explicit**

Chamada implícita de construtores

- Qualquer construtor de **um único argumento** pode ser utilizado pelo compilador para realizar uma conversão implícita — o tipo recebido pelo construtor é convertido em um objeto da classe em que o construtor é definido;
- A conversão é automática e o programador não precisa utilizar um operador de coerção;
- Em algumas situações, as conversões implícitas são indesejáveis ou propensas a erros.

Chamada explícita de construtores

Basta utilizar a palavra reservada **explicit**

Exemplo

Classe **Inteiro**

Array...

Exemplo – Sobrecarga do operador []

```
class SafeArray {
private:
    int arr[SIZE];
public:
    SafeArray() {
        for(int i = 0; i < SIZE; i++) { arr[i] = i;}
    }
    int &operator[](int i) {
        if( i > SIZE ) {
            cout << "Index out of bounds" <<endl;
            return arr[0];
        }
        return arr[i];
    }
};
```

Array...

Exercício – Implemente a **SafeArray** Completa

```
class SafeArray {
  friend ostream &operator<<(ostream &, const
     SafeArray & );
  friend istream &operator>>(istream &, SafeArray & );
public:
  explicit SafeArray(int = 10 );
  SafeArray(const SafeArray & );
  ~SafeArray(void);
  int getSize(void) const;
  const SafeArray & operator=(const SafeArray & );
  bool operator == (const SafeArray & ) const;
  bool operator != (const SafeArray & ) const;
  const int & operator [](int) const;
private:
  int size; int * ptr;
};
```