

Linguagens de Programação

Prof. Miguel Elias Mitre Campista

`http://www.gta.ufrj.br/~miguel`

Parte IV

Introdução à Programação em C++ (Continuação)

Relembrando da Última Aula...

- Objetos e funções membro `const`
- Funções `friend`
- Uso do ponteiro `this`
- Gerenciamento de memória
 - Uso do `new` e do `delete`
- Classes `proxy`
- Mais exemplos de programação orientada a objetos...

Sobrecarga de Operadores

- Uso de operadores com objetos (sobrecarga de operadores)
 - São mais claros que as chamadas de função para determinadas classes
 - Os operadores são sensíveis ao contexto
- Exemplos
 - <<
 - Operador de inserção de fluxo e de bits de deslocamento para a esquerda
 - +
 - Executa aritmética em múltiplos itens (inteiros, pontos flutuantes etc.) de maneira diferente

Fundamentos de Sobrecarga de Operadores

- Tipos de sobrecarga de operadores
 - Predefinidos (`int`, `char`) ou definidos pelo usuário (classes)
 - É possível usar operadores existentes com tipos definidos pelo usuário
 - Não é possível criar novos operadores

Fundamentos de Sobrecarga de Operadores

- Operadores sobrecarregados
 - Criam uma função para a classe
 - **Função não-estática ou global**
 - Funções estáticas não precisam do objeto da classe, assim, não haveria como definir o contexto
 - Nome da função do operador
 - **Palavra-chave `operator` seguida de um símbolo**
 - Ex.: `operator+` para o operador de adição `+`
 - A sobrecarga de operadores contribui para a extensibilidade do C++
 - **Característica desejável da linguagem**

Fundamentos de Sobrecarga de Operadores

- Uso da sobrecarga de operadores deve ser empregada para o programa ficar mais claro
 - Ao invés de realizar as mesmas operações com funções
- Operadores sobrecarregados devem simular a funcionalidade de suas contrapartes predefinidas
 - Ex.: o operador + deve ser sobrecarregado para realizar adição, não subtração
- Uso excessivo ou inconsistente de sobrecarga de operadores deve ser evitado
 - Programa pode se tornar complexo e difícil de ler

Fundamentos de Sobrecarga de Operadores

- Usando operadores em um objeto de classe
 - Operador é sobrecarregado para essa classe
 - Exceções: Operadores que não precisam ser sobrecarregados para serem usados com objetos de uma classe
 - Operador de atribuição (=)
 - » Atribuição de membro a membro entre objetos
 - Operador de endereço (&)
 - » Retorna o endereço do objeto

Fundamentos de Sobrecarga de Operadores

- Com a sobrecarga é possível contar com uma notação concisa

```
object2 = object1.add( object2 );
```

versus

```
object2 = object1 + object2;
```

Restrições à Sobrecarga de Operadores

- Não é possível mudar...
 - Precedência do operador (ordem de avaliação)
 - Use parênteses para forçar a ordem dos operadores
 - Associatividade (se o operador for aplicado da esquerda para a direita ou da direita para a esquerda)
 - Número de operandos
 - Por exemplo, & é unário e só pode atuar em um único operando
 - Os operadores agem em tipos de dados predefinidos (isto é, não é possível alterar a adição de inteiros)

Restrições à Sobrecarga de Operadores

- Não é possível criar novos operadores
- Os operadores devem ser sobrecarregados explicitamente
 - Sobrecarregar `+ e` = não sobrecarrega `+=`
- O operador ternário (`? :`) não pode ser sobrecarregado
 - Tentar sobrecarregar um operador não sobrecarregável é um erro de sintaxe

Sobrecarga de Operadores

Operadores que podem ser sobrecarregados

+	-	*	/	%	^	&	
~	!	=	<	>	+=	--	*=
/=	%=	^=	&=	=	<<	>>	>>=
<<=	==	!=	<=	>=	&&		++
--	->*	,	->	[]	()	new	delete
new[]	delete[]						

Operadores que não podem ser sobrecarregados

. .* :: ?:

Sobrecarga de Operadores

- São erros:
 - Tentar alterar o número de operandos de um operador via sobrecarga é um erro de compilação
 - Operadores unários devem permanecer unários, assim como operadores binários devem permanecer binários
 - Tentar criar novos operadores via sobrecarga de operadores é um erro de sintaxe
 - Não é possível criar operadores como o `**` para exponenciação sobrecarregando o operador `*`

Sobrecarga de Operadores

- São erros:
 - Tentar modificar a maneira como um operador funciona com objetos de tipos fundamentais
 - O operador + para somar inteiros vai sempre ser usado da mesma maneira
 - Supor que sobrecarregar um operador como + sobrecarrega operadores relacionados como += ou que sobrecarregar == sobrecarrega um operador relacionado como != pode provocar erros
 - Os operadores só podem ser sobrecarregados explicitamente; não há sobrecarga implícita

Fçs Operadoras como Membros de Classe vs. Fçs Globais

- Funções operadoras como funções-membro da classe
 - O objeto da extrema esquerda deve ser da mesma classe que a função operadora
 - Ex.: `Miguel + Campista`
 - Operando único do operador unário é dessa classe
 - Use a palavra-chave `this` para obter implicitamente o argumento do operando esquerdo
 - Os operadores `()`, `[]`, `->` ou qualquer operador de atribuição devem ser sobrecarregados como função-membro de uma classe

Fçs Operadoras como Membros de Classe vs. Fçs Globais

- Funções operadoras como funções globais
 - Podem ter objetos de classes diferentes das do operador
 - **Precisam ser globais para poderem ser usadas por objetos de classes diferentes**
 - Precisam de parâmetros para ambos os operandos
 - Podem ser `friend` para acessar dados `private` ou `protected`

Fçs Operadoras como Membros de Classe vs. Fçs Globais

- Se operando da esquerda precisar ser:
 - Um objeto de uma classe diferente da classe definida
 - ou
 - Um tipo fundamental

Função operadora deve ser global!

- Função global pode ser `friend` se precisar acessar dados `private` ou `protected`
- Funções membro são chamadas quando:
 - Operando da esquerda é da própria classe
 - Só há um operando

Fçs Operadoras como Membros de Classe vs. Fçs Globais

- Operador sobrecarregado <<
 - Operando esquerdo do tipo `ostream &`
 - Como o objeto `cout` em `cout << classObject`
 - De modo semelhante, o operador sobrecarregado >> tem o operador esquerdo de `istream &`
 - Portanto, ambos devem ser **funções globais**
 - São utilizados por **classes diferentes**
 - Se elas não fossem globais elas precisariam de um objeto da classe original à esquerda

Fçs Operadoras como Membros de Classe vs. Fçs Globais

- Operadores comutativos
 - Podem exigir que a função seja global
 - Ex.: Talvez exijam que + seja comutativo
 - Logo, tanto "a + b" quanto "b + a" devem funcionar

E se os operadores fossem de classes diferentes?

Fçs Operadoras como Membros de Classe vs. Fçs Globais

- Suponha que tivéssemos duas classes diferentes
 - O operador sobrecarregado só pode ser uma função-membro quando sua classe está à esquerda
 - `HugeIntClass + long int`
 - Pode ser uma função-membro
 - De outra maneira, é necessária uma função sobrecarregada global
 - `long int + HugeIntClass`
 - Função global pode trocar a ordem dos operandos e chamar a função membro

Fçs Operadoras como Membros de Classe vs. Fçs Globais

- Operadores << e >>
 - Já sobrecarregados para processar cada tipo predefinido
 - Também podem processar uma classe definida pelo usuário
 - Sobrecarga usando funções globais **friend**
- Ex.: Classe `PhoneNumber`
 - Abriga um número de telefone
 - Imprime número formatado automaticamente
 - (123) 456-7890

Primeiro Exemplo de Operadores Sobrecarregados em C++

```
/*
 * Aula 11 - Exemplo 1
 * Arquivo phoneNumberCap11Ex1.h
 * Autor: Miguel Campista
 */
#ifndef PHONE_H
#define PHONE_H

#include <iostream>
#include <string>
#include <iomanip>

using namespace std;

class PhoneNumber {
    friend ostream &operator<<(ostream &, const PhoneNumber &);
    friend istream &operator>>(istream &, PhoneNumber &);

private:
    string areaCode;
    string exchange;
    string line;
};

#endif
```

Primeiro Exemplo de Operadores Sobrecarregados em C++

```
/*  
 * Aula 11 - Exemplo 1  
 * Arquivo phonenumberCap11Ex1.cpp  
 * Autor: Miguel Campista  
 */  
#include "phonenumberCap11Ex1.h"  
  
ostream &operator<<(ostream & output, const PhoneNumber & number) {  
    output << "(" << number.areaCode << ") "  
        << number.exchange << "-" << number.line;  
    return output; // permite cout << a << b << c  
}  
  
istream &operator>>(istream & input, PhoneNumber & number) {  
    input.ignore(); // pula  
    input >> setw (2) >> number.areaCode; // Entrada do cód. de área  
    input.ignore (2); // pula ) e espaço  
    input >> setw (4) >> number.exchange; // Entrada do prefixo  
    input.ignore (); // pula traço (-)  
    input >> setw (4) >> number.line; // Entrada de linha  
    return input; // permite cin >> a >> b >> c;  
}
```

Primeiro Exemplo de Operadores Sobrecarregados em C++

```
/*
 * Aula 11 - Exemplo 1
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include "phonenumberCap11Ex1.h"

int main() {
    PhoneNumber phone;

    cout << "Entre com o telefone na forma (12) 3456-7890:" << endl;

    // cin >> phone invoca operador>> emitindo implicitamente
    // a chamada da função global operator>>(cin, phone)
    cin >> phone;

    // cout << phone invoca operador<< emitindo implicitamente
    // a chamada da função global operator<<(cout, phone)
    cout << "\nTelefone: " << phone << endl;

    return 0;
}
```


Primeiro Exemplo de Operadores Sobrecarregados em C++

```
/*  
 * Aula 11 - Exemplo 1  
 * Arquivo Principal  
 * Autor: Miguel Campista  
 */  
#include "phonenumberCap11Ex1.h"
```

C:\Users\Miguel\Documents\UFRJ\disciplinas\linguagens\projetos\aula11-ex1.exe

```
Entre com o telefone na forma <12> 3456-7890:  
<21> 2233-3322
```

```
Telefone: <21> 2233-3322  
Pressione qualquer tecla para continuar. . . _
```

```
// cout << phone invoca operador<< emitindo implicitamente  
// a chamada da função global operator<<(cout, phone)  
cout << "\nTelefone: " << phone << endl;  
  
return 0;  
}
```

Fçs Operadoras como Membros de Classe vs. Fçs Globais

- Retornar uma referência a partir de uma função operadora << ou >> sobrecarregada é interessante
 - A maioria dos objetos de fluxo são globais ou pelo menos de longa duração
 - Retornar uma referência a uma variável automática ou a um objeto temporário é perigoso, pois cria "referências oscilantes" a objetos não-existentes
- Novas capacidades de entrada/saída para tipos definidos pelo usuário são adicionadas ao C++ sem modificar as classes de entrada/saída da biblioteca-padrão do C++
 - Exemplo de extensão do C++

Sobrecarregando Operadores Unários

- É possível sobrecarregar uma função-membro não-`static` sem nenhum argumento
- **ou**
- Sobrecarregar operadores unários como função global com um argumento
 - O argumento deve ser um objeto de classe ou uma referência a um objeto de classe
- Lembre-se: as funções `static` acessam apenas dados `static`, portanto as funções não podem ser `static`

Sobrecarregando Operadores Unários

- Exemplo a ser apresentado
 - Sobrecarregue ! para verificar se a string está vazia
 - Se for uma função-membro não-`static`, não será necessário nenhum argumento
 - `class String {`
 `public:`
 `bool operator!() const;`
 `...`
 `};`
 - !s torna-se `s.operator!()`
 - Se for uma função global, necessita de um único argumento
 - `bool operator!(const String &)`
 - !s torna-se `operator!(s)`

Sobrecarregando Operadores Binários

- Função-membro não-`static`, um argumento
- Função global, dois argumentos
 - Um argumento deve ser objeto de classe ou referência

Sobrecarregando Operadores Binários

- Exemplo a ser apresentado: Carregando +=
 - Se for uma função-membro não-`static`, necessita de um argumento
 - `class String {`
 - `public:`
 - `const String & operator+=(const String &);`
 - `...`
 - `};`
 - `y += z` torna-se `y.operator+=(z)`
 - Se for uma função global, necessita de dois argumentos
 - `const String &operator+=(String &, const String &);`
 - `y += z` torna-se `operator+=(y, z)`

Estudo de Caso: Classe Array

- Arrays baseados em ponteiro no C++
 - Não há verificação de intervalo
 - Não podem ser comparados de maneira significativa com ==
 - Não há atribuição de array (os nomes de array são ponteiros const)
 - Se for passado um array a uma função, o tamanho deve ser passado como um argumento separado

Estudo de Caso: Classe Array

- Exemplo: Implemente uma classe `Array` com:
 - Verificação de intervalo
 - Atribuição de array
 - Arrays que conhecem seu próprio tamanho
 - Entrada/Saída de arrays inteiros com `<< e >>`
 - Comparação entre arrays com `== e !=`

Estudo de Caso: Classe Array

- Construtor de cópia
 - Usado quando se precisa de uma cópia de um objeto:
 - É chamado sempre que um objeto é passado por valor para uma função
 - Inicializa um objeto com uma cópia de outro do mesmo tipo
 - `Array newArray (oldArray);`
ou
`Array newArray = oldArray (ambos são idênticos)`
» `newArray` é uma cópia de `oldArray`

Estudo de Caso: Classe Array

- Construtor de cópia da classe `Array`
 - Evita problemas com dados manipulados por referências
 - Para isso, utiliza referência
 - Protótipo para a classe `Array`
 - `Array (const Array &);`
 - Deve obter referência
 - Do contrário, o argumento será passado por valor por padrão...

Segundo Ex. de Operadores Sobrecarregados em C++

```
/*
 * Aula 11 - Exemplo 2
 * Arquivo arrayCap11Ex2.h
 * Autor: Miguel Campista
 */
#ifndef ARRAY_H
#define ARRAY_H

#include <iostream>
#include <cstdlib>
#include <iomanip>

using namespace std;

class Array {
    friend ostream &operator<<(ostream &, const Array &);
    friend istream &operator>>(istream &, Array &);

public:
    Array (int = 10); // Construtor padrão
    Array (const Array &); // Construtor de cópia
    ~Array (); // destrutor
    int getSize () const;

    const Array &operator=(const Array &); // Operador de atribuição
    bool operator==(const Array &) const; // Operador de igualdade
    // Operador de desigualdade
    bool operator!=(const Array &right) const;
```

Segundo Ex. de Operadores Sobrecarregados em C++

```
        // Operador subscripto de objetos não-const
        // retorna lvalue modificável
        int &operator[] (int);

        // Operador subscripto de objetos const retorna rvalue
        int operator[] (int) const;
private:
        int size; // Tamanho do array baseado em ponteiro
        int *ptr; // Ponteiro para o primeiro elemento do array
                // baseado em ponteiro
};

#endif
```

Segundo Ex. de Operadores Sobrecarregados em C++

```
/*
 * Aula 11 - Exemplo 2
 * Arquivo arrayCap11Ex2.cpp
 * Autor: Miguel Campista
 */

#include "arrayCap11Ex2.h"

Array::Array (int arraySize) {
    size = (arraySize > 0 ? arraySize : 10); // valida arraySize
    ptr = new int [size]; // Cria espaço para array baseado em ponteiro

    for (int i = 0; i < size; i++)
        ptr [i] = 0; // Configura elemento do array baseado em ponteiro
}

// Copia o construtor da classe Array
// Deve receber uma referência para impedir a recursão infinita
Array::Array (const Array &arrayToCopy) : size (arrayToCopy.size) {
    ptr = new int [size]; // Cria espaço para array baseado em ponteiro

    for (int i = 0; i < size; i++)
        ptr [i] = arrayToCopy.ptr [i]; // Copia para o objeto
}

// Destrutor para a classe Array
Array::~Array () {
    delete [] ptr; // Libera espaço do array baseado em ponteiro
}
```

Segundo Ex. de Operadores Sobrecarregados em C++

```
// Retorna o número de elementos do Array
int Array::getSize () const { return size; }

// Operador de atribuição sobrecarregado
// retorno const evita: (a1 = a2) = a3
const Array &Array::operator=(const Array &right) {
    if (&right != this) {
        // Para arrays de tamanhos diferentes,
        // desaloca array do lado esquerdo original, então
        // aloca o novo array à esquerda
        if (size != right.size) {
            delete [] ptr; // Libera espaço
            size = right.size; // Redimensiona esse objeto
            ptr = new int [size]; // Cria espaço para cópia do array
        }
        for (int i = 0; i < size; i++)
            ptr [i] = right.ptr [i]; // Copia o array para o objeto
    }
    return *this;
}

bool Array::operator==(const Array &right) const {
    if (size != right.size)
        return false; // Arrays com tamanhos diferentes

    for (int i = 0; i < size; i++) {
        if (ptr [i] != right.ptr [i])
            return false; // Conteúdo do array não é igual
    }

    return true; // Arrays são iguais
}
```

Segundo Ex. de Operadores Sobrecarregados em C++

```
// Operador de desigualdade; retorna o oposto do operador ==
bool Array::operator!=(const Array &right) const {
    return ! (*this == right); // Invoca Array::operator ==
}

// Operador de subscripto sobrecarregado para Array não-const
// Retorno de referência cria um lvalue modificável
int &Array::operator[](int index) {
    // Verifica erro de índice fora do array
    if (index < 0 || index >= size) {
        cerr << "\nError: Subscript " << index
            << " fora do intervalo" << endl;
        exit (1);
    }

    return ptr [index]; // Retorno da referência
}

// Operador de subscripto sobrecarregado para Array const
// Retorno de referência const cria um rvalue
int Array::operator[](int index) const {
    // Verifica erro de índice fora do array
    if (index < 0 || index >= size) {
        cerr << "\nError: Subscript " << index
            << " fora do intervalo" << endl;
        exit (1);
    }

    return ptr [index]; // Retorno de cópia
}
```

Segundo Ex. de Operadores Sobrecarregados em C++

```
// Operador de entrada sobrecarregado para classe Array
// entrada de valores para o Array inteiro
istream &operator>>(istream &input, Array &a) {
    for (int i = 0; i < a.size; i++)
        input >> a.ptr [i];

    return input; // Permite cin >> x >> y;
}

// Operador de saída sobrecarregado para classe Array
ostream &operator<<(ostream &output, const Array &a) {
    int i;
    // Gera saída do array baseado em ptr private
    for (i = 0; i < a.size; i++) {
        output << setw (12) << a.ptr [i];

        if ((i + 1) % 4 == 0) // 4 números por linha de saída
            output << endl;
    }

    if (i % 4 != 0) // Termina a última linha de saída
        output << endl;

    return output; // Permite cout << x << y;
}
```


Segundo Ex. de Operadores Sobrecarregados em C++

```
/*
 * Aula 11 - Exemplo 2
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include "arrayCap11Ex2.h"

int main() {
    Array integers1 (7); // Array de sete elementos
    Array integers2; // Array de 10 elementos por padrão

    // Imprime o tamanho e o conteúdo de integers1
    cout << "Tamanho do array integers1 eh: "
         << integers1.getSize ()
         << "\nArray antes da inicializacao:\n" << integers1;

    // Imprime o tamanho e o conteúdo de integers2
    cout << "Tamanho do array integers2 eh: "
         << integers2.getSize ()
         << "\nArray antes da inicializacao:\n" << integers2;

    // Insere e imprime integers1 e integers2
    cout << "\nEntre com 17 inteiros:" << endl;
    cin >> integers1 >> integers2;

    cout << "Depois da entrada os arrays contem:\n"
         << "integers1:\n" << integers1
         << "integers2:\n" << integers2;
```

Segundo Ex. de Operadores Sobrecarregados em C++

```
// Utiliza o operador de desigualdade (!=) sobrecarregado
cout << "\nAvaliando: integers1 != integers2" << endl;

if (integers1 != integers2)
    cout << "integers1 e integers2 nao sao iguais" << endl;

// Cria array integers3 usando integers1 como um inicializador
// Imprime tamanho e conteúdo
Array integers3 (integers1); // Invoca o construtor de cópia

cout << "Tamanho do array integers3 eh: "
    << integers3.getSize ()
    << "\nArray antes da inicializacao:\n" << integers3;

// Utiliza operador atribuição (=) sobrecarregado
cout << "\nAtribuindo integers2 para integers1:" << endl;
integers1 = integers2; // Note que o array alvo é menor

cout << "\nintegers1:\n" << integers1
    << "integers2:\n" << integers2;

// Utiliza operador de igualdade sobrecarregado (==) sobrecarregado
cout << "\nAvaliando: integers1 == integers2" << endl;

cout << "\nAvaliando: integers1 == integers2" << endl;

if (integers1 == integers2)
    cout << "integers1 e integers2 sao iguais" << endl;
```

Segundo Ex. de Operadores Sobrecarregados em C++

```
// Utiliza operador de subscripto sobrecarregado para criar rvalue
cout << "\nintegers1 [5] eh " << integers1 [5];

// Utiliza operador de subscripto sobrecarregado para criar lvalue
cout << "\n\nAtribuindo 1000 para integers1 [5]" << endl;
integers1 [5] = 1000;
cout << "integers1:\n" << integers1;

// Tentativa de utilizar subscripto fora do intervalo
cout << "\nTentativa de atribuir 1000 para integers1 [5]" << endl;
integers1 [15] = 1000; // Erro: fora do intervalo

return 0;
}
```

Segundo Ex. de Operadores Sobrecarregados em C++

```
C:\Windows\system32\cmd.exe
C:\Users\Miguel\Documents\UFRJ\disciplinas\linguagens\projetos>aula11-ex2.exe
Tamanho do array integers1 eh: 7
Array antes da inicializacao:
    0      0      0      0
    0      0      0
Tamanho do array integers2 eh: 10
Array antes da inicializacao:
    0      0      0      0
    0      0      0
    0      0
Entre com 17 inteiros:
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
Depois da entrada os arrays contem:
integers1:
    1      2      3      4
    5      6      7
integers2:
    8      9      10     11
    12     13     14     15
    16     17
Avaliando: integers1 != integers2
integers1 e integers2 nao sao iguais
Tamanho do array integers3 eh: 7
Array antes da inicializacao:
    1      2      3      4
    5      6      7
Atribuindo integers2 para integers1:
integers1:
    8      9      10     11
    12     13     14     15
    16     17
integers2:
    8      9      10     11
    12     13     14     15
    16     17
Avaliando: integers1 == integers2
```

Segundo Ex. de Operadores Sobrecarregados em C++

```
// Utiliza operador de subscrito sobrecarregado para criar rvalue  
cout << "\nintegers1 [5] eh " << integers1 [5];  
  
// Utiliza operador de subscrito sobrecarregado para criar lvalue
```

```
Avaliando: integers1 == integers2  
integers1 e integers2 sao iguais  
  
integers1 [5] eh 13  
  
Atribuindo 1000 para integers1 [5]  
integers1:  
      8      9      10      11  
     12     1000     14     15  
     16      17  
  
Tentativa de atribuir 1000 para integers1 [5]  
Error: Subscript 15 fora do intervalo  
C:\Users\Miguel\Documents\UFRJ\disciplinas\linguagens\projetos>_
```

Construtor de Cópia

- O argumento para um construtor de cópia deve ser uma referência `const` para que o processo de cópia não altere o objeto original
- Observe que um construtor de cópia deve receber seu argumento por referência, não por valor
 - Do contrário, a chamada do construtor de cópia provoca recursão infinita (um erro de lógica fatal)
 - Receber um objeto por valor requer que o construtor de cópia faça uma cópia do objeto de argumento
 - Logo, toda vez que uma cópia por valor for realizada, um construtor de cópia é invocado formando o loop infinito

Construtor de Cópia

- Se o construtor de cópia simplesmente copiasse o ponteiro no objeto de origem para o ponteiro do objeto-alvo
 - Então ambos os objetos apontariam para a mesma memória dinamicamente alocada
 - Logo, o primeiro destrutor a executar excluiria a memória dinamicamente alocada, e o `ptr` do outro objeto seria indefinido, uma situação chamada de "ponteiro oscilante"
 - **Provoca erro em tempo de execução quando o ponteiro é utilizado**

Construtor de Cópia

- Construtor de cópia, um destrutor e um operador de atribuição sobrecarregado
 - Normalmente são fornecidos como um grupo para qualquer classe que aloca memória dinamicamente
- Erro de lógica
 - Não fornecer um operador de atribuição sobrecarregado e um construtor de cópia para uma classe quando os objetos dessa classe contêm ponteiros para memória alocada dinamicamente

Construtor de Cópia

- É possível impedir que um objeto de uma classe seja atribuído a outro
 - Isso é feito declarando o operador de atribuição como um membro `private` da classe
- É possível impedir que objetos de classe sejam copiados
 - Torne ambos, o operador de atribuição sobrecarregado e o construtor de cópia dessa classe, `private`

Conversão entre Tipos

- Conversão
 - Geralmente converte-se inteiros em pontos flutuantes etc.
 - Talvez seja necessário converter entre tipos definidos pelo usuário
- Operador de conversão (operador de coerção)
 - Converta
 - Uma classe em outra
 - Uma classe em um tipo predefinido (`int`, `char` etc.)
 - Deve ser uma função-membro não-`static`
 - Não especifique um tipo de retorno
 - Retorne um tipo implicitamente ao tipo para o qual está convertendo

Conversão entre Tipos

- Operador de (operador de coerção)
 - Exemplo
 - Protótipo
 - `A::operator char * () const;`
 - **Converte objetos da classe A em um `char *` temporário**
 - `static_cast< char * >(s)` chama `s.operator char *()`
 - Além disso,
 - `A::operator int () const;`
 - » **Converte objeto da classe A em um inteiro**
 - `A::operator OtherClass () const;`
 - » **Converte objeto da classe A em objeto da classe OtherClass**

Conversão entre Tipos

- A conversão evita a necessidade de sobrecarga
 - Suponha que uma classe `String` possa ser convertida em `char *`
 - `cout << s; // s é uma String`
 - O compilador converte implicitamente `s` em `char *` como saída
 - Não há necessidade de sobrecarregar `<<`

Estudo de Caso: Classe `String`

- Construção e manipulação da classe `String`
 - Semelhante à classe `string` da biblioteca-padrão
- Construtor de conversão
 - Qualquer construtor de um único argumento pode ser visto como um construtor de conversão
 - Transforma objetos de outros tipos em objetos de classe
 - Ex.: `String s1("happy"); //cria uma String de um char *`
- Sobrecarga de um operador de chamada de função (`()`)
 - Funções podem aceitar listas de parâmetros arbitrariamente longas e complexas

Terceiro Ex. de Operadores Sobrecarregados em C++

```
/*
 * Aula 11 - Exercício 3
 * Arquivo sobrecargaCap11Ex3.h
 * Autor: Miguel Campista
 */
#ifndef STRING_H
#define STRING_H

#include <iostream>
#include <iomanip>
#include <cstring>
#include <cstdlib>

using namespace std;

class String {
    friend ostream &operator<<(ostream &, const String &);
    friend istream &operator>>(istream &, String &);

public:
    String (const char * = ""); // Construtor de conversão padrão
    String (const String &); // Construtor de cópia
    ~String (); // Destrutor

    const String &operator=(const String &); // Operador de atribuição
    const String &operator+=(const String &); // Operador de concatenação

    bool operator!() const; // A String está vazia?
    bool operator==(const String &) const; // teste s1 == s2
    bool operator<(const String &) const; // teste s1 < s2
};
```

Terceiro Ex. de Operadores Sobrecarregados em C++

```
// teste s1 != s2
bool operator!=( const String &right ) const {
    return !( *this == right );
}

// teste s1 > s2
bool operator>(const String &right) const {
    return right < *this;
}

// teste s1 <= s2
bool operator<=(const String &right) const {
    return !(right < *this);
}

// teste s1 >= s2
bool operator>=(const String &right) const {
    return !( *this < right );
}

char &operator[](int); // operador de subscripto
                        // (lvalue modificável)
char operator[](int) const; // operador de subscripto
                        // (rvalue)
String operator() (int, int = 0) const; // retorna uma substring
int getLength () const; // retorna o comprimento da string
private:
    int length; // comprimento da string (sem contar o terminador nulo)
    char *sPtr; // ponteiro para iniciar string baseada em ponteiro

    void setString (const char *);
};

#endif
```

```

/*
 * Aula 11 - Exercicio 3
 * Arquivo sobrecargaCap11Ex3.cpp
 * Autor: Miguel Campista
 */
#include "sobrecargaCap11Ex3.h"

String::String (const char *s) : length ((s != 0) ? strlen (s) : 0) {
    cout << "Construtor de conversao e (padrao): " << s << endl;
    setString (s);
}

String::String (const String &copy) : length (copy.length) {
    cout << "Construtor de copia: " << copy.sPtr << endl;
    setString (copy.sPtr);
}

String::~String () {
    cout << "Destructor: " << sPtr << endl;
    delete [] sPtr;
}

const String &String::operator=(const String &right) {
    cout << "operator= chamado" << endl;

    if (&right != this) {
        delete [] sPtr;
        length = right.length;
        setString (right.sPtr);
    } else {
        cout << "Tentativa de atribuicao de uma String para ela mesma"
            << endl;
    }
    return *this;
}

```



```
const String &String::operator += (const String &right) {
    size_t newLength = length + right.length;
    char *tempPtr = new char [newLength + 1];

    strcpy (tempPtr, sPtr);
    strcpy (tempPtr + length, right.sPtr);

    delete [] sPtr;
    sPtr = tempPtr;
    length = newLength;
    return *this;
}

bool String::operator!() const {
    return length == 0;
}

bool String::operator==(const String &right) const {
    return strcmp (sPtr, right.sPtr) == 0;
}

bool String::operator<(const String &right) const {
    return strcmp (sPtr, right.sPtr) < 0;
}

char &String::operator[](int subscript) {
    if (subscript < 0 || subscript >= length) {
        cerr << "Error: Subscript " << subscript
            << " out of range" << endl;
        exit (1);
    }
    return sPtr [subscript];
}
```

Terceiro Ex. de Operadores Sobrecarregados em C++

```
char String::operator[](int subscript) const {
    if (subscript < 0 || subscript >= length) {
        cerr << "Error: Subscript " << subscript
            << " out of range" << endl;
        exit (1);
    }
    return sPtr [subscript];
}

String String::operator()(int index, int subLength) const {
    if (index < 0 || index >= length || subLength < 0)
        return "";

    int len;

    if (subLength == 0 || (index + subLength > length))
        len = length - index;
    else
        len = subLength;

    char * tempPtr = new char [len + 1];

    strncpy (tempPtr, &sPtr [index], len);
    tempPtr [len] = '\0';

    String tempString (tempPtr);
    delete [] tempPtr;
    return tempString;
}
```

Terceiro Ex. de Operadores Sobrecarregados em C++

```
int String::getLength () const {
    return length;
}

void String::setString (const char * string2) {
    sPtr = new char [length + 1];

    if (string2 != 0)
        strcpy (sPtr, string2);
    else
        sPtr [0] = '\0';
}

ostream &operator<<(ostream &output, const String &s) {
    output << s.sPtr;
    return output;
}

istream &operator>>(istream &input, String &s) {
    char temp [100];
    input >> setw(100) >> temp;
    s = temp;
    return input;
}
```

Terceiro Ex. de Operadores Sobrecarregados em C++

```
/*
 * Aula 11 - Exercicio 3
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>
#include "sobrecargaCap11Ex3.h"

int main() {
    String s1 ("Feliz");
    String s2 (" aniversario");
    String s3;

    // testa igualdade e relacionais sobrecarregados
    cout << "s1 eh \"></pre></div>
```

Terceiro Ex. de Operadores Sobrecarregados em C++

```
// testa o operador de concatenação de String sobrecarregado
cout << "\n\ ns1 += s2 permite s1 = ";
s1 += s2; // testa concatenação sobrecarregada
cout << s1;

// testa construtor de conversão
cout << "\n\ ns1 += \" para vc\" permite" << endl;
s1 += " para vc"; // testa construtor de conversão
cout << "s1 = " << s1 << "\n\n";

// testa o operador de chamada de função sobrecarregado ()
// para string
cout << "O subscrito de s1 começando na\n"
    << "posicao 0 para 14 caracteres, s1(0, 14), eh:\n";
cout << s1( 0, 14 ) << "\n\n";

// testa a opção de substring "to-end-of-String"
cout << "O subscrito de s1 começando na\n"
    << "posicao 15, s1(15), eh:\n";
cout << s1( 15 ) << "\n\n";

// testa construtor de cópia
String *s4Ptr = new String( s1 );
cout << "\n*s4Ptr = " << *s4Ptr << "\n\n";

// testa operador de atribuição (=) com auto-atribuição
cout << "atribuindo *s4Ptr to *s4Ptr" << endl;
*s4Ptr = *s4Ptr; // testa atribuição sobrecarregada
cout << "*s4Ptr = " << *s4Ptr << endl;

// testa destrutor
delete s4Ptr;
```

Terceiro Ex. de Operadores Sobrecarregados em C++

```
// testa usando operador de subscripto lvalue para criar lvalue modificável
s1[ 0 ] = 'H';
s1[ 6 ] = 'B';
cout << "\ns1 depois s1[0] = 'H' e s1[6] = 'B' eh: "
      << s1 << "\n\n";

// testa subscripto fora de intervalo
cout << "Tentativa de atribuir 'd' para s1[30] permite:" << endl;
s1[ 30 ] = 'd'; // ERROR: subscripto fora de intervalo

return 0;
}
```

Terceiro Ex. de Operadores Sobrecarregados em C++

```
C:\Users\Miguel\Documents\UFRJ\disciplinas\linguagens\projetos>aula11-ex3.exe
Construtor de conversao e <padrao>: Feliz
Construtor de conversao e <padrao>: aniversario
Construtor de conversao e <padrao>:
s1 eh "Feliz"; s2 eh " aniversario"; s3 eh ""

Os resultados da comparacao s2 e s1:
s2 == s1 permite false
s2 != s1 permite true
s2 > s1 permite false
s2 < s1 permite true
s2 >= s1 permite false
s2 <= s1 permite true

Testando !s3:
s3 esta vazio; atribuindo s1 a s3;
operator= chamado
s3 eh "Feliz"

s1 += s2 permite s1 = Feliz aniversario

s1 += " para vc" permite
Construtor de conversao e <padrao>: para vc
Destrutor: para vc
s1 = Feliz aniversario para vc

O subscrito de s1 comecando na
posicao 0 para 14 caracteres, s1<0, 14>, eh:
Construtor de conversao e <padrao>: Feliz aniversa
Construtor de copia: Feliz aniversa
Destrutor: Feliz aniversa
Feliz aniversa

Destrutor: Feliz aniversa
O subscrito de s1 comecando na
posicao 15, s1<15>, eh:
Construtor de conversao e <padrao>: io para vc
Construtor de copia: io para vc
Destrutor: io para vc
io para vc

Destrutor: io para vc
Construtor de copia: Feliz aniversario para vc

*s4Ptr = Feliz aniversario para vc
```

cável

Terceiro Ex. de Operadores Sobrecarregados em C++

```
atribuindo *s4Ptr to *s4Ptr
operator= chamado
Tentativa de atribuicao de uma String para ela mesma
*s4Ptr = Feliz aniversario para vc
Destrutor: Feliz aniversario para vc

s1 depois s1[0] = 'H' e s1[6] = 'B' eh: Heliz Bniversario para vc

Tentativa de atribuir 'd' para s1[30] permite:
Error: Subscript 30 out of range

C:\Users\Miguel\Documents\UFRJ\disciplinas\linguagens\projetos>_
// testa subscripto fora de intervalo
cout << "Tentativa de atribuir 'd' para s1[30] permite:" << endl;
s1[ 30 ] = 'd'; // ERROR: subscripto fora de intervalo

return 0;
}
```


Estudo de Caso: Classe String

- Construtor de conversão é utilizado para realizar uma conversão implícita
 - O C++ pode aplicar apenas uma chamada de construtor de conversão implícita para tentar atender às necessidades de outro operador sobrecarregado
 - Uma única conversão definida pelo usuário
 - O compilador não consegue realizar uma série de conversões implícitas
 - O compilador não atenderá às necessidades de um operador sobrecarregado para realizar uma série de conversões implícitas definidas pelo usuário

Estudo de Caso: Classe `String`

- Sobrecarregar o operador de concatenação `+=` com uma versão adicional que aceita um único argumento do tipo `const char *` é mais eficiente do que ter apenas uma versão que aceita um argumento `String`
 - Sem a versão `const char *` do operador `+=`, um argumento `const char *` seria primeiro convertido em um objeto `String` com o construtor de conversão da classe `String`
 - Então o operador `+=` que recebe um argumento `String` seria chamado para realizar a concatenação

Estudo de Caso: Classe `String`

- Conversões implícitas com operadores sobrecarregados ao invés de sobrecarregar operadores para muitos tipos de operandos
 - Exige uma quantidade menor de código, o que torna uma classe mais fácil de modificar, manter e depurar
- Implementação de funções-membro com funções-membro anteriormente definidas
 - Reduz a quantidade de código que deve ser escrita e mantida por reutilização de código
- É perigoso retornar uma referência não-const `char` de um operador de subscrito sobrecarregado em uma classe `String`
 - O cliente poderia utilizar essa referência para inserir um nulo (`'\0'`) em qualquer lugar da string

Exemplo 1

- Escreva um programa que defina uma classe `Agenda` que contém um `vector` para armazenar os contatos. O objetivo é, na função principal, concatenar os contatos de duas agendas em uma só utilizando operadores sobrecarregados. A concatenação deve usar o operador `+`.

Exemplo 1

```
#include <iostream>
#include "agendaOperadorSobrecarregado.h"

int main () {
    Agenda a1 (10);
    Agenda a2 (10);
    string n;

    cout << "Entre com os nomes da agenda a1..." << endl;
    for (int i = 0; i < 3; i++) {
        cout << "Entre com o nome: ";
        getline (cin, n);
        if (a1.insereContato (n) == -1)
            break;
    }

    cout << endl;
    cout << "Entre com os nomes da agenda a2..." << endl;
    for (int i = 0; i < 3; i++) {
        cout << "Entre com o nome: ";
        getline (cin, n);
        if (a2.insereContato (n) == -1)
            break;
    }

    cout << endl;
    cout << "Agenda 1:\n" << a1 << "\nAgenda 2\n" << a2;
```

Exemplo 1

```
// Concatenação das agendas...
// Operação poderia ser realizada apenas com a sentença a1 + a2
// Fazendo a acumulação, após a soma a1 = a1
// A sobrecarga do = não realiza nenhuma operação.
a1 = a1 + a2;

cout << endl;
cout << "Agenda Concatenada:\n" << a1 << endl;

// Chamada do construtor
Agenda a3;
// Uso do operador = sobrecarregado
// a3 poderia também ser inicializado utilizando o construtor de cópia
// Para isso, ele seria inicializado como Agenda a3 = a1;
// Ou ainda Agenda a3 (a1);
a3 = a1;

cout << "Agenda a3:\n" << a3 << endl;

return 0;
}
```

Exemplo 1

```
#include <iostream>
#include <vector>
#include <string>
#include <iterator>

#ifndef AGENDA_H
#define AGENDA_H

using namespace std;

class Agenda {
    friend ostream &operator<< (ostream &, Agenda &);
public:
    Agenda (unsigned = 1);
    Agenda (const Agenda &);
    ~Agenda ();

    int insereContato(string);
    string getContato (int);
    int getSize ();

    Agenda &operator+(Agenda &);
    Agenda &operator=(Agenda &);

private:
    static int numAgendas;
    int id;
    unsigned maxContatos;
    vector <string> v;
};

#endif
```

Exemplo 1

```
#include "agendaOperadorSobrecarregado.h"

int Agenda::numAgendas = 0;

void debug (vector <string> &v) {
    vector<string>::iterator it = v.begin ();
    while (it != v.end ()) {
        cout << "Nome: " << *it << endl;
        it++;
    }
}

Agenda::Agenda (unsigned s) : id (++numAgendas), maxContatos (s) {}

Agenda::Agenda (const Agenda &a) : id (++numAgendas),
    maxContatos (a.maxContatos),
    v (a.v) { cout << "Construtor de cópia!\n"; }

Agenda::~Agenda () {
    cout << "No destrutor do id " << id << "..." << endl;
}
```


Exemplo 1

```
int Agenda::insereContato (string n) {
    vector<string>::iterator it;

    if (v.size () >= maxContatos) {
        cout << "Agenda lotada!" << endl;
        return -1;
    }

    if (!v.size ()) {
        v.push_back (n);
        return 0;
    } else {
        it = v.begin();

        if (n > *(--v.end())) {
            v.push_back (n);
            return 0;
        } else {
            while (n > *it)
                it++;

            if (n != *it) {
                v.insert (it, n);
                return 0;
            } else {
                cout << "Nome já existe. ";
                cout << "Nenhuma operação foi realizada." << endl;
                return 1;
            }
        }
    }
}
```

Exemplo 1

```
string Agenda::getContato (int idx) {
    return v.at (idx);
}

int Agenda::getSize () {
    return v.size ();
}

Agenda &Agenda::operator+(Agenda &a) {
    unsigned i = 0;
    while (i < a.v.size ()) {
        this->insereContato (a.getContato (i));
        i++;
    }
    return *this;
}

Agenda &Agenda::operator=(Agenda &a) {
    if (this != &a) {
        maxContatos = a.maxContatos;
        v = a.v;
    } else
        cout << "\nTermos eram iguais..." << endl;

    return *this;
}
```

Exemplo 1

```
ostream &operator<< (ostream &output, Agenda &a) {  
    unsigned int i = 0;  
    while (i < a.v.size ()) {  
        output << a.v.at (i) << endl;  
        i++;  
    }  
    return output;  
}
```

Sobrecarregando ++ e --

- Operadores de incremento/decremento podem ser sobrecarregados
 - Suponha que queiramos adicionar 1 a um objeto `Date`, `d1`
 - Protótipo (função-membro)
 - `Date &operator++()`;
 - `++d1 torna-se d1.operator++()`
 - Protótipo (função global)
 - `Date &operator++(Date &);`
 - `++d1 torna-se operator++(d1)`

Sobrecarregando ++ e --

- Para distinguir entre incremento prefixado e pós-fixado
 - O incremento pós-fixado tem um parâmetro fictício por convenção para o compilador identificá-lo
 - Um `int` com valor 0
 - Protótipo (função-membro)
 - `Date operator++(int);`
 - `d1++ torna-se d1.operator++(0)`
 - Protótipo (função global)
 - `Date operator++(Date &, int);`
 - `d1++ torna-se operator++(d1, 0)`

Sobrecarregando ++ e --

- Valores de retorno
 - Incremento prefixado
 - Retorna por referência (Date &) com o novo valor
 - *lvalue* (pode ser atribuído)
 - Incremento pós-fixado
 - Retorna por valor
 - Retorna um objeto temporário com um valor antigo já que o atual já foi incrementado
 - *rvalue* (não pode estar no lado esquerdo da atribuição)
- Tudo isso também se aplica aos operadores de decremento

Sobrecarregando ++ e --

- O objeto extra que é criado pelo operador de incremento (ou decremento) pós-fixado pode provocar um problema de desempenho significativo devido ao retorno da função ser feito por valor
 - Especialmente quando o operador é utilizado em um loop já que múltiplas cópias por valor podem ser necessárias
 - Por essa razão, só se deve utilizar o operador de incremento (ou decremento) pós-fixado quando a lógica do programa exigir pós-incremento (ou pós-decremento)

Estudo de caso: uma classe Date

- Exemplo de classe Date
 - Operador de incremento sobrecarregado
 - Muda dia, mês e ano
 - Operador sobrecarregado +=
 - Função para testar anos bissextos
 - Função para determinar se o dia é o último do mês

Quarto Ex. de Operadores Sobrecarregados em C++

```
/*
 * Aula 11 - Exemplo 4
 * Arquivo dateCap11Ex4.h
 * Autor: Miguel Campista
 */
#ifndef DATE_H
#define DATE_H

#include <iostream>

using namespace std;

class Date {
    friend ostream &operator<<( ostream &, const Date & );
public:
    Date( int m = 1, int d = 1, int y = 1900 );
    void setDate( int, int, int );
    Date &operator++();
    Date operator++( int );
    const Date &operator+=( int );
    bool leapYear( int ) const;
    bool endOfMonth( int ) const;
private:
    int month;
    int day;
    int year;

    static const int days[];
    void helpIncrement();
};

#endif
```

Quarto Ex. de Operadores Sobrecarregados em C++

```
/*
 * Aula 11 - Exemplo 4
 * Arquivo dateCap11Ex4.cpp
 * Autor: Miguel Campista
 */
#include "dateCap11Ex4.h"

// inicialização de membro estático em escopo de arquivo
const int Date::days[] =
    { 0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };

Date::Date( int m, int d, int y ) {
    setDate( m, d, y );
}

// atribuição de mês, dia e ano
void Date::setDate( int mm, int dd, int yy ) {
    month = ( mm >= 1 && mm <= 12 ) ? mm : 1;
    year = ( yy >= 1900 && yy <= 2100 ) ? yy : 1900;

    // Testa ano bissexto
    if ( month == 2 && leapYear( year ) )
        day = ( dd >= 1 && dd <= 29 ) ? dd : 1;
    else
        day = ( dd >= 1 && dd <= days[ month ] ) ? dd : 1;
}

// Operador de incremento prefixado sobrecarregado
Date &Date::operator++() {
    helpIncrement();
    return *this;
}
```

Quarto Ex. de Operadores Sobrecarregados em C++

```
Date Date::operator++( int ) {
    Date temp = *this;
    helpIncrement();

    return temp;
}

const Date &Date::operator+=( int additionalDays ) {
    for ( int i = 0; i < additionalDays; i++ )
        helpIncrement();

    return *this; // permite cascadeamento
}

bool Date::leapYear( int testYear ) const {
    if ( testYear % 400 == 0 ||
        ( testYear % 100 != 0 && testYear % 4 == 0 ) )
        return true;
    else
        return false;
}

bool Date::endOfMonth( int testDay ) const {
    if ( month == 2 && leapYear( year ) )
        return testDay == 29;
    else
        return testDay == days[ month ];
}
```

Quarto Ex. de Operadores Sobrecarregados em C++

```
void Date::helpIncrement() {
    if ( !endOfMonth( day ) )
        day++;
    else if ( month < 12 ) {
        month++;
        day = 1;
    } else {
        year++;
        month = 1;
        day = 1;
    }
}

ostream &operator<<( ostream &output, const Date &d )
{
    static char *monthName[ 13 ] = { "", "January", "February",
        "March", "April", "May", "June", "July", "August",
        "September", "October", "November", "December" };
    output << monthName[ d.month ] << ' ' << d.day << ", " << d.year;
    return output;
}
```

Quarto Ex. de Operadores Sobrecarregados em C++

```
/*
 * Aula 11 - Exemplo 4
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include "dateCap11Ex4.h"

int main() {
    Date d1;
    Date d2( 12, 27, 1992 );
    Date d3( 0, 99, 8045 );

    cout << "d1 eh " << d1 << "\nd2 eh " << d2 << "\nd3 eh " << d3;
    cout << "\nd2 += 7 eh " << ( d2 += 7 );

    d3.setDate( 2, 28, 1992 );
    cout << "\n\n d3 eh " << d3;
    cout << "\n++d3 eh " << ++d3 << " (anos bissextos permite ate o dia 29)";

    Date d4( 7, 13, 2002 );

    cout << "\n\nTestando o operador de incremento prefixado:\n"
        << " d4 eh " << d4 << endl;
    cout << "++d4 eh " << ++d4 << endl;
    cout << " d4 eh " << d4;

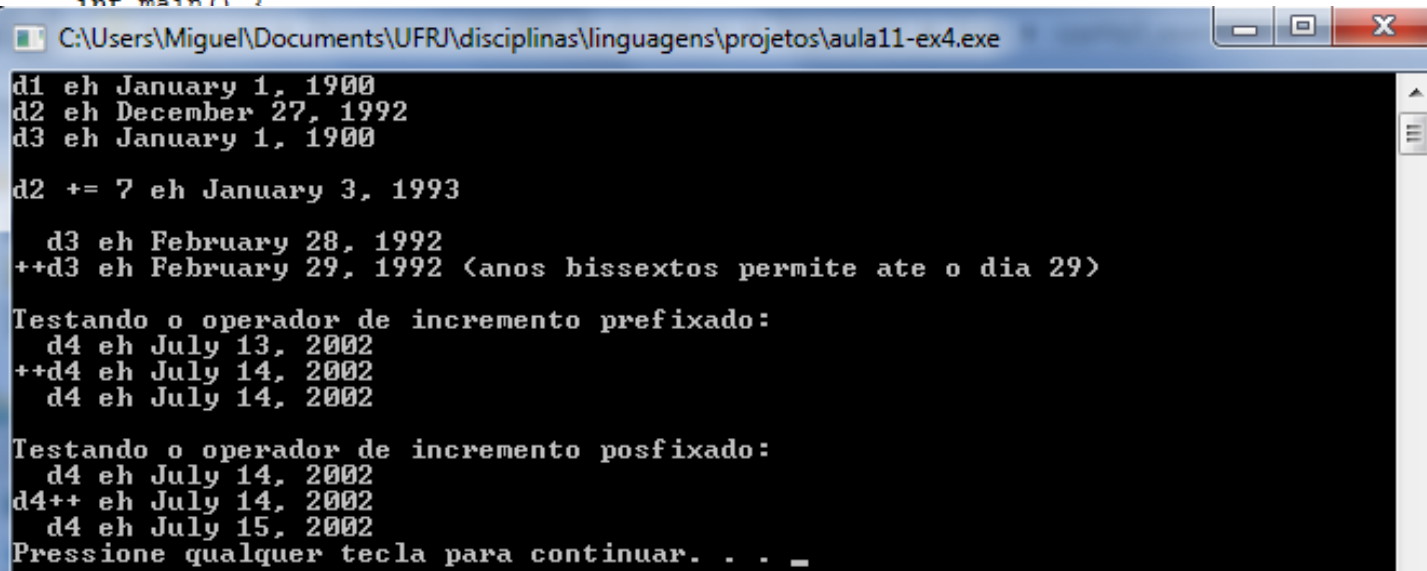
    cout << "\n\nTestando o operador de incremento posfixado:\n"
        << " d4 eh " << d4 << endl;
    cout << "d4++ eh " << d4++ << endl;
    cout << " d4 eh " << d4 << endl;

    return 0;
}
```

Quarto Ex. de Operadores Sobrecarregados em C++

```
/*  
 * Aula 11 - Exemplo 4  
 * Arquivo Principal  
 * Autor: Miguel Campista  
 */  
#include "dateCap11Ex4.h"
```

```
int main() {
```



```
C:\Users\Miguel\Documents\UFRJ\disciplinas\linguagens\projetos\aula11-ex4.exe  
d1 eh January 1, 1900  
d2 eh December 27, 1992  
d3 eh January 1, 1900  
d2 += 7 eh January 3, 1993  
d3 eh February 28, 1992  
++d3 eh February 29, 1992 (anos bissextos permite ate o dia 29)  
Testando o operador de incremento prefixado:  
d4 eh July 13, 2002  
++d4 eh July 14, 2002  
d4 eh July 14, 2002  
Testando o operador de incremento posfixado:  
d4 eh July 14, 2002  
d4++ eh July 14, 2002  
d4 eh July 15, 2002  
Pressione qualquer tecla para continuar. . . _
```

```
cout << " d4 eh " << d4;
```

```
cout << "\n\nTestando o operador de incremento posfixado:\n"  
    << " d4 eh " << d4 << endl;  
cout << "d4++ eh " << d4++ << endl;  
cout << " d4 eh " << d4 << endl;
```

```
return 0;
```

```
}
```

Classe `string` da Biblioteca-padrão

- Classe predefinida no C++
 - Disponível para uso por qualquer pessoa
 - Classe `string`
 - Semelhante à classe `String` desenvolvida
- Refaça a classe `String` usando `string`

Classe `string` da Biblioteca-padrão

- Classe `string`
 - Cabeçalho `<string>`, namespace `std`
 - Pode inicializar `string s1("hi");`
 - `<<` sobrecarregado (como em `cout << s1`)
 - Operadores relacionais sobrecarregados `==, !=, >=, >, <=, <`
 - Operador de atribuição `=`
 - Concatenação (`+=` sobrecarregado)

Classe `string` da Biblioteca-padrão

- Classe `string`
 - Função-membro de substring `substr`
 - `s1.substr(0, 14);`
 - Inicia na posição 0, obtém 14 caracteres
 - `s1.substr(15);`
 - Início da substring na posição 15, até o fim
 - `[]` sobrecarregado
 - **Acessa um caractere**
 - **Não há verificação de intervalo (se o subscrito for inválido)**

Classe `string` da Biblioteca-padrão

- Classe `string`
 - Função-membro `at`
 - **Acessa um caractere**
 - Ex.: `s1.at(10);`
 - **Tem verificação de limites. Lança uma exceção se o subscrito for inválido**
 - Terminará o programa

Quinto Ex. de Operadores Sobrecarregados em C++

```
/*
 * Aula 11 - Exemplo 5
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>
#include <string>

using namespace std;

int main() {
    string s1( "Feliz" );
    string s2( " aniversario" );
    string s3;

    // testa igualdade e relacionais sobrecarregados
    cout << "s1 eh \"" << s1 << "\"; s2 eh \"" << s2
        << "\"; s3 eh \"" << s3 << "\"\n"
        << "\n\nOs resultados da comparacao s2 e s1:"
        << "\ns2 == s1 permite " << ( s2 == s1 )
        << "\ns2 != s1 permite " << ( s2 != s1 )
        << "\ns2 > s1 permite " << ( s2 > s1 )
        << "\ns2 < s1 permite " << ( s2 < s1 )
        << "\ns2 >= s1 permite " << ( s2 >= s1 )
        << "\ns2 <= s1 permite " << ( s2 <= s1 );

    // teste função membro vazia de String
    cout << "\n\nTestando s3.empty():" << endl;

    if ( s3.empty() ) {
        cout << "s3 esta vazio; atribuindo s1 a s3;" << endl;
        s3 = s1;
        cout << "s3 is \"" << s3 << "\"\n";
    }
}
```

Quinto Ex. de Operadores Sobrecarregados em C++

```
// testa operador sobrecarregado de concatenação de string
cout << "\n\ns1 += s2 permite s1 = ";
s1 += s2;
cout << s1;

// testa operador sobrecarregado de concatenação com string do tipo C
cout << "\n\ns1 += \" para vc\" permite" << endl;
s1 += " para vc";
cout << "s1 = " << s1 << "\n\n";

// testa função membro de string substr
cout << "A substring de s1 começando na posicao 0 para\n"
      << "14 caracteres, s1.substr(0, 14), eh:\n"
      << s1.substr( 0, 14 ) << "\n\n";

// testa opção "to-end-of-string" substr
cout << "A substring de s1 começando na\n"
      << "posicao 15, s1.substr(15), eh:\n"
      << s1.substr( 15 ) << endl;

// testa construtor de cópia
string *s4Ptr = new string( s1 );
cout << "\n*s4Ptr = " << *s4Ptr << "\n\n";

// testa operador de atribuição (=) com auto-atribuição
cout << "atribuicao *s4Ptr para *s4Ptr" << endl;
*s4Ptr = *s4Ptr;
cout << "*s4Ptr = " << *s4Ptr << endl;

// testa destrutor
delete s4Ptr;
```

Quinto Ex. de Operadores Sobrecarregados em C++

```
// testa operador de subscripto para criar lvalue
s1[ 0 ] = 'H';
s1[ 6 ] = 'B';
cout << "\ns1 depois s1[0] = 'H' e s1[6] = 'B' eh: "
      << s1 << "\n\n";

// testa subscripto fora do intervalo com a função membro "at" de string
cout << "Tentativa de atribuir 'd' para s1.at( 30 ) permite:" << endl;
s1.at( 30 ) = 'd'; // ERROR: subscripto fora do intervalo

return 0;
}
```

Quinto Ex. de Operadores Sobrecarregados em C++

```
C:\Windows\system32\cmd.exe
C:\Users\Miguel>cd Documents\UFRJ\disciplinas\linguagens\projetos
C:\Users\Miguel\Documents\UFRJ\disciplinas\linguagens\projetos>aula11-ex5.exe
s1 eh "Feliz"; s2 eh " aniversario"; s3 eh ""

Os resultados da comparacao s2 e s1:
s2 == s1 permite 0
s2 != s1 permite 1
s2 > s1 permite 0
s2 < s1 permite 1
s2 >= s1 permite 0
s2 <= s1 permite 1

Testando s3.empty():
s3 esta vazio; atribuindo s1 a s3;
s3 is "Feliz"

s1 += s2 permite s1 = Feliz aniversario

s1 += " para vc" permite
s1 = Feliz aniversario para vc

A substring de s1 comecando na posicao 0 para
14 caracteres, s1.substr(0, 14), eh:
Feliz aniversa

A substring de s1 comecando na
posicao 15, s1.substr(15), eh:
io para vc

*s4Ptr = Feliz aniversario para vc

atribuicao *s4Ptr para *s4Ptr
*s4Ptr = Feliz aniversario para vc

s1 depois s1[0] = 'H' e s1[6] = 'B' eh: Heliz Bniversario para vc

Tentativa de atribuir 'd' para s1.at( 30 ) permite:

This application has requested the Runtime to terminate it in an unusual way.
Please contact the application's support team for more information.

C:\Users\Miguel\Documents\UFRJ\disciplinas\linguagens\projetos>
```

Construtores explicit

- Conversões implícitas
 - São realizadas pelo compilador por meio de construtores de um único argumento
 - Às vezes, as conversões implícitas são indesejáveis ou propensas a erros
 - Palavra-chave **explicit**
 - Exclua as conversões implícitas por meio dos construtores de conversão
- Infelizmente, o compilador pode usar conversões implícitas em casos inesperados
 - Resulta em expressões ambíguas que geram erros de compilação ou provocam erros de lógica em execução

Sexto Ex. de Operadores Sobrecarregados em C++

```
/*
 * Aula 11 - Exemplo 6
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include "arrayCap11Ex6.h"

void outputArray( const Array & ); // protótipo

int main() {
    Array integers1( 7 ); // array de 7 elementos
    outputArray( integers1 ); // exhibe Array integers1
    outputArray( 3 ); // converte 3 para um an Array e exhibe conteúdo do array

    return 0;
}

// imprime o conteúdo do array
void outputArray( const Array &arrayToOutput ) {
    cout << "O array recebido tem " << arrayToOutput.getSize()
         << " elementos. Os conteúdos são:\n" << arrayToOutput << endl;
}
```


Sexto Ex. de Operadores Sobrecarregados em C++

```
/*
 * Aula 11 - Exemplo 6
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include "arrayCap11Ex6.h"

void outputArray( const Array & ); // protótipo

int main() {
    Array integers1( 7 ); // array de 7 elementos
    outputArray( integers1 ); // exibe Array integers1
    outputArray( 3 ); // converte 3 para um an Array e exibe conteúdo do array

    return 0;
}

// imprime o conteúdo do array
void outputArray( const Array &arrayToOutput ) {
    cout << "O array recebido tem " << arrayToOutput.getSize()
    << " elementos. Os conteúdos são:" << arrayToOutput << endl;
}
```

Compilador assume que deve ser feita uma conversão implícita de int para array e tenta usar o construtor. Entretanto, não há nenhuma função outputArray (int)

```
/*
 * Aula 11 - Exemplo 6
 * Arquivo arrayCap11Ex6.h
 * Autor: Miguel Campista
 */
#ifndef ARRAY_H
#define ARRAY_H

#include <iostream>
#include <iomanip>
#include <new>
#include <cstdlib>

using namespace std;

class Array {
    friend ostream &operator<<( ostream &, const Array & );
    friend istream &operator>>( istream &, Array & );

public:
    explicit Array( int = 10 );
    Array( const Array & );
    ~Array();
    int getSize() const;

    const Array &operator=( const Array & );
    bool operator==( const Array & ) const;

    bool operator!=( const Array &right ) const {
        return ! ( *this == right );
    }

    int &operator[]( int );

    const int &operator[]( int ) const;
```

```

/*
 * Aula 11 - Exemplo 6
 * Arquivo arrayCap11Ex6.h
 * Autor: Miguel Campista
 */
#ifndef ARRAY_H
#define ARRAY_H

#include <iostream>
#include <iomanip>
#include <new>
#include <cstdlib>

using namespace std;

class Array {
    friend ostream &operator<<( ostream &, const Array & );
    friend istream &operator>>( istream &, Array & );

public:
    explicit Array( int = 10 );
    Array( const Array & );
    ~Array();
    int getSize() const;

    const Array &operator=( const Array & );
    bool operator==( const Array & ) const;

    bool operator!=( const Array &right ) const {
        return ! ( *this == right );
    }

    int &operator[]( int );

    const int &operator[]( int ) const;

```

Só permite conversões explícitas

Sexto Ex. de Operadores Sobrecarregados em C++

```
private:  
    int size;  
    int *ptr;  
};  
  
#endif
```

Sexto Ex. de Operadores Sobrecarregados em C++

```
/*
 * Aula 11 - Exemplo 6
 * Arquivo arrayCap11Ex6.cpp
 * Autor: Miguel Campista
 */
#include "arrayCap11Ex6.h"

Array::Array( int arraySize ) {
    size = ( arraySize > 0 ? arraySize : 10 );
    ptr = new int[ size ];

    for ( int i = 0; i < size; i++ )
        ptr[ i ] = 0;
}

Array::Array( const Array &arrayToCopy ) : size( arrayToCopy.size ) {
    ptr = new int[ size ];

    for ( int i = 0; i < size; i++ )
        ptr[ i ] = arrayToCopy.ptr[ i ];
}

Array::~Array() {
    delete [] ptr;
}

int Array::getSize() const {
    return size;
}
```

```

const Array &Array::operator=( const Array &right ) {
    if ( &right != this ) {
        if ( size != right.size ) {
            delete [] ptr;
            size = right.size;
            ptr = new int[ size ];
        }

        for ( int i = 0; i < size; i++ )
            ptr[ i ] = right.ptr[ i ];
    }

    return *this;
}

bool Array::operator==( const Array &right ) const {
    if ( size != right.size )
        return false;

    for ( int i = 0; i < size; i++ )
        if ( ptr[ i ] != right.ptr[ i ] )
            return false;
    return true;
}

int &Array::operator[]( int subscript ) {
    if ( subscript < 0 || subscript >= size ) {
        cout << "\nErro: Subscrito " << subscript
            << " fora do intervalo" << endl;
        exit( 1 );
    }

    return ptr[ subscript ];
}

```

Sexto Ex. de Operadores Sobrecarregados em C++

```
const int &Array::operator[]( int subscript ) const {
    if ( subscript < 0 || subscript >= size ) {
        cout << "\nErro: Subscrito " << subscript
            << " fora do intervalo" << endl;
        exit( 1 );
    }

    return ptr[ subscript ];
}

istream &operator>>( istream &input, Array &a ) {
    for ( int i = 0; i < a.size; i++ )
        input >> a.ptr[ i ];

    return input;
}

ostream &operator<<( ostream &output, const Array &a ) {
    int i;

    for ( i = 0; i < a.size; i++ ) {
        output << setw( 12 ) << a.ptr[ i ];

        if ( ( i + 1 ) % 4 == 0 )
            output << endl;
    }

    if ( i % 4 != 0 )
        output << endl;

    return output;
}
```

Sexto Ex. de Operadores Sobrecarregados em C++

```
const int &Array::operator[]( int subscript ) const {  
    if ( subscript < 0 || subscript >= size ) {  
        cout << "\nErro: Subscrito " << subscript  
            << " fora do intervalo" << endl;  
        exit( 1 );  
    }  
  
    return ptr[ subscript ];  
}
```

Line	File	Message
	C:\Users\Miguel\Documents\UFRJ\...	In function 'int main()':
13	C:\Users\Miguel\Documents\UFRJ\...	invalid initialization of reference of type 'const Array&' from expression of type 'int'
8	C:\Users\Miguel\Documents\UFRJ\...	in passing argument 1 of 'void outputArray(const Array&)'
	C:\Users\Miguel\Documents\UFRJ\...	[Build Error] exe: **** [aula11-ex6.o] Error 1

```
ostream &operator<<( ostream &output, const Array &a ) {  
    int i;  
  
    for ( i = 0; i < a.size; i++ ) {  
        output << setw( 12 ) << a.ptr[ i ];  
  
        if ( ( i + 1 ) % 4 == 0 )  
            output << endl;  
    }  
  
    if ( i % 4 != 0 )  
        output << endl;  
  
    return output;  
}
```


Construtor explicit

- Caso o construtor não fosse explicit...
 - O compilador assumiria que a chamada `outputArray(const Array &a);` → `outputArray(3)`
 - Seria uma chamada a um construtor e tentaria converter int para array
 - O array a seria inicializado com 3 elementos

Sexto Ex. de Operadores Sobrecarregados em C++

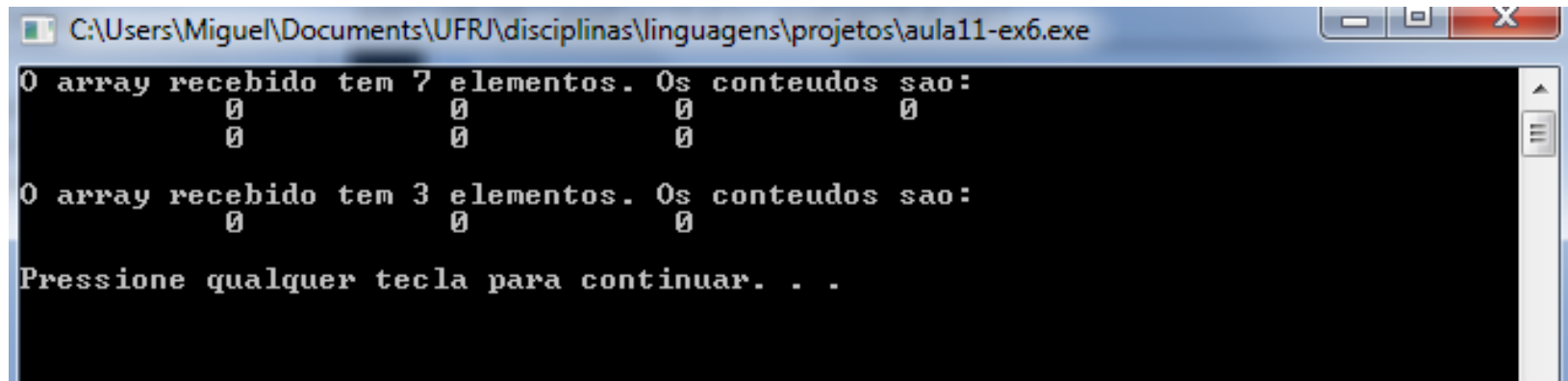
```
const int &Array::operator[]( int subscript ) const {  
    if ( subscript < 0 || subscript >= size ) {
```

Caso o explicit não fosse usado...

```
}
```

```
    return ptr[ subscript ];
```

```
}
```



```
C:\Users\Miguel\Documents\UFRJ\disciplinas\linguagens\projetos\aula11-ex6.exe  
0 array recebido tem 7 elementos. Os conteudos sao:  
    0      0      0      0      0      0      0  
0 array recebido tem 3 elementos. Os conteudos sao:  
    0      0      0  
Pressione qualquer tecla para continuar. . .
```

```
        if ( ( i + 1 ) % 4 == 0 )  
            output << endl;
```

```
    }
```

```
    if ( i % 4 != 0 )  
        output << endl;
```

```
    return output;
```

```
}
```

Construtor explicit

- Erros de compilação
 - Tentar invocar um construtor `explicit` para uma conversão implícita
 - Utilizar a palavra-chave `explicit` em membros de dados ou funções-membro diferentes de um construtor de um único argumento
- Utilize a palavra-chave `explicit` em construtores de um único argumento que não devem ser utilizados pelo compilador para realizar conversões implícitas

Leitura Recomendada

- Capítulos 11 do livro
 - Deitel, "*C++ How to Program*", 5th edition, Editora Prentice Hall, 2005